Feature-Based Fast Grasping for Unknown Objects

Qujiang LEI

Feature-Based Fast Grasping for Unknown Objects

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. dr. ir. T.H.J.J. van der Hagen,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op
woensdag 7 maart 2018 om 12:30 uur

door

Qujiang LEI

Master of Science in Mechanical Design and Theory
Chongqing University, China
geboren te Sichuan, China

The research presented in this thesis was supported by TU Delft/CSC doctoral programme.

Keywords: object feature, gripper feature, unknown object grasping, fast grasping, partial point cloud, force balance

Author email: leiqujiang@126.com

An electronic version of this dissertation is available at
http://repository.tudelft.nl

To my family

Contents

Contents VII

Summary XI

1 Introduction 1

1.1 Motivation 3

1.2 Thesis goals 8

1.3 Proposed approaches 9

1.4 Thesis structure 10

2 A survey of unknown object grasping 13

2.1 Introduction 14

2.2 Survey about unknown object grasping 15

2.2.1 Global grasping approaches 15

2.2.2 Local grasping approaches 17

2.3 Comparison 19

2.3.1 Comparison table 19

2.3.2 Comparison discussion 21

2.4 Conclusion 22

3 Fast grasping of unknown objects using principal component analysis 23

3.1 Introduction 24

3.2 A fast grasping approach 25

3.2.1 Algorithm 25

3.2.2 Grasping unknown object based on the single-view partial point cloud 26

3.2.3 Solution for the exceptional case 33

3.3 Simulation test 35

3.3.1 Structure of simulation setup 35

3.3.2 Simulations based on a single-view partial point cloud 35

3.4 Experimental validation 37

3.4.1 Experimental description 37

3.4.2 Results 38

3.5 Minimizing grasping uncertainty by using two 3D cameras 39

3.5.1 Building a “big” partial point cloud using two 3D cameras.......................... 39
3.5.2 Grasp allocation between the two camera sensor points.............................. 41
3.5.3 Constructing virtual object coordinate systems.. 42
3.5.4 Grasp allocation for a virtual viewpoint... 43
3.5.5 Force balance computation... 44
3.5.6 Simulation for using two 3D sensors 46
3.6 Conclusion.. 49

4 Object grasping by combining caging and force closure............................... 51
4.1 Motivation 52
4.2 Introduction 53
4.3 Detailed algorithm.. 54
4.3.1 Grasp candidates generation... 55
4.3.2 Force closure analysis 61
4.3.3 Gravity analysis.. 64
4.4 Simulation 65
4.5 Experiment... 67
4.6 Comparison.. 67
4.7 Conclusion.. 69

5 Unknown object grasping by using concavity 71
5.1 Introduction 72
5.2 Detailed algorithm.. 75
5.2.1 Concavity calculation... 75
5.2.2 Concavity extraction... 76
5.2.3 Construct the concavity coordinate system.. 77
5.2.4 Analyze concavity and generate grasp candidates 78
5.2.5 Generate grasp candidates.. 81
5.2.6 Force balance computation and manipulability analysis.............................. 84
5.3 Simulation 86
5.4 Conclusion.. 88

6 Fast grasping of unknown objects using C-shape configuration.................. 89
6.1 Introduction 90
6.2 Fast C-shape grasping for unknown objects....................................... 92

6.2.1 Mathematical description of the C-shape configuration 92
6.2.2 Outline of our fast grasping algorithm 94
6.3 Selection of motion planners for grasping execution 102
6.3.1 Motion planning using MoveIt! 103
6.3.2 Overview of OMPL planners available in MoveIt!.................................... 103
6.3.3 Methodologies of comparing motion planners in MoveIt! 106
6.3.4 Parameter selection... 107
6.3.5 Comparison results.. 108
6.4 Simulation 110
6.5 Experiments 111
6.6 Discussion.. 112
6.7 Conclusion... 113
7 Conclusions, discussions and future directions... 115
7.1 Conclusions 116
7.2 Discussions 118
7.3 Future directions 123
References 125
Appendix 141
Acknowledgements... 159
About the author 161
List of publications.. 163

Summary

According to the report by the United Nations in 2015, the global population of older persons aged 60 years or over is predicted to grow to 1.4 billion by 2030. A rapidly aging population poses a challenging problem for human beings, i.e. supply shortage of working-age people. To solve this problem, increasing research efforts are poured into the field of robotics, especially in service robotics. Service robots are believed to be a solid solution to the challenging problem of an aging population. The Strategic Research Agenda (SRA) for Robotics in Europe, a development guideline for European robotics from 2014 to 2020, classifies robots' functions into eight basic categories, i.e., assembly, surface process, interaction, exploration, transporting, inspection, grasping and manipulation. From SRA, we can find that grasping is an important basic function for robots. Combining grasping with other basic functions, robots can perform many service tasks to free humans from tedious housework, for example, cleaning rooms, cooking and washing dishes.

According to the existing literature, grasping approaches of objects can be classified into three categories: known object grasping, familiar object grasping and unknown object grasping. Grasping of unknown objects with neither appearance data nor object models given in advance is a challenging task for service robots that work in an unfamiliar environment. This thesis focuses on the challenging problem of unknown object grasping for service robots. According to analysis of existing literature, the challenging problem of unknown object grasping can be divided into four subquestions, i.e. how to increase grasp speed, how to enhance grasp stability, how to raise grasp security and how to increase grasp generality. These four subquestions are ranked according to the number of corresponding literature. Most literature concerns how to increase grasp speed, and then it is how to enhance grasp stability, followed by how to raise grasp security and how to increase grasp generality. To enable service robot as agile as possible, the overall goal of this thesis is to design a fast, stable, secure and general grasping algorithm for unknown objects to answer above four subquestions to thus solve the challenging problem of unknown object grasping.

To answer the subquestion of how to increase grasp speed, this thesis proposes to employ the features (features of target objects and features of grippers) to accelerate grasp searching process. Grasp configurations in 3D space means countless possibilities. To reduce useless grasp candidates, object features including principal axis, boundary and concavity are utilized to accelerate grasp searching. As to the subquestion of how to enhance grasp stability, the optimized approximate force closure grasp is returned as final grasp to ensure the grasp stability. The geometric shape of the two grasp sides are fit into two straight lines, and the angle between the two straight lines is used to evaluate force closure quality of a grasp. In such a way, the optimized grasp with best approximate force closure is chosen as final grasp to enhance grasp stability. For the subquestion of how to raise grasp security, we propose

two methods to deal with occlusions resulted in by using partial point cloud. The first method is to constrain grasp configurations on the seen part of the target object. The second method is to add manmade obstacles for the target object. Using the two methods, the robot can avoid unexpected contact with target object to thus raise grasp security. As to the subquestion of how to increase grasp generality, we propose to simplify the gripper into a C-shape, which is used to match with the partial point cloud of the target object to find suitable grasps. All grippers including parallel grippers, under-actuated grippers and dexterous hands can be simplified as C-shapes, therefore, the algorithms based on C-shape can be widely used by various grippers to thus increase grasp generality.

To achieve the overall goal aforementioned, the four subquestions need to be answered. However, it is significantly difficult to design a grasping algorithm that can answer all of the four subquestions. Therefore, Chapter 3 to Chapter 5 shows three grasp algorithms that can solve part of the four subquestions. In Chapter 6, a grasping algorithm that can answer all of the four subquestions is presented.

Chapter 3 uses the principal axis of a single-view partial point cloud to direct the grasp configurations. Grasp candidates are allocated along the principal axis such that the possibility of useless grasp candidates can be greatly decreased. Approximation of force balance on the two grasp sides is used to evaluate the quality of a grasp. The stable grasp with the best force balance is chosen as the final grasp. To minimize grasping uncertainty resulted in by occlusions, robots with two 3D cameras are utilized to help to construct a “big” partial point cloud. Then grasp candidates are constrained on the seen part of the object to ensure the security of the final grasp. Overall, the designed grasping algorithm in Chapter 3 can fast achieve stable and secure grasp on a single-view partial point cloud within one second. However, we did not consider grasp generality among different grippers in this chapter.

Chapter 4 utilizes the boundary of the target object to guide the grasp configurations to accelerate the grasp searching process. The boundary is obtained using the oriented bounding box of the partial point cloud of the target object. Inspired by the idea that caging grasping that generates finger points along the object’s boundary, we allocate finger candidates along the boundary of the object. Differing from caging grasping, we did not simplify the robot finger as a point. On the contrary, we considered the geometric property of the grippers to achieve more stable grasps than caging. After finger candidates are allocated along the object’s boundary, any two of the finger candidates can form a grasp candidate, which is analyzed by using approximate force closure to choose the best grasp to execute. Meanwhile, grasp stability during manipulation of the object is guaranteed by considering the gravity of the object. To sum up, Chapter 4 presents a fast and stable grasping algorithm that can quickly work out stable grasps for the target unknown object within one second, however, we did not consider security and generality in this chapter.

Chapter 5 employs the concavity of the target object to achieve a fast grasp. Shortest path concavity is employed to work out the concavity value for every vertex of the unknown object followed by concavity extraction to obtain the most salient concave areas. Grasp candidates are generated at the most salient concave areas and evaluated by using force balance computation. Grasp candidates are ranked according to the results of force balance computation and the manipulability of every grasp candidate. The grasp with the best force balance and manipulability is chosen as the final grasp. In summary, Chapter 5 presents a fast and stable grasping algorithm for unknown objects. However, we did not consider generality in this chapter.

Differing from the previous three grasping approaches, Chapter 6 starts from the feature of the grippers. The geometric shapes of the grippers are approximated as a C-shape, which is used to fit the single-view partial point cloud of the target unknown object along the normal lines to find a suitable grasp. The number of grasp candidates is significantly reduced by using the normal lines to direct configuration of grasp candidates. Then a random searching process is utilized to quickly locate suitable grasps for the target object. Meanwhile, local geometry analysis and force balance analysis are utilized to ensure the stability of the final grasp. To eliminate the occlusion uncertainty resulted in by using a partial point cloud, manmade obstacles are added to the single-view partial point cloud to avoid unexpected contacts to thus enhance grasp security. More importantly, the grasping algorithm in Chapter 6 does not rely on object features so that this grasping algorithm can be widely used by various grippers. Overall, Chapter 6 presents a fast and general grasping algorithm for unknown objects that can quickly work out stable and secure grasp on a single-view partial point cloud within one second.

Overall, simulations and experiments of the grasping approaches presented in this thesis show significant improvements of time efficiency, stability, security and generality over the existing grasping approaches in the literature. We believe that the presented approaches can have significant contribution for solving the challenging problem of unknown object grasping.

1

Introduction

In 2015, according to the report [1] by the United Nations, the global population of older persons aged 60 years or over is 901 million, accounting for about one in eight people. By 2030, the aged population is predicted to grow by 56 percent to 1.4 billion [2], which means around one in every six people will be aged 60 years or over. By the middle of this century, the aged population will double its size in 2015 to reach almost 2.1 billion [3], approximately one old in every five people. A rapidly aging population poses a challenging problem for all countries in the world. Fewer working-age people result in supply shortage of qualified labors for our society. Therefore, fewer people can be available to take care of old citizens. The ageing process first appears in high-income countries. By the end of 2015, Japan had more aged population than any other countries in the world (around 33% of Japan's population was aged 60 years or over). Japan was closely followed by Germany (28%), Italy (28%) and Finland (27%) [1]. Supply shortage of working-age people in high-income countries leads to the rapid advancement of the service robot technology. These high-income countries with aged population have both demand and financial ability for service robots. For the challenging problem of the globaly-aged population, it is believed that service robots will be a solid solution.

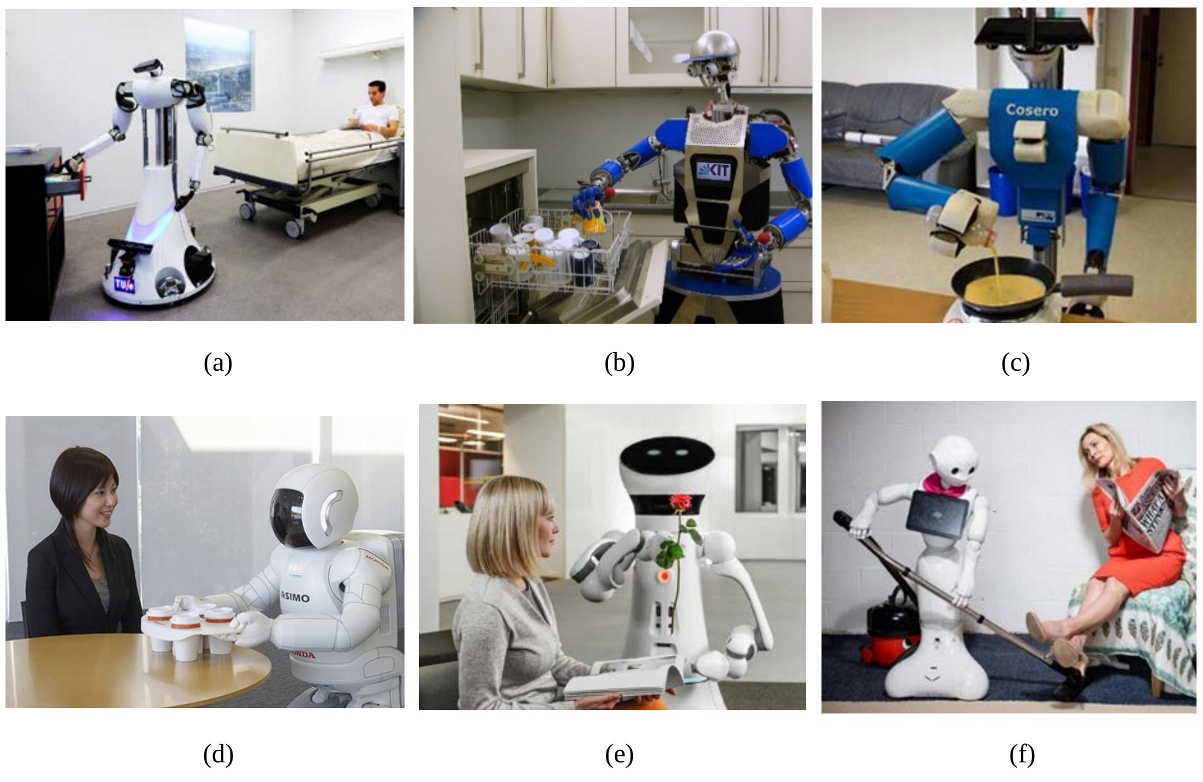


Figure 1.1: Several well-known service robots: the first row shows three famous service robots from research institutes; the second row shows three brilliant service robots from companies. (a) Amigo [4] by Eindhoven University of Technology, (b) Armar III [5] by Karlsruhe Institute of Technology, (c) Cosero [6] by University of Bonn, (d) Asimo [7] by Honda. (e) Care-o-bot 4 [8] by Fraunhofer IPA, (f) Pepper [9] by Aldebaran Robotics.

Figure 1.1 presents the most dominant service robots that can perform complex service tasks, for example, cleaning the room, cooking, serving coffee, washing dishes, etc. Before service robots can agilely work as human servants in our homes, offices and shopping malls, there is a long scientific way to go. Many key problems are waiting for us to solve. In order to enable a service robot to be as agile as humans, many fundamental crucial functions are necessary for service robots. The Strategic Research Agenda for Robotics in Europe [10] classifies robots' functions into eight basic categories, i.e., assembly [11-13], surface process [14-16], interaction [17-19], exploration [20-22], transporting [23-25], inspection [26-28], grasping [29-31] and manipulation [32-34]. Grasping is an important basic function for robots. Combining grasping with other basic functions, robots can perform many complex service tasks to free humans from tedious housework, for example, cleaning rooms, cooking and washing dishes. All service robots shown in Figure 1.1 are capable of an essential function, that is object grasping.

Existing approaches of object grasping can be classified into three categories: known object grasping, familiar object grasping and unknown object grasping [35]. The concepts of "known object", "familiar object" and "unknown object" are related to the amount of prior information of the target object. Known object grasping approaches [36-47] rely on the available prior information of the object to perform stable grasps. Familiar object grasping approaches [48-60] also rely on available prior object information. However, they are able to grasp an object when it is similar to the known ones. Unknown object grasping approaches [61-72] do not need any prior information of the object to perform grasps.

For the grasping problem of known and familiar objects, 3D models or 2D images of the target objects are stored in a database in advance. Using the geometry information of the 3D models or 2D images, the grasping problem of known objects and familiar objects is usually formulated into optimization problem of locations of grasping points or grasping regions, many grasping algorithms can provide excellent solutions to this kind of optimization problems. They can work out stable grasps in a very short amount of time. However, in our daily environments, it is impossible to create a database to store 3D models or 2D images for huge variety of objects. Therefore, grasping algorithms for unknown objects are necessary. Existing algorithms [74, 75, 78, 120] of unknown object grasping are usually slow, which may take from one minute to several hours [116] to form a suitable grasp. In order to enable service robots as agile as possible, fast grasping algorithms for unknown objects are in crucial demand. Therefore, this thesis is focused on the design of fast grasping algorithms for unknown objects.

1.1 Motivation

Comparing with known object grasping and familiar object grasping, unknown object grasping is still a quite difficult problem because unknown objects widely exist in our daily environments that are usually unstructured and dynamic. An unknown object means an item that has neither geometric model nor appearance information. Grasping of unknown objects is highly challenging for service robots working at unfamiliar environments [73]. According to existing literature, the challenging problem of unknown object grasping can be divided into four subproblems, i.e. how to increase grasp speed, how to raise grasp security, how to enhance grasp stability and how to increase grasp generality. The first subproblem is actually that time efficient grasping algorithms of unknown objects are scarce. The second subproblem is lacking of efficient methods to deal with grasp uncertainty resulted in by using partial information of the target unknown object. The third subproblem is how to utilize the metrics of force balance on a partial model to quickly achieve a stable grasp. The last subproblem is lacking of cheap and general fast grasping algorithms for unknown objects. The motivation of the thesis is to find answers to the above four subproblems to thus solve the challenging problem of unknown object grasping.

Subproblem 1: How to improve the time efficiency of unknown object grasping?

From the perspective of the data used by existing grasping algorithms, there are mainly two methods to solve the problem of unknown object grasping. The first method is building a full 3D model using many images or point clouds of the target unknown object. The full 3D model is then used to compute suitable grasps for the target object. [74-79] are benchmark papers that employ full 3D model to work out proper grasps. The second method is directly utilizing partial information of an object to realize grasping [55, 80-82]. Comparing with suing full 3D model, utilizing partial information can significantly reduce computational load to thus accelerate grasp searching process for unknown objects.

Building 3D model is time-consuming and many robotic applications require real time grasping. In some cases, it is even impossible to get all necessary information to construct a full 3D model, for example, an object in the fridge, where the robot cannot see the other side of the target object. Meanwhile, many grasping algorithms require accurate 3D model, it means grasping algorithms may fail when the 3D model has some errors or noise. Overall, using partial information of unknown objects to achieve a grasp is usually faster and more practical than using full 3D models.

Normally, the fast grasping approaches employ geometric properties (e.g. symmetries [83], surface [77], edges [84], boundary [85], silhouette [86] and saliency [87]) of the target unknown object to accelerate the grasp searching process. This is because using geometric properties of unknown objects can determine the geometry contours. Using geometric properties can account for much information of the target object for constructing the geometry contour, which can significantly reduce computational load and thus accelerate the grasp searching process. Therefore, this thesis proposes to utilize both partial information and geometric features of unknown objects to accelerate the grasp searching process. Object features including principal axis (Chapter 3), boundary (Chapter 4) and concavity (Chapter 5) are employed to achieve fast grasping of unknown objects. In addition to object features, Chapter 6 utilizes geometric feature (C-shape) of grippers to accomplish a fast grasping approach for unknown objects.

Subproblem 2: How to deal with grasp uncertainties resulted in by occlusions?

Using partial information of the target unknown object is a double-edged sword, it can definitely accelerate the process of grasp searching, however, it also inevitably introduces occlusions that may lead to grasp uncertainty and result in grasp failure. In general, two methods are used to deal with the uncertainties introduced by using partial information, i.e. tactile sensor based exploration and vision based exploration.

The first method is to utilize tactile sensors to explore the unseen part of the target object, as tactile sensors enable direct sensing of aspects such as contact force or relative velocity at contact points, without being affected by the occlusions. [88-92] are benchmark papers that utilize partial object data and tactile feedback from fingers to achieve secure grasps for the target objects. These attempts can help to overcome the occlusions resulting from the uncompleted data of the unknown objects. Tactile sensors can help to modify the robot’s behavior when unexpected contact is made during the grasp execution or the fingertip contacts appear less stable than expected. However, tactile sensor based exploration requires a large amount of computation.

The other method to explore the unseen part of the target object is to use a robot arm carrying a camera to move around the target object to do active exploration. [93] utilizes a camera at the end of the robot arm to move around the target object to actively explore the unseen part of the object. The maximum curvature of Elliptic Fourier Descriptors silhouette is explored to work as the final grasp. [94] simplifies the shape of Barrett Hand as pre-shapes (spherical, cylindrical, box and disk). An eye in hand system with a 3D camera moves around the target object to explore it. Shape matching between pre-shapes of Barrett Hand and the point cloud of the target object is then carried out to find suitable grasps. Similarly, [95] utilizes a mobile robot to carry three range sensors to move around the target object to explore the unseen part. Then two parallel planes on the boundary of the object are selected out as final grasp.

Tactile sensors will send continuous feedback to control system to help to do reactive grasping planning which is fairly time consuming. A robot arm carrying a camera to move around the target object to do active exploration is also time expensive. These two methods take dozens of seconds to find a suitable grasp. Therefore, it leads to the necessity of new fast and secure approaches to deal with grasp uncertainty resulted in by using partial information of the target unknown object. This thesis proposes two methods to minizie grasp uncertainty resulted in by using partial information, i.e., virtual exploration on a “big” partial point cloud (Chapter 3) and manmade unseen parts for target unknown objects (Chapter 6).

Subproblem 3: How to quickly achieve a force closure grasp on a partial point cloud?

Force closure and form closure are the most common two methods to analyze the property of forces and motions of a grasp candidate. Force closure is widely utilized to analyze balance of forces and torques that the robot hand applies on the target object to achieve a stable grasp [96-101]. Form closure is another significant alternative way to attain a stable grasp by immobilizing the target object without depending on the contact surface friction [102-107]. Force closure grasps stand for that the object’s motion is restrained by suitable contact forces and torques on the base of considering contact constraints between the robot hand and the target object. The force closure grasp can resist any arbitrary forces and torques. Form closure grasp mean immobilizing a target object using several frictionless point contacts. Form closure is more difficult to achieve because it can be understood as force closure without considering friction [93].

GraspIt! [108] is the most renowned and prominent grasp simulation tool to achieve a force closure grasp. However, GraspIt! is not based on modular architecture, which makes it hard to improve, add functionality and integrate with other tools and frameworks. Therefore, OpenRAVE [109] (the Open Robotics and Animation Virtual Environment) is designed to work as an improved version of GraspIt!. OpenRAVE has a modular design, which allows extension and further development by other users. Both GraspIt! and OpenRAVE requires full 3D meshed model of the target object to work out a force closure grasp. However, it is hard to construct the full 3D model of the target unknown object and it is also difficult to know the physical properties of the target unknown object, for example, the friction coefficient.

In real environments, precise computation of force closure requires the full 3D model and the friction coefficient of the object surface. It is rather difficult to meet the two requirements for robots working in the unpredictable environments where grasping unknown objects is in demand. Therefore, approximation of force closure like [110, 95] becomes necessary, [110] utilize Hough transformation to gain the edges of objects in a 2D image. Two parallel edges suiting the gripper’s width are chosen to work as the final grasp. Similarly, [95] utilizes a mobile robot carrying three range sensors to move around the target object to construct a 3D model. The flat parallel surfaces are used to work as final grasp. [110] and [95] inspired us to think about a question, i.e., how to use such approximation of force closure on a partial mode (for example, a partial point cloud) of the target unknown object to achieve a fast grasp. Therefore, this thesis proposes to fit the two grasp sides of the target object into two straight lines and the angle between the two fit lines is used to approximately evaluate the force clousure quality of the grasp candidate. In such way, Chapter 3 to Chapter 6 can quickly achieve approximate force closure grasps for target unknown objects.

Subproblem 4: How to achieve a cheap and general grasping algorithm?

In the past, many searches focused on the problem of unknown object grasping using dexterous hands, for example, Shadow Hand [111], iCub Hand [112] and Barrett Hand [94]. Even though dexterous hands are very good at flexibility, the high complexity and high price stop them to become popular in the research field of fast grasping of unknown objects.

For the existing fast grasping algorithm [84, 110, 113, 114], all of them are specially designed for parallel grippers, which is much cheaper than dexterous hands. However, these fast grasping algorithms are not general enough. Specifically, [84] and [110] try to find two parallel edges to work as final grasp; [113] tries to fit the shape of the parallel gripper on the point cloud of the objects to obtain a grasp for robots; [114] grasps the gravity center along the principal axis of the target unknown object. All these fast grasping algorithms did great contributions to solve the problem of fast grasping of unknown objects. However, they have inevitable shortcomings. [84] and [110] rely on two parallel edges ignoring those objects without parallel edges, for instance, balls. [113] uses the shape of parallel gripper to match with point cloud of the objects ignoring the local geometry property and force balance of the grasp candidate. [114] grasps the object at the gravity center ignoring that many object cannot be grasped by gravity center, for example, the table tennis racket.

The above two facts lead us to think of a question, i.e., can we find a cheap and general solution to the problem of fast grasping of unknown objects. The solution should be based on cheap grippers and not rely on geometric properties of the target objects. To achieve cheap and general grasping of unknown objects, Chapter 6 proposes to utilize geometric feature (C-shape) of under-actuated grippers to achieve cheap and general grasping for unknown objects, which does not depend on object features.

The above four subproblems bring the necessity of new grasping algorithms for unknown objects by considering time efficiency (subproblem 1), grasp uncertainty (subproblem 2), force balance (subproblem 3) and generality (subproblem 4). Therefore, the overall goal of this thesis is to design new grasping algorithms for unknown objects and these new grasping algorithms should have high time efficiency, low grasp uncertainty, superb force balance and wide generality.

1.2 Thesis goals

This thesis aims to create fast, secure, stable and general grasping algorithms of unknown objects for the cheap grippers shown in Figure 1.2. Fast means reducing computing time of the searching process of grasping unknown objects. Secure means the unseen part of target unknown objects can be safely handled to avoid grasp failure. Stable means the force balance needs to be considered under the situation of using partial data of the target unknown object to obtain suitable grasps. General means the created algorithms do not rely on the geometric features of the target unknown objects so that they can be widely used. Three subgoals and one general goal are set as follows:

Subgoal 1: Improve the time efficiency for unknown object grasping

Subgoal 2: Enhance the grasping security of using partial point cloud

Subgoal 3: Ensure the grasp stability when friction coefficient is unknown

General goal: Create a general fast grasping algorithm

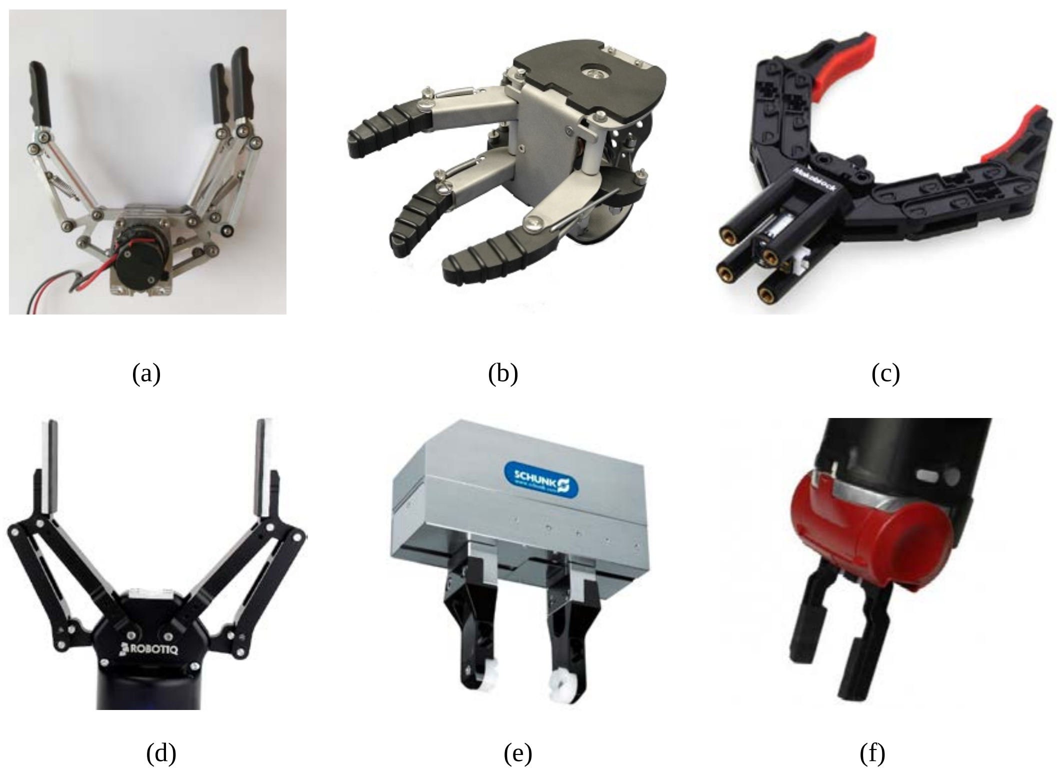


Figure 1.2: Some cheap grippers (comparing with dexterous hands): (a) under-actuated gripper by Delft University of Technology; (b) under-actuated gripper by Lacquey Company; (c), (d), (e) and (f) are parallel grippers respectively from Makeblock, ROBOTIQ, SCHUNK and Rethink Robotics.

1.3 Proposed approaches

The core idea of the proposed approaches in this thesis is to employ the features (features of target objects and features of grippers) to achieve fast grasping for unknown objects. The fast grasping algorithms based on object features are similar to [83-86], but faster than them. The fast grasping algorithm based on grippers' features is designed to achieve a general grasping algorithm without relying on objects' features. Object features are significantly useful clues for grasp finding. In this work, object features (principal axis, boundary and concavity) are utilized to assist grasp finding. In order to enable robots to grasp various unknown objects, we employ the geometric feature of cheap grippers of Figure 1.2 to design a general grasping algorithm without depending on object features.

Grasp configurations in 3D space means many possibilities. To reduce the possibilities to accelerate grasp searching, the principal axis of a single-view partial point cloud is used to direct the grasp configurations. Our first fast grasping approach is to allocate grasp candidates along the principal axis such that the possibility of useless grasp candidates can be greatly decreased. Approximation of force balance on the two grasp sides is used to evaluate the quality of a grasp. The grasp with the best force balance is chosen as the final grasp. To minimize the grasping uncertainty, the merits of the robot hardware with two 3D cameras can be utilized to suffice the partial point cloud. After that, virtual exploration is carried out on the "big" partial point cloud. Graspable candidates are allocated between the two camera sensor points, which can ensure the grasp candidates are allocated on the seen part of the target object. In such a way, grasp security is enhanced.

The second fast grasping approach for unknown objects is to utilize the boundary of the target object to quickly synthesize a grasp. Inspired by the idea that caging grasping generates finger points along the object's boundary, we also allocate finger candidates along the boundary of the object. However, differing from caging grasping, we did not simplify robot fingers as points. On the contrary, we considered the geometric property of the grippers. After a discrete set of finger candidates are allocated along the object's boundary, any two of the finger candidates can form a grasp candidate, which is analyzed by using force closure to choose the best grasp candidate as the final grasp execution. The grasp quality during the manipulation of the object is guaranteed by considering the gravity of the object.

The third approach of achieving a fast grasp is to utilize the concavity feature of the target objects. Shortest path concavity is first employed to work out the concavity value for every vertex of the unknown objects followed by concavity extraction to obtain the most salient concave areas. Grasp candidates are generated at the most salient concave areas and evaluated by using force balance computation. Grasp candidates are ranked according to the results of force balance computation and the manipulability of every grasp candidate. The grasp with the best force balance and manipulability is chosen as the final grasp.

The fourth grasping approach is differing from the previous three grasping approaches. We start from the feature of the grippers shown in Figure 1.2. The geometric shapes of the grippers are approximated as a C-shape, which is used to fit the single-view partial point cloud of the target unknown object along the normal lines to find a suitable grasp. The number of grasp candidates is greatly reduced by using the normal lines to direct the configuration of grasp candidates. A novel method is designed to eliminate the occlusion uncertainty resulted in by using a sing-view partial point cloud to achieve a secure grasp. Meanwhile, local geometry analysis and force balance analysis are utilized to ensure the stability of the final grasp.

1.4 Thesis structure

The structure of this thesis is visualized in Figure 1.3. In Chapter 2, a comprehensive survey about unknown object grasping is presented. Existing literatures about unknown object grasping are classified and compared. Chapter 3, 4, and 5 present three fast grasping algorithms based on using the features of the target unknown objects. Specifically, Chapter 3 shows the algorithm of utilizing the principal axis of the unknown object to achieve a fast, stable and secure grasp. Chapter 3 addresses the aforementioned subproblem 1, 2 and 3; Chapter 4 elaborates the algorithm of using boundary of the target object to accomplish a fast and stable grasp. Chapter 4 handles the subproblem 1 and 3; Chapter 5 demonstrates the algorithm of employing the concavity of the target object to realize a fast and stable grasp. Chapter 5 deales with the subproblem 1 and 3. In Chapter 6, an elaborate fast grasping approach using the feature of grippers is presented. All of the grippers in Figure 1.2 can be simplified as a C-shape, which is used to fit the single-view partial point cloud of the target unknown object to achieve a fast, stable, secure and general grasp. Chapter 6 provides a solution to subproblem 1, 2, 3 and 4. Chapter 7 finalizes this thesis with conclusions, discussions and future directions.

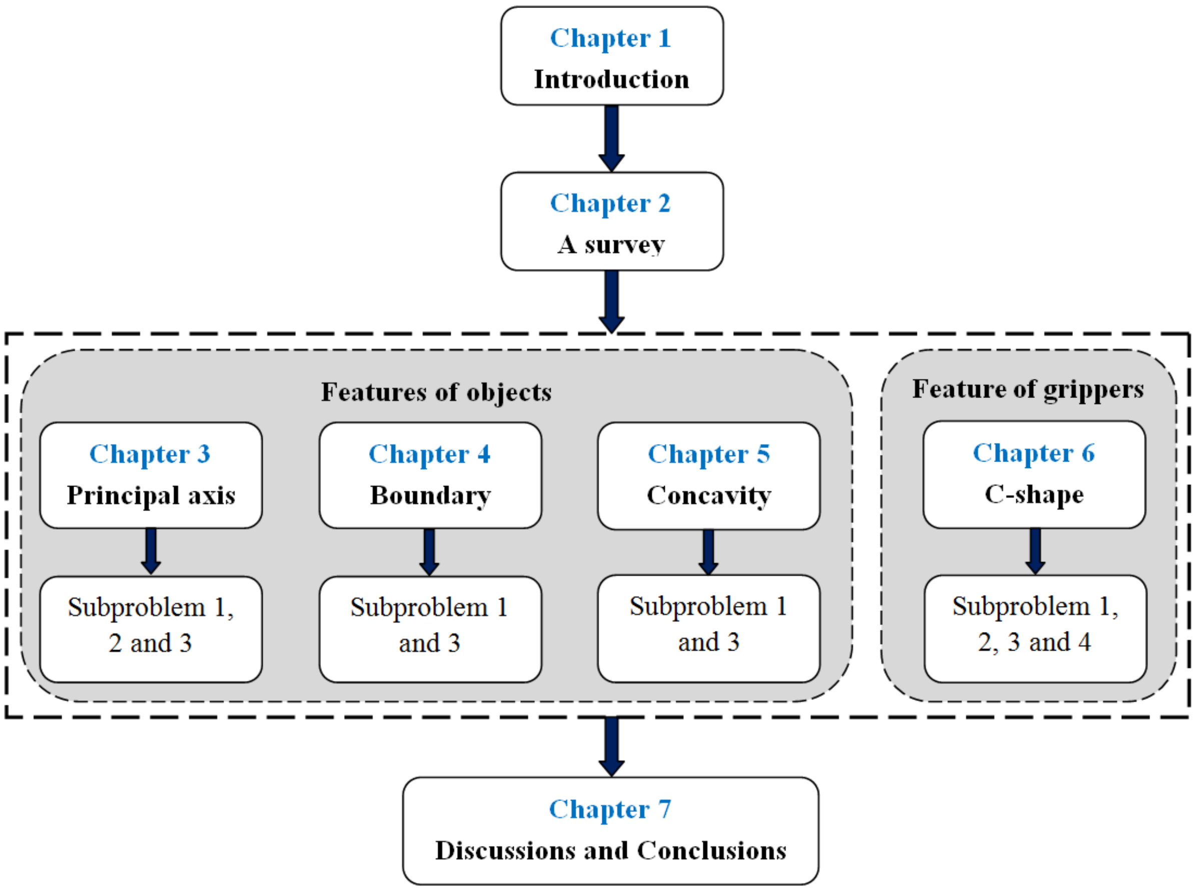


Figure 1.3: Visual outline of this thesis.

2

A survey of unknown object grasping

This Chapter was published at the 2017 ICCAR conference:

Qujiang Lei, Jonathan Meijer, Martijn Wisse. 2017 IEEE 3rd International Conference on Control, Automation and Robotics (ICCAR), pp. 150-157, Nagoya, Japan.

Abstract

Grasping of unknown objects with neither appearance data nor object models given in advance is very important for robots that work in an unfamiliar environment. In recent years, extensive researches have been conducted in the domain of unknown object grasping and many successful grasping algorithms for unknown objects are created. However, so far there is not a very general fast grasping algorithm that suits various kinds of unknown objects. Therefore, choice among different grasping algorithms becomes necessary for users. In order to make it more convenient for users to quickly understand and choose a suitable grasping algorithm, a survey about the latest research results of unknown object grasping is made in this chapter. We compared different grasping algorithms with each other and obtained a table to clearly show the result of comparison. The comparison could give researchers meaningful information in order to quickly pick a grasping approach with their requirements.

2.1 Introduction

In 2015, the number of professional service robots sold increased by 25% than that in 2014. It has been forecasted that this increase will continue for the upcoming years [115]. To help people with household tasks, grasping and manipulation are key functions for service robots. However, finding a suitable grasp is a complex task. Grasping approaches are designed to find meaningful grasp on a target object. However, due to the amount of researches of the past decade in this field, there is an abundance of different grasping approaches.

As explained by Bohg et al. [116], empirical grasping methodologies rely on sampling grasp candidates for an object and ranking these candidates with the use of a metric. In the study of Bohg et al. [35], the empirical grasping methodologies are divided into three categories: known, familiar and unknown object grasping approaches. Known object grasping approaches rely on the available information of the object to perform stable grasps. Familiar object grasping approaches also rely on available object information. However, they are able to grasp an object when the object is similar to the known ones. Unknown object grasping approaches do not need any prior information of the object to perform grasps.

In human environments, a great variety of different kinds of objects exist. Providing detailed information about all these objects would be a time-consuming task. The use of familiar object grasping approaches could help simplifying the aforementioned task. However, if these approaches pick a wrong similar object, grasps can become unreliable or imprecise. Since unknown object grasping approaches do not rely on available information, they are suitable to grasp a great variety of objects.

The survey by Bohg et al. [35] already focuses on the use of different unknown object grasping approaches. However, this survey did not compare the different approaches. Moreover, after the publication of this survey, more grasping approaches have been developed. For the survey part in this chapter, we aim to give an updated overview on the existing unknown object grasping approaches and provide a simple comparison. This will be done by collecting meaningful data found in the corresponding literatures, for example success rate and execution time.

In this chapter, we divide the existing unknown object grasping approaches into two groups, namely global and local grasping approaches. Global grasping approaches try to represent the full 3D model of the unknown object to find suitable grasps, which can be done by recreating the model with the use of multiple views of the object, symmetries, decomposition into 3D shapes or by closing the surface area of the retrieved data. Local grasping approaches only use the data available to work out suitable grasps, which use information in particular like edges, boundaries or silhouettes of the unknown object.

In above paragraphs, we explained what is unknown object grasping and why we do the survey about the existing approaches of unknown object grasping.

2.2 Survey about unknown object grasping

Grasping of unknown objects can be done in a variety of ways. In this section, the existing grasping approaches are classified and shortly explained.

Existing unknown object grasping approaches can be categorized into two groups: global grasping approaches and local grasping approaches. Global grasping approaches consider the whole object in order to find the best grasp. Local grasping approaches only work with partial data of the object to find a suitable grasp.

To segment the unknown object from the scene, grasping approaches usually only consider objects placed on flat surfaces. In a point cloud representation of the scene, a RANSAC (Random Sample Consensus) can help to distinguish flat surfaces. Isolating a point cloud cluster that represents the unknown object is done by removing all the points on the found flat surface.

2.2.1 Global grasping approaches

A. Multiple views

A way to consider the whole object is to look at the unknown object from multiple locations. From these locations, either 2D or 3D data can be retrieved in order to get accurate information of the model to successfully grasp the object.

In the work by Bone et al. [75], 2D images and structured-light data from multiple views are being used to create a 3D model of the unknown object. From the 2D images, silhouettes are extracted to create a 3D visual hull, which is merged with the more precise 3D shape data retrieved from the structured light technique. The approach in turn analyzes the model and generates a robust force closure grasp.

Dune et al. [117] determine the quadric that best resembles the shape of the object, which is done by using multiple view measurements. The quadric is estimated in each 2D view. The robot arm will already start moving towards the unknown object after the first quadric estimation is obtained, which results in a fast real-time grasping algorithm.

Similar work is presented by Yamazaki et al. [78]. In this approach, the 3D model of the unknown object is retrieved through SFM, which stands for ‘structure from motion’. By considering the gripper’s width, a good grasp is said to be found in a short amount of time.

Lippiello et al. [68] place a virtual elastic surface around the point cloud of the object, then this surface is shrunk at every iteration step (new image acquisition) until this intercepts with some points of the object. Attractive forces of points on the object will make an equilibrium with the elastic forces of the virtual surface in order to present the 3D model. During the construction of the virtual surface, the grasp planner is already active thus moving the end effector towards the unknown object.

B. Symmetries

When working with one 3D camera and without changing the angle on a specific object, the obtained point cloud contains occlusions. For instance, when the camera is in front of an object, no information of the back of the object can be given. The approach [116] by Bohg overcomes this problem by considering symmetries found in human-made objects. Their algorithm first tries to determine the planar symmetry on which the detected point cloud of the object will be mirrored about. After the mirroring of the points, a surface approximation is applied, this closes the object in order to find grasping locations on the object.

C. Decomposition

The decomposition with respect to the object, it means that the object is factorized into different parts. Factorizing into simple parts will decrease computation times when trying to grasp complex models.

Miller et al. [79] and Goldfeder et al. [118] use shape primitives to simplify the object, however they consider knowing the model before. The principle can still be implemented to use it for unknown objects as shown in the work of Eppner and Brock [94]. The grasping approach transforms the point cloud into shape primitives and a grasp is chosen depending on these shapes.

Huebner and Kragic [74] also use shape primitives to represent an unknown object. The point cloud of the object is transformed into a minimum volume bounding box (MVBB). This MVBB is split into multiple MVBBs and fitted in order to get more resolution of the actual model. The splitting is continued until more splits are not beneficial.

In the work of Hsiao et al. [92], a bounding box is placed around the available point cloud of an unknown object. Heuristics are applied to find the most suitable grasp. This approach also incorporates a local grasping approach.

D. Surface

A more straightforward approach to grasp an unknown object is to look at the available point cloud of the object and reconstruct a fitting surface of the object using those points.

In the work of Lee et al. [77], a 3D model is retrieved by using stereo matching. From the matching, a dense map is created. A three-dimensional interpolation (the triangular mesh method) is applied on the dense map. Suitable grasps can be located on the triangular mesh of the target object.

2.2.2 Local grasping approaches

A. Edges

A grasping approach with the use of edges of an object has been used by Jiang et al. [119]. The algorithm finds grasping locations by fitting a so-called “grasping rectangle” on an image plane. The rectangle describes the configuration of the gripper. The grasping approach also includes a learning algorithm in order to select the best grasping location depending on the object shape. The use of the learning algorithm increases the success rate of the grasp but increases the computation time.

Lin et al. [84] extends the principle of the grasping rectangle by looking at the contact area of the grasping rectangle. For instance, if the contact area is too small, the grasp is likely to fail and a better grasp can be picked. The success-rate when incorporating this technique is higher than Jiang et al. [119].

In Popovic et al. [65], grasps are generated based on edge and texture information of the unknown object. Baumgartl and Henrich [110] use Hough transformation to find edges in a 2D image. A check has been done to verify if the edges are long enough to be grabbed by the gripper. Another check is done to verify if the parallel edges fit into the gripper’s width. The two quick checks result in a fast grasping approach.

Richtsfeld and Vincze [120] detect grasp points on top surfaces of unknown objects. Firstly, a 3D mesh generation is applied on the segmented point cloud, and then the top surface can be extracted using a 2D DeLauney triangulation. Only information of the rim points and feature edges are left. One grasp point is found by finding the minimum distance from the center of mass to the edge. The second grasping point can be selected by extending the line of the first grasp point to the center of mass to the edge on the other side.

Similarly, Bodenhagen et al. [121] use machine learning to find suitable grasp on 3D edges of the unknown object. They refine an initial grasping behavior based on the 3D edge information by learning. A prediction function is used to compute likelihood for the success of a grasp using either an offline or an online learning scheme.

B. Boundary

The proposed grasping approach of Ala et al. [85] retrieves graspable boundaries and convex segments of an unknown object. From a 3D camera, the scene is segmented and a point cloud of the unknown object is left. With the use of blob detection, the boundaries of the object are retrieved. These boundary lines are then transformed into straight lines. The grasp planner tries to find parallel contact points in order to execute an envelope grasp. When an unknown object has a desirable thickness, then one contact point can be retrieved in order to execute a boundary grasp.

Maldonado et al. [122], ten Pas and Platt [113] try to fit the shape of the gripper on the available point cloud of the object(s). The latter uses a detailed segmentation to be able to pick objects from dense scenes and incorporates learning that significantly improves the grasp success rate.

In the grasping approach of Navarro [47], the unknown object center is estimated with the available point cloud cluster. Only round objects are considered with this approach and the objects are tracked on a conveyer belt. The gripper is aligned above the object to grasp it.

The work of Suzuki and Oka [114] estimate the principal axis and centroid of the unknown object on the retrieved point cloud to produce a stable grasp. The approach is shown to produce a high success rate for a set of household objects.

C. Silhouette

In the work of Calli et al. [86], the grasping algorithm uses curvature information of the silhouette of an unknown object. Using Elliptic Fourier Descriptors (EFD), the silhouette of the object can be modeled from a 2D image. To find grasping points, local minima and maxima curves of the silhouette are evaluated. Force closure tests are applied onto the grasping points to get the final, likely stable, grasping points. The grasping points are 2D points to help align the gripper.

Lei and Wisse [123] perform a force balance calculation in order to find suitable grasping points. Once a point cloud cluster of an unknown object is retrieved with the use of one or two 3D cameras, the coordination system of the object is created. After that the cloud points are projected on the XOY plane and a concave hull method is applied to extract the contours of the object. A graspable zone is calculated from this contour and then the force balance is computed on the XOY plane to find the maximum force balance. In order to match the gripper's angle with the angle of the object, a force balance is also computed on the XOZ plane. This is a robust grasping approach which is faster than for example [94].

The work of Lei and Wisse [124] which is based on [123] utilizes data from two 3D cameras to build virtual object coordination systems (VOCS) from different virtual viewpoints. From these coordination systems, multiple XOY and XOZ planes can be created. Force balance can be computed on all these planes. The maximum force balance resembles the best possible grasp. This grasping approach is robust and finds favorable grasps.

D. Saliency

In the work of Bao et al. [87], saliency is being used to segment the scene and find unknown objects. The algorithm is mainly useful for dealing with multiple unknown objects.

E. Tactile feedback

As for global grasping approaches, there can also be local grasping approaches that use tactile sensory data to find a suitable grasp. This is shown by the work of Haschke [125] where with the use of tactile servoing, it can, for example, establish and maintain grasping.

The approach of Hsiao et al. [92] also includes a local grasping approach part. The grippers in this approach are fitted with tactile sensors to help to adjust the grasp when collisions are found during the execution of the grasp found by the global grasping approach part.

2.3 Comparison

In this section, we will make comparisons about the different grasping approaches investigated in section 2.2. The different approaches will be compared with each other by looking at characteristics that the approaches have in common. In the end of this section, the comparison outcome is discussed.

2.3.1 Comparison table

The approaches described in section 2.2 have been added to Table 2.1. This aids in comparing the different approaches by highlighting chosen approach characteristics. The following characteristics are chosen:

- Object-grasp representation: as explained in the beginning of section 2.2, existing unknown object grasping approaches can be categorized into two groups: global and local grasping approaches. This can show if one group in particular is better performing than the other.

Table 2.1: Comparison of existing unknown object grasping approaches ('NA'= Not Applicable)

Literature	Object		Object features			Vision based only	Camera position		Multi-fingered	Grasp closure		Non-grasping movement of arm	Cluttered scene handling	Rate of success	Execution time
	Local	Global	2D	3D	multi		Overhead	Eye in hand		Form	Force				
Bohg et al. [116]		√		√		√	√		√		√			+	-
Bone et al. [75]		√			√			√			√	√		++	--
Dune et al. [117]		√	√			√		√			√	√		-	+
Eppner and Brock [94]		√		√		√	√		√	√	√		√	-	?
Huebner et al. [74]		√		√		√	√		√		√			-	-
Lee et al. [77]		√		√		√	√		√		√			--	?
Lippiello et al. [68]		√		√		√		√	√		√	√		++	+
Yamazaki et al. [78]		√		√		√		√			√	√		++	-
Ala et al.[85]	√			√		√	√				√		√	++	+
Bao et al. [87]	√			√		√	√				√		√	+	?
Baumgartl et al. [110]	√		√			√	√				√		√	++	+++
Bodenhagen et al. [121]	√			√		√	√				√		√	--	?
Calli et al. [86]	√		√			√		√		√				?	?
Haschke [125]	√			√			NA	NA	√		√	√		++	?
Jiang et al. [119]	√			√		√	√				√		√	++	?
Lei and Wisse [123]	√			√		√		√		√	√	√		++	+++
Lei and Wisse [124]	√			√		√		√		√	√			++	+++
Lin et al. [84]	√			√		√		√			√		√	++	+++
Maldonado et al. [122]	√			√		√	√		√		√		√	++	?
Navarro [47]	√			√		√	√				√			?	?
Ten Pas and Platt [113]	√			√		√	√				√		√	++	+++
Popovic et al. [65]	√			√		√	√				√		√	--	?
Richtsfeld et al. [120]	√			√		√	√				√		√	+	--
Suzuki and Oka [114]	√			√		√	√				√			+	++
Hsiao et al. [92]	√	√			√		√				√			++	?

- Object features: the data given to the approach can be 2D, 3D or a combination called 'multi' in the table. This information helps determining which data is most suitable for grasping.

- Vision-based only: if an approach is not using vision data only, the approach can be more difficult to implement and likely more expensive since more hardware are needed. A good example of this is the approach of Hsiao et al. [92] in which tactile sensors are mounted on the gripper.

- Camera-position: the camera-position can be of great importance for retrieving valuable information about the object. Approaches using an eye-in-hand camera can view the objects from multiple viewpoints [75, 117].

- Multi-fingered: when using approaches with multiple fingers (more than two), a grasp can be more stable since there are more places the object is grasped. This assumption can be checked with this comparison.

- Grasp closure: there can be two kinds of grasp closures: form and force closures. Form closures depend on the shape of the target object, these grasps usually place the fingers of the gripper in such a way that the object cannot fall out of the hand easily. This closure is for instance being used by Calli et al. [86]. Force closures press the fingers of the grippers (using force) on the object in order to keep it in the gripper.

- Non-grasping movement of arm: some approaches have to perform an extra motion of the arm to get more data of the unknown object. This can be time consuming.

- Cluttered-scene handling: this means that the approach is able to distinguish multiple unknown objects and is able to grasp them separately.

- Rate of success: from the literature, an estimate can be given on the success rate of the grasping approach. Lower than 70%, between 70% - 80%, between 80% - 90% and higher than 90% success rate is marked with --, -, + and ++ respectively. When no information about the success rate is given it is marked with ?.

- Execution time: to identify fast performing approaches we looked in the literature to find meaningful information about execution times. Since different processing power is used in the approaches, we limit ourselves to the presented execution times in the corresponding paper. Approaches in the literature which can finish the grasping process within 4 seconds are marked with +++. Between 4-8 seconds with ++, between 8-12 seconds with +, between 12-16 seconds with - and approaches that take longer than 12 seconds are marked with a --. When no information is given in the literature, a ? has been given instead.

2.3.2 Comparison discussion

From Table 2.1, it can be noted that among the eight global grasping approaches, five of them are designed for multi-fingered grippers (62.5%). Comparing this to the 2 of the 16 local grasping approaches (12.5%), it can be noticed that global grasping approaches are more suitable for multi-fingered grippers. Grasps found by Global grasping approaches are mostly with a force closure.

Once multiple object features are used for grasping, then the approach is not vision-based only. These approaches use for instance tactile sensor data. Except for Calli et al. [86], all the approaches use a force closure.

For approaches of non-grasping movements of the arm, all have an eye-in-hand camera position. When the camera is fixed, movement of the arm will not result in any change with respect to the data of the unknown object. When a movement is made with the arm incorporating eye-in-hand camera position, there will be change in the data. Cluttered-scene handling is usually connected to an overhead camera position. This is to be expected since multiple unknown objects can then be identified.

When looking at the rate of success, two things can be noticed. Firstly the overall performance of the global grasping approaches is less than the local grasping approaches. Since global grasping approaches try to represent a full 3D model, resulting in a lot of details of the unknown object are lost, a good example of this is the decomposition of the unknown object into blocks [74]. Secondly an eye-in-hand camera position performs better, likely because the unknown object data obtained by eye-in-hand system is more detailed to perform stable grasps.

From the available information in the literature, local grasping approaches have the lowest execution times. Not all approach literatures include information on execution time. As we mentioned before, the characteristic of execution time is dependent on the computing power.

Some approaches perform well in all the specified areas, which are all local grasping approaches. Take for instance the work of ten Pas and Platt [113], this planner is able to use 3D vision data to perform stable grasps for unknown objects from a cluttered-scene in a short amount of time. The work from Lei and Wisse [124] also shows favorable results like [114] though it does not work in cluttered scenes.

2.4 Conclusion

This chapter presented an overview on the existing unknown object grasping approaches. The approaches were sorted in groups and a short description of each approach was given. With the use of a comparison table that included all the approaches, remarks were given on common grasping characteristics. The comparison table clearly shows the advantage and disadvantage of every grasping algorithm to help the future researchers quickly picking a suitable grasping approach with their requirements.

Acknowledgement

The work leading to these results in this chapter was supported by the EU project: Factory in a day, which received funding from the European Community’s Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 609206.

3

Fast grasping of unknown objects using principal component analysis

This Chapter is a journal paper in Journal of AIP Advances:

Qujiang Lei, Guangming Chen, Martijn Wisse. Fast grasping of unknown objects using principalcomponent analysis. In Journal of AIP Advances, Volume: 7, Issue: 9, Pages: 1-21, 2017.

This journal paper is based on two earlier conference papers:

1. Qujiang Lei, Martijn Wisse, "Fast grasping of unknown objects using force balance optimization", 2014 IEEE International Conference on Intelligent Robots and Systems (IROS 2014), pp. 2454-2460. Chicago, USA.

2. Qujiang Lei, Martijn Wisse, "Unknown object grasping using force balance exploration on a partial point cloud", 2015 IEEE International Conference on Advanced Intelligent Mechatronic (AIM 2015), pp. 7-14. Busan, Korea.

Abstract

Fast grasping of unknown objects has crucial impact on the efficiency of robot manipulation especially subjected to unfamiliar environments. In order to accelerate grasping speed of unknown objects, principal component analysis is utilized to direct the grasping process. In particular, a single-view partial point cloud is constructed and grasp candidates are allocated along the principal axis. Force balance optimization is employed to analyze possible graspable areas. The obtained graspable area with the minimal resultant force is the best zone for the final grasping execution. It is shown that an unknown object can be grasped more quickly provided that the component analysis principle axis is determined using single-view partial point cloud. To cope with the grasp uncertainty, robot motion is assisted to obtain a new viewpoint. Virtual exploration and experimental tests are carried out to verify this fast gasping algorithm. Both simulation and experimental tests demonstrated excellent performances based on the results of grasping a series of unknown objects. To minimize the grasping uncertainty, the merits of the robot hardware with two 3D cameras can be utilized to suffice the partial point cloud. As a result of utilizing the robot hardware, the grasping reliance is highly enhanced. Therefore, this research demonstrates practical significance for increasing grasping speed and thus increasing robot efficiency under unpredictable environments.

3.1 Introduction

Unknown object means an item that has neither geometric model nor appearance information. Grasping unknown objects is highly challenging for the robots working in unfamiliar environments [126]. With the increasing demand of various robots that are being used in contemporary society, increasing grasping speed becomes one of the primary tasks for improving the efficiency of robots manipulation [35].

A vast amount of research has been conducted on grasping unknown objects over the past few decades, and many achievements have been attained. To grasp an unknown object, geometric properties (i.e. symmetries [116], surface [77], edges [119-121], boundary [47, 85, 114], silhouette [86], saliency [87]) are generally used to construct contours of the target object. For instance, Maldonado et al. [122], ten Pas and Platt [113] fitted the shape of the gripper on the boundary of partial point cloud of the target unknown object. To obtain geometric contours of the unknown object, two methods are commonly used. One is to use tactile sensors to detect the geometric properties of unknown target object [127-131]. The other is to use a camera to move around to explore the unseen part [86, 94, 95]. Both methods have high grasp security but are very time expensive. The reason for the first method is that it requires long time to carry out sufficient contacts with the object. For the second method, much time is used because of the movement of the camera.

To save time, several fast grasping approaches have been proposed, which can be found in [84, 94, 110, 113, 114]. Among these approaches, Johannes Baumgartl [110] uses RGB images as input, which can quickly provide 2D geometric information of an unknown object. However, using this approach cannot always promote successful grasping because an unknown object can have no parallel edges. By contrast, the other researchers [84, 94, 113, 114] employ a partial point cloud, which can formulate more realistic geometric model. Because the geometry contour is approximated based on partial point cloud, it can significantly reduce computational load and thus accelerating grasping speed. Nevertheless, ignoring other information, such as occlusions, may introduce grasp uncertainty and result in grasp failure. To deal with the uncertainty with the usage of partial point cloud, a new method for reducing geometric information for grasping unknown objects can be explored.

The goal of this chapter is to reduce grasping time for unknown objects whilst the grasping security is maintained. In this chapter, we propose a novel approach to guide the grasping procedures of unknown objects based on the principal component analysis. Based on this, a single-view point cloud is used to reduce the data for formulating geometric contour to save the computational time. The feature of our grasping approach is to allocate grasp candidates along the principal axis such that the possibility of useless grasp candidates can be greatly decreased. This algorithm is shown to be successful on the base of both simulation and experimental tests. By taking the advantage of robot hardware, the grasping uncertainty is minimized. Therefore, this research demonstrates practical significance for increasing grasping speed.

This chapter is organized in this way: section 3.2 introduces our fast grasping algorithm; section 3.3 shows the simulation results; section 3.4 gives the experiment validation; section 3.5 outlooks an approach on enhancing the grasp security using two 3D cameras. Finally, the conclusion of this research is provided in section 3.6.

3.2 A fast grasping approach

This section presents a detailed explanation of our fast grasping approach for unknown objects. This approach adopts a grasping algorithm which utilizes a single-view partial point cloud. Furthermore, the solutions for tackling exceptional cases of grasping failure by applying this algorithm are elaborated.

3.2.1 Algorithm

Because the configuration of the robot hand follows a Special Euclidean group SE (3) in practice, it implies many possibilities when locating a robot hand in three-dimensional (3D) space. In our approach, the principal axis of the target unknown object is used to find out proper positions for executing a successful grasping action. Figure 3.1 outlines our fast grasping algorithm, in which it shows a single-view partial point cloud of the target object is used as input. For general case of grasping unknown objects, seven steps are required, the details of which are described in section 3.2.2. For the exceptional case to achieve a successful grasping, the solution is illustrated in section 3.2.3.

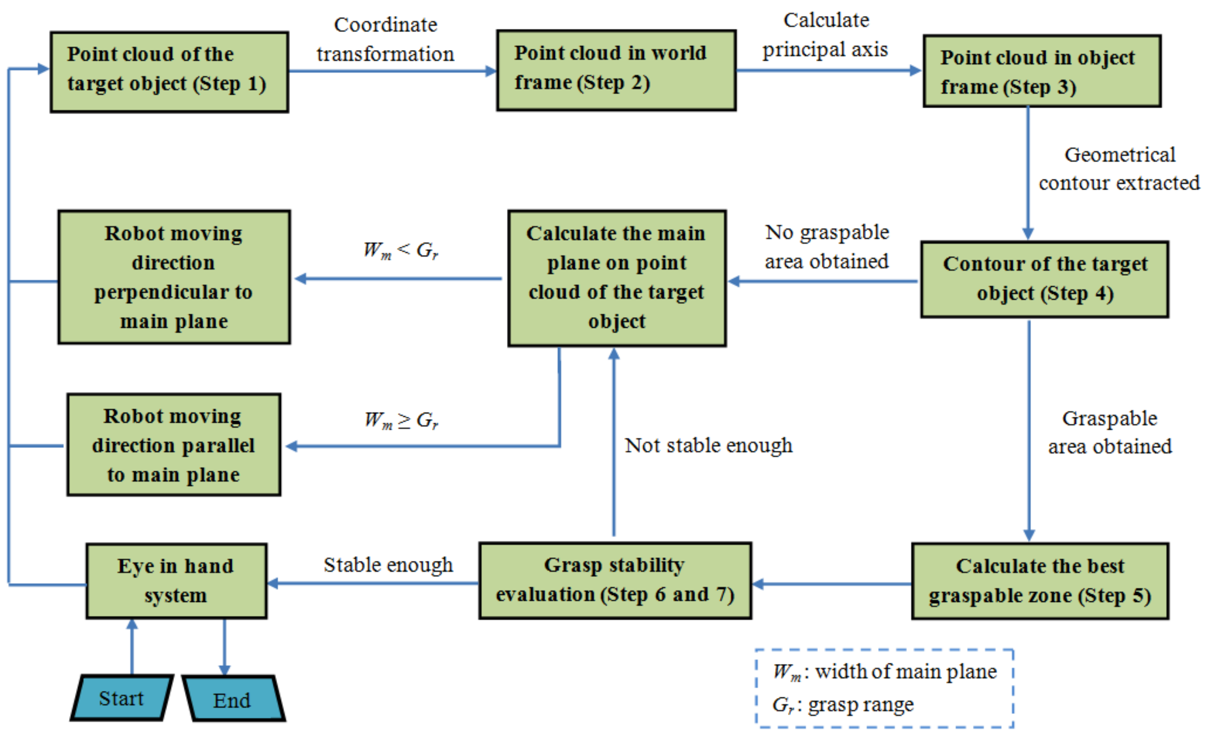

Point cloud of the target object (Step 1)
Coordinate transformation
Point cloud in world frame (Step 2)
Calculate principal axis
Point cloud in object frame (Step 3)
Geometrical contour extracted
Contour of the target object (Step 4)
No graspable area obtained
Calculate the main plane on point cloud of the target object
$W_m < G_r$
Robot moving direction perpendicular to main plane
$W_m \geq G_r$
Robot moving direction parallel to main plane
Not stable enough
Graspable area obtained
Calculate the best graspable zone (Step 5)
Grasp stability evaluation (Step 6 and 7)
Stable enough
Eye in hand system
Start
End
W_m: width of main plane
G_r: grasp range

Figure 3.1: Overview of our fast grasping algorithm

3.2.2 Grasping unknown object based on the single-view partial point cloud

Figure 3.2 presents the procedure to grasp the target unknown objet based on the single-view partial point cloud. Figure 3.2 (a) shows a simulation setup in which a spray bottle is used as the target unknown object. An eye-in-hand system is composed of a 3D camera sensor and a UR5 robot. The 3D camera sensor is used to acquire the raw point cloud for the given environment. In order to accelerate computing speed, distance filtering is initially applied on the raw point cloud to remove those points that are out of the reach of the robot arm, as shown in Figure 3.2 (b). Figure 3.2 (c) shows the transformation of the partial point cloud to the world frame. Figure 3.2 (d) illustrates the transformation of the partial point cloud to the object frame. Figure 3.2 (e) gives the projected point cloud in the object frame. Figure 3.2 (f) presents the concave hull contour of the projected point cloud. Figure 3.2 (g) depicts all the crossing points. Figure 3.2 (h) shows possible grasp zone within grasping range of the gripper. Figure 3.2 (i) points the method to obtain the best grasp on the graspable zone. Finally, Figure 3.2 (j) provides an example of grasp execution. The detailed seven steps for conducting grasping an unknown object are presented as follows.

Step 1: Obtaining the single-view partial point cloud of the target unknown object

Figure 3.2 (a) shows a simulation setup in which a spray bottle is used as the target unknown object. An eye-in-hand system is composed of a 3D camera sensor and a UR5 robot. The 3D camera sensor is used to acquire the raw point cloud for the given environment. In order to accelerate computing speed, distance filtering is initially applied on the raw point cloud to remove those points that are out of the reach of the robot arm, as shown in Figure 3.2 (b). Then down sampling is used to reduce the density of the points. After that, Random Sample Consensus (RANSAC) method is employed to remove the table plane. The determination of principal axis is given as follows.

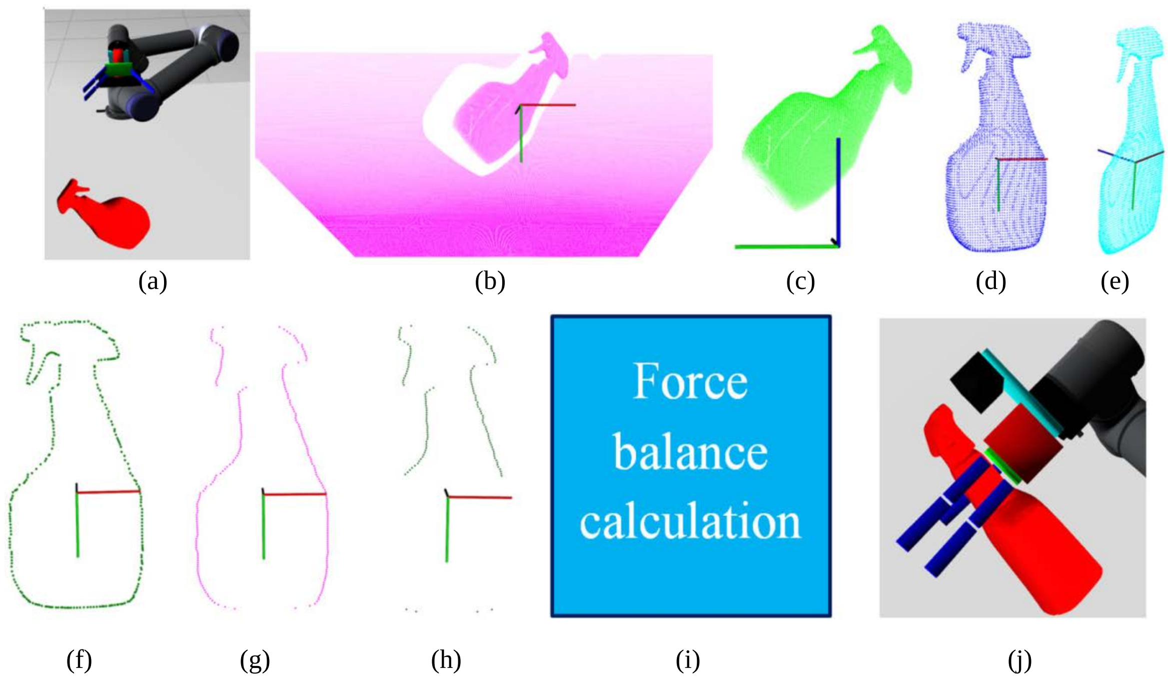


Figure 3.2: The procedure to process the single-view partial point cloud. (a) simulation setup; (b) the filtered distance and down-sampled point cloud. (c) transformation of the partial point cloud to the world frame; (d) transformation of the partial point cloud to the object frame; (e) projected point cloud in the object frame; (f) concave hull contour of the projected point cloud. (g) all the crossing points; (h) possible grasp zone within grasping range of the gripper. (i) the method to obtain the best grasp on the graspable zone. (j) an example of successful grasp execution.

After the single-view partial point cloud is obtained by a 3D camera, Principal Component Analysis (PCA) is performed to approximate the centroid and the principal axis of the object. PCA is a statistical technique for analyzing correlation between observed data. Let $X = (\chi_1, \chi_2, ..., \chi_n)$ be the object point set, where χ_i is a point in the 3D space R3. The centroid of the point set $P_{centroid}$ is calculated by the following equation:

$$p_{centriod} = \frac{1}{n}\sum_{i=1}^{n}\chi_i \tag{3.1}$$

Giving the values of a point cloud $X = (\chi_1, \chi_2, ..., \chi_n)$, the covariance matrix s can be calculated by using Equation (3.2):

$$s = \begin{pmatrix} \sigma_{xx}^2 & \sigma_{xy}^2 & \sigma_{xz}^2 \\ \sigma_{yx}^2 & \sigma_{yy}^2 & \sigma_{yz}^2 \\ \sigma_{zx}^2 & \sigma_{zy}^2 & \sigma_{zz}^2 \end{pmatrix} \tag{3.2}$$

in which the nine elements are the values of covariance for the 3D coordinates. The eigenvalues $\lambda_1 > \lambda_2 > \lambda_3$ ($\lambda_i \in R$ and i =1, 2, 3), and the corresponding eigenvectors μ_1, μ_2, μ_3 ($\mu_i \in R^3$ and i =1, 2 ,3) can be obtained. The eigenvector μ_1 corresponds to the largest eigenvalue λ_1, which approximates the direction of the principal axis. Using the centroid ($P_{centroid}$) of the single-view partial point cloud and the direction of the principal axis, the principal axis can be obtained.

Step 2: Transforming single-view partial point cloud from camera sensor frame to world frame

The obtained single-view partial point cloud in step 1 is retained in the camera frame. To carry out the analysis using coordinate system, the single-view partial point cloud must be transformed into the world frame. Feedbacks from the joint encoders of the robot arm are used to construct the transformation matrix from the end effector of the robot to the base link of the robot. The transformed single-view partial point cloud in the world frame can be seen in Figure 3.2 (c). The transformed single-view partial point cloud in the object frame is shown in Figure 3.2 (d).

Step 3: Constructing object frame

Figure 3.3 (a) displays three coordinate systems from the eye-in-hand system and the target unknown object, namely, the world frame ($X_{world}Y_{world}Z_{world}$), the 3D camera frame ($X_{3Dsenor}Y_{3Dsenor}Z_{3Dsenor}$) and the object frame ($X_{obj}Y_{obj}$ Z_{obj}). The principle axis and the mass center of the partial point cloud are used to build the object frame. As shown in Figure 3.3 (b), the mass center of the target object is used as the origin point of the object frame.

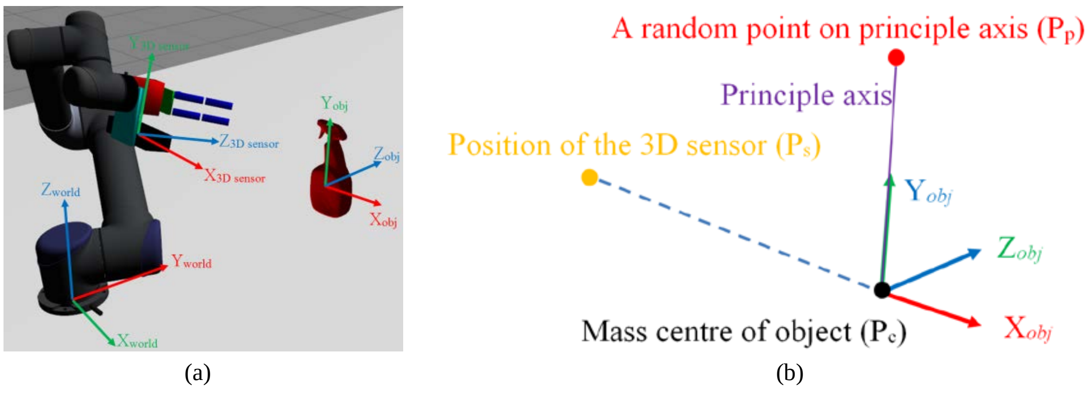

Y3D sensor
Z3D sensor
X3D sensor
Yobj
Zobj
Xobj
Zworld
Yworld
Xworld
(a)
A random point on principle axis (Pp)
Principle axis
Position of the 3D sensor (Ps)
Yobj
Zobj
Mass centre of object (Pc)
Xobj
(b)

Figure 3.3: Establishment of the coordinate systems. (a) world frame, the 3D sensor frame and the object frame. (b) illustration of building the object frame of the target unknown object.

The principal axis is used as the Y axis of the object frame. The X axis and the Z axis of the object frame can be determined by applying Equation (3.3).

$$\begin{cases} \overrightarrow{O_c X} = \overrightarrow{O_c P_s} \times \overrightarrow{O_c P_P} \\ \overrightarrow{O_c Z} = \overrightarrow{O_c Y} \times \overrightarrow{O_c X} \end{cases} \tag{3.3}$$

in which P_p and P_s respectively stand for a random point on the principal axis and the position of the 3D camera sensor.

Step 4: Acquiring concave hull contour of the single-view partial point cloud

After we obtained the partial point cloud in the object frame, the contour of the partial point cloud will be abstracted. Specifically, the partial point cloud is firstly projected to the XOY ($X_{obj}O_{obj}Y_{obj}$) plane of the object frame. Figure 3.2 (e) shows the projected point cloud. There are two methods to obtain the contour of the projected point cloud, namely, the concave hull contour and the convex hull contour. For this scenario, concave hull contour can better represent the geometric shape of the target unknown object. Concave hull contour of the target object is extracted as shown in Figure 3.2 (f).

Step 5: Calculating graspable zones

To figure out all graspable area within grasping range on the contour of the target object, grasp candidates are allocated along the principal axis (the Y axis of the object frame). The minimum Y value ($y_{\min}$) of all the points on the concave hull contour is firstly decided. Subsequently, a straight line parallel to the X axis is used during the search process from the top to the bottom. In this manner, the most left and most right crossing points between the straight line and the concave hull contour can be determined. An appropriate step (Δy shown in Figure 3.4) is added to $y_{\min}$ such that the searching process can apply to the whole concave hull contour from the top to the bottom.

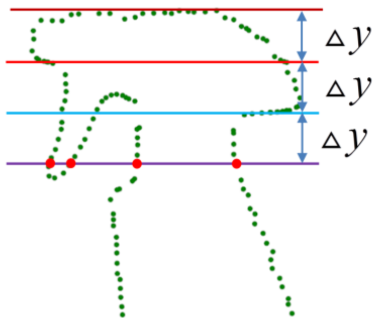


Figure 3.4: The method to determine the graspable zone of the target unknown object.

The straight line parallel to the X axis can be obtained according to Equation (3.4). All crossing points can construct a point cloud as shown in Figure 3.2 (g). For each straight line, when the distance between the most left point and the most right is smaller than the grasping range of the robot hand, the most left point and the most right point will be saved to construct a point cloud, which is shown in Figure 3.2 (h). The graspable zone of the target unknown object is represented by the point cloud in Figure 3.2 (h).

$$y = y_{\min} + n\Delta y \tag{3.4}$$

Step 6: Optimizing total force on the XOY plane of the object frame

In the step 5, the graspable zone is extracted out by considering the grasp range of the robot hand. To obtain the best grasp on the graspable zone, Figure 3.5 illustrates the evaluation process to allocate grasp candidates along the principal axis from the top to the bottom. As shown in Figure 3.5 (a), the green, blue and red rectangles stand for three example grasp candidates. In order to achieve a stable final grasp, force balance computation is carried out for every grasp candidate. The blue points in Figure 3.5 (b) stand for a grasp candidate. Points on the two grasp sides are used to fit two straight lines, and the angle between the two straight lines is used to evaluate the stability of this grasp. The bigger the angle is, the less stable the grasp is.

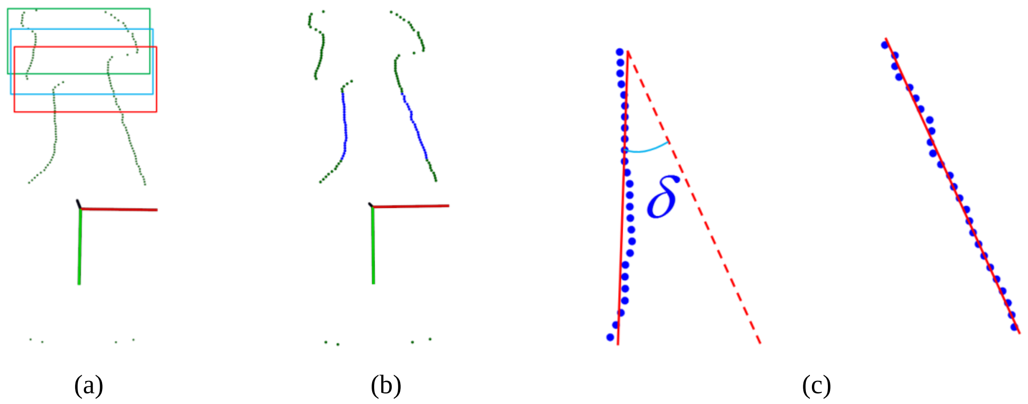


Figure 3.5: Evaluation of the grasping candidates within grasp range (a) green, blue and red rectangles of grasp candidates; (b) blue points of a grasping candidate; (c) two fitting lines.

The straight line can be expressed as $y = kx + b$, in which the coefficients k and b can be determined using Equation (3.5). The two red lines in Figure 3.5 (c) stand for the two fitting lines. Figure 3.6 demonstrates the results of force balance computation of the graspable zone. We can observe that force balance reaches the minimum value when the number comes to search index 33, which corresponds to the best force balance.

$$\begin{cases} k = \dfrac{\sum_{i=1}^{n} x_i y_i - \dfrac{1}{n}\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{\sum_{i=1}^{n} x_i^{\,2} - \dfrac{1}{n}\sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i} \\ b = \dfrac{1}{n}\sum_{i=1}^{n} y_i - k\dfrac{1}{n}\sum_{i=1}^{n} x_i \end{cases} \tag{3.5}$$

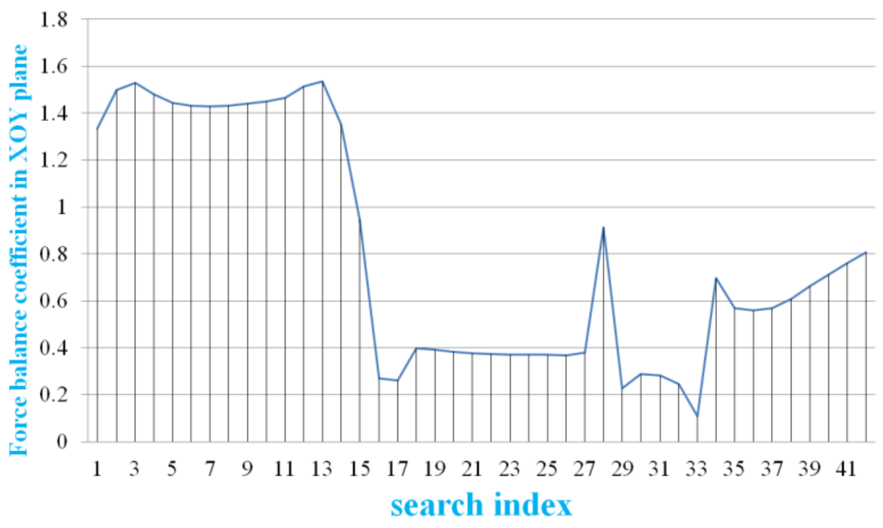
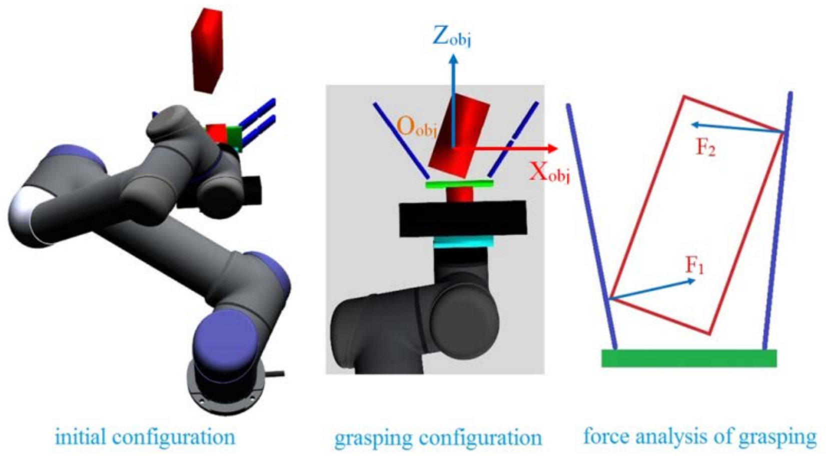


Figure 3.6: The results of force balance computation on the XOY plane.

Step 7: Force balance evaluation on the XOZ plane of the object frame.

In the previous step, we explained the method to determine the best grasp using force balance computation on the XOY plane of the object frame. Next, the effect of force balance on the XOZ plane on the stability must be also studied. Figure 3.7 (a) shows an initial configuration of the robot and the target unknown object. Figure 3.7 (b) shows that the robot is approaching the grasp point. Figure 3.7 (c) shows the force balance analysis of this grasp candidate on the XOZ plane. F_1 and F_2 respectively stand for the force that the gripper imposes on the target object.



(a) (b) (c)

Figure 3.7: Evaluation of the grasp candidates on the XOZ plane (a) initial configuration of the robot and the target unknown object. (b) the robot becomes contact with the grasp point. (c) the force balance analysis of this grasp candidate on the XOZ plane.

The point cloud covered by the grasp candidate is extracted to be used for force balance analysis on the XOZ plane, which is illustrated in Figure 3.8. The green points in Figure 3.8(a) stand for the area covered by a grasp candidate. Figure 3.8 (b) shows the point cloud covered by grasp candidate is extracted and the two grasp sides are visualized as the red points in Figure 3.8 (b). The average Z values of the left and the right grasp sides are worked out shown as the Z_1 and Z_2 in Figure 3.8 (c). The difference between Z_1 and Z_2 is used to evaluate the stability of the grasp candidate on the XOZ plane. A threshold ($Z_{dif\,\max}$) can be set by this system, thus, when $|Z_1 - Z_2| < Z_{dif\,\max}$, the grasp candidate is saved, otherwise the grasp candidate is removed.

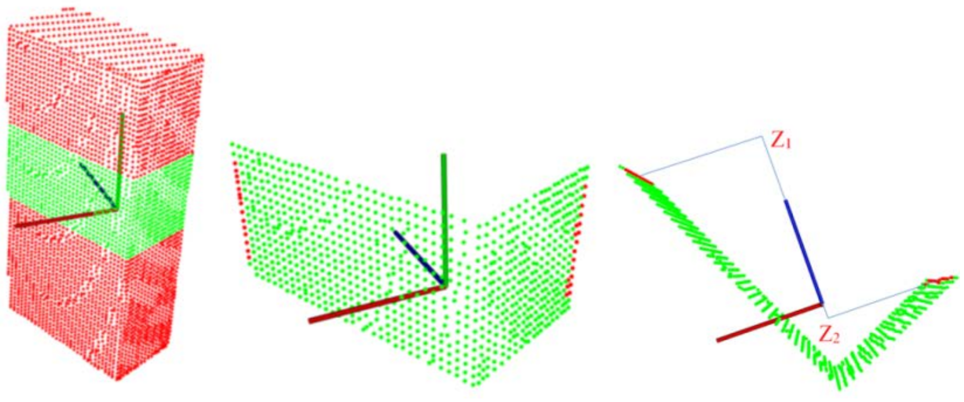


Figure 3.8: Evaluation of grasping an object on the XOZ plane. (a) the green area of point cloud covered by a grasp candidate. (b) extracted grasping areas and the two grasp sides of the red points. (c) The average values for Z of the left and the right grasping sides.

The above seven steps illustrate the fast grasping strategy of using principal component analysis based on the obtained single-view point cloud. However, there are four risks that can lead to grasping failures. Firstly, a grasping failure can occur when the principle axis cannot be obtained using single-view point. Figure 3.9 illustrates three successful grasps from the three perspectives of left, middle and right by rotating the robot arm. It also includes a grasping failure due to the fact that principle axis was not determined because of the point data loss.

Secondly, a graspable zone is not possible to be obtained when the gripper cannot cover the target object, which can be deduced in step 5. Thirdly, the angle (δ) corresponding to the force balance calculation can be too large (as shown in step 6), which infers that the object will be squeezed out when the robot tries to perform the grasping action. In addition, grasping failure can be triggered because of the resultant unbalanced force when the range between Z_1 and Z_2 for all grasp candidates are greater than the threshold (step 7).

Nevertheless, the grasping failure that is caused by the unbalanced force on the XOZ in step 7 can be resolved by using changing viewpoint. The solution for this exceptional case was already pointed out in Figure 3.1 of this algorithm. In the following subsection, the detailed explanation of the solution to the exceptional case is provided.

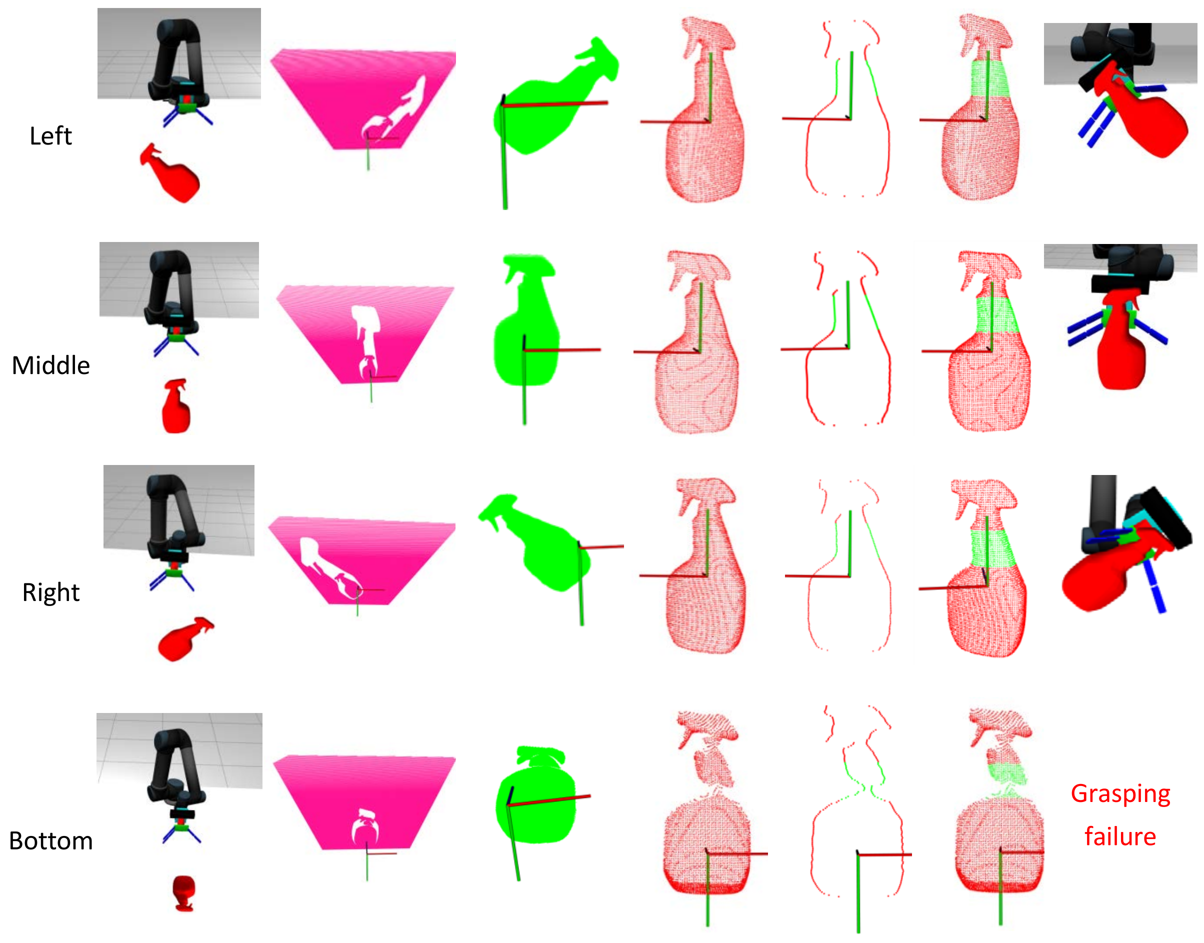
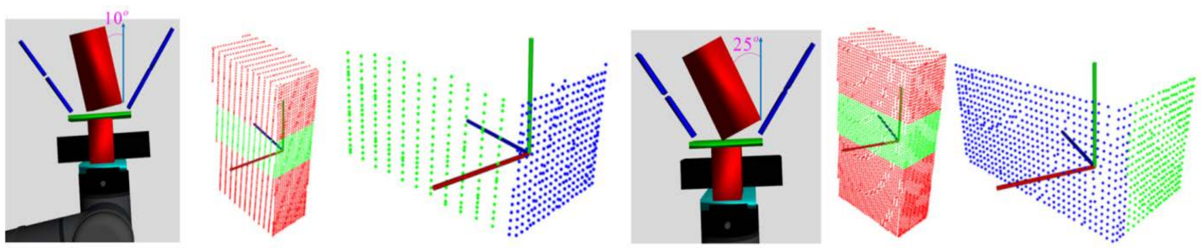


Figure 3.9: Grasping the same object from four different perspectives (left, middle, right and bottom) using a single-view point cloud by following the seven steps.

3.2.3 Solution for the exceptional case

To deal with the exceptional case that grasping positions cannot be achieved at one viewpoint, the main plane to guide the robot arm can be employed to activate another motion of the robot arm. In this way, a robot can move to another viewpoint to calculate to search for an executable grasp by following the above seven steps.

Figure 3.10 illustrates the activation of another motion of the robot arm, in which Figure 3.10 (a) and (d) provides two scenarios of two possible graspable areas obtained by the grasp algorithm. The green points in Figure 3.10 (b) and Figure 3.10 (e) respectively stand for the corresponding best grasping areas returned from the grasping algorithm at this viewpoint. The blue points in Figure 3.10 (c) and (f) respectively stand for the main plane of the grasp in Figure 3.10 (b) and (e). For the scenario as seen in Figure 3.10 (a), the shortest distance of movement of the robot arm is to move to the perpendicular direction of the main plane as shown in Figure 3.11 (a). For the other scenario shown in Figure 3.10 (d), the shortest distance of movement of the robot arm is to move to the tangent direction of the main plane as shown in Figure 3.11 (b).

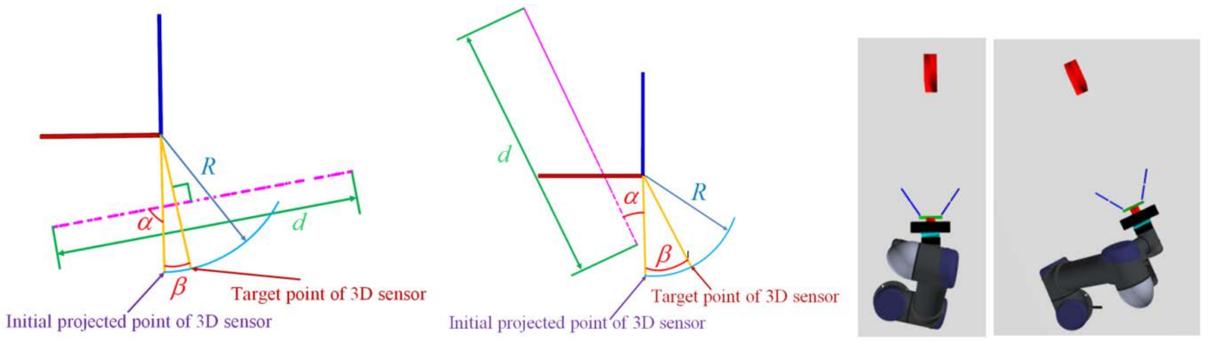


(a) (b) (c) (d) (e) (f)

Figure 3.10: The method to deal with the exceptions that no suitable grasp is found at a viewpoint plane. (a) and (d) two cases of best grasp found by the grasp algorithm; (b) and (e) the corresponding best grasping areas; (c) and (f) the main plane.

Referring to Figure 3.1, when the width of the main plane (W_m) is smaller than the graspable range of the gripper (G_r), then the robot will move to the perpendicular of the main plane. This situation corresponds to for the Figure 3.11 (a) which is obtained from Figure 3.10 (a). The angle (β) of the movement between the initial sensor point to the target sensor point should be $\beta = \pi / 2 - a$. Figure 3.11(b) shows the other situation which is that the width (d) of the main plane (W_m) is bigger than the grasp range of the gripper (G_r). The angle (β) of the movement between the initial sensor point to the target sensor point should be $\beta = \alpha$. The specific rotation can be worked out using Equation (3.6). Figure 3.11 (c) and (d) shows the robot arm arrives at the target sensor point corresponding to Figure 3.10 (a) and (d). When the robot arm arrives at the target viewpoint, the steps from step 1 to step 7 are repeated to search a suitable grasping area for the target object.

$$\begin{bmatrix} P_t.x \\ P_t.y \\ P_t.z \end{bmatrix} = \begin{bmatrix} x^2(1-c)+c & xy(1-c)-zs & xz(1-c)+ys \\ yx(1-c)+zs & y^2(1-c)+c & yz(1-c)-xs \\ xz(1-c)-ys & yz(1-c)+xs & z^2(1-c)+c \end{bmatrix} \begin{bmatrix} P_i.x \\ P_i.y \\ P_i.z \end{bmatrix} \tag{3.6}$$



R
α
d
β
Target point of 3D sensor
Initial projected point of 3D sensor
d
α
R
β
Target point of 3D sensor
Initial projected point of 3D sensor

(a) (b) (c) (d)

Figure 3.11: Strategies of moving the robot arm when no suitable grasp is found at one viewpoint, in which (a) and (b) are the obtained projected main plane (two purples lines), (c) and (d) are illustrations of the robot arm approaching the target sensor point.

To sum up, the application of our algorithm of using principal component analysis has been elaborated based on the seven steps. To solve the grasping failure of an exceptional case, robot motion is used to obtain single-view point data from another perspective. It can be inferred that our novel algorithm of using single-view partial point cloud can save time compared to other algorithms that used partial point cloud data. To verify and validate the efficiency and applicability of our approach, simulation and experimental tests are presented in the subsequent two sections.

3.3 Simulation test

In this section, our grasping algorithm is verified in a simulation environment. First, the structure of the simulation setup are illustrated. Next, the simulations for grasping unknown objects with different geometric shapes are performed using a single-view partial point cloud.

3.3.1 Structure of simulation setup

Figure 3.12 illustrates the simulation setup that consists of ROS, Gazebo and MoveIt!. ROS [186] (an open source robot operating system) is widely used in the community of robotics due to the simply operations. Gazebo (an Open Dynamics Engine simulator) is the state of art simulator that offers the ability to efficiently and accurately simulate complex task for robots. MoveIt! (a cutting edge software for robot motion planning) incorporates the latest achievements in navigation, manipulation and kinetics. The first and foremost part is simulation for a single-view partial point cloud obtained using a 3D camera, which is illustrated below.

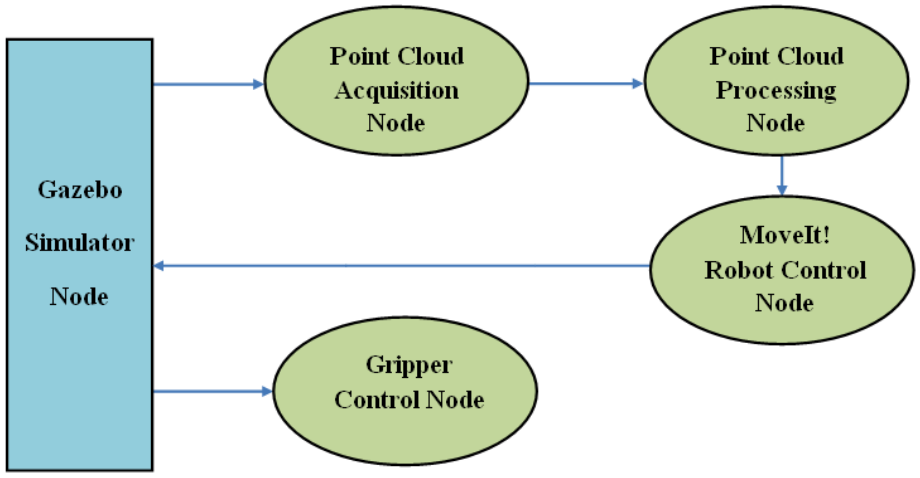

Point Cloud Acquisition Node
Point Cloud Processing Node
Gazebo Simulator Node
MoveIt! Robot Control Node
Gripper Control Node

Figure 3.12: Illustration of simulation setup consisting of ROS, Gazebo and MoveIt!.

3.3.2 Simulations based on a single-view partial point cloud using a 3D camera

The simulations of using a single-view partial point cloud as input are conducted to obtain suitable grasp. In the simulations, a spray bottle, a cup, a water bottle and an oatmeal box which have varying geometries are employed. The first column shows the simulation setup of the robot and the target object. The first three rows of Figure 3.13 illustrate the simulations to determine suitable grasp on a single-view partial point cloud. The last row of Figure 3.13 shows the method of using the main plane of the single-view partial point cloud to guide the robot to the second viewpoint. For the first three rows, the second column shows concave hull boundary of the single-view partial point acquired using one 3D camera. The blue points in the third column stand for the region with grasping range of the robot hand and this region is graspable zone. The grasp candidates are allocated from the top to the bottom of the graspable zone. Force balance computation on the XOY plane is conducted on these grasp candidates and the forth column shows the results of force balance computation. The corresponding best grasping areas are shown as green points in the fifth column. The last column demonstrates successful grasp execution for the target unknown objects. The last row demonstrates how to use the main plane of the single-view partial point cloud of the target object to guide the robot to the second viewpoint to achieve a suitable grasp.

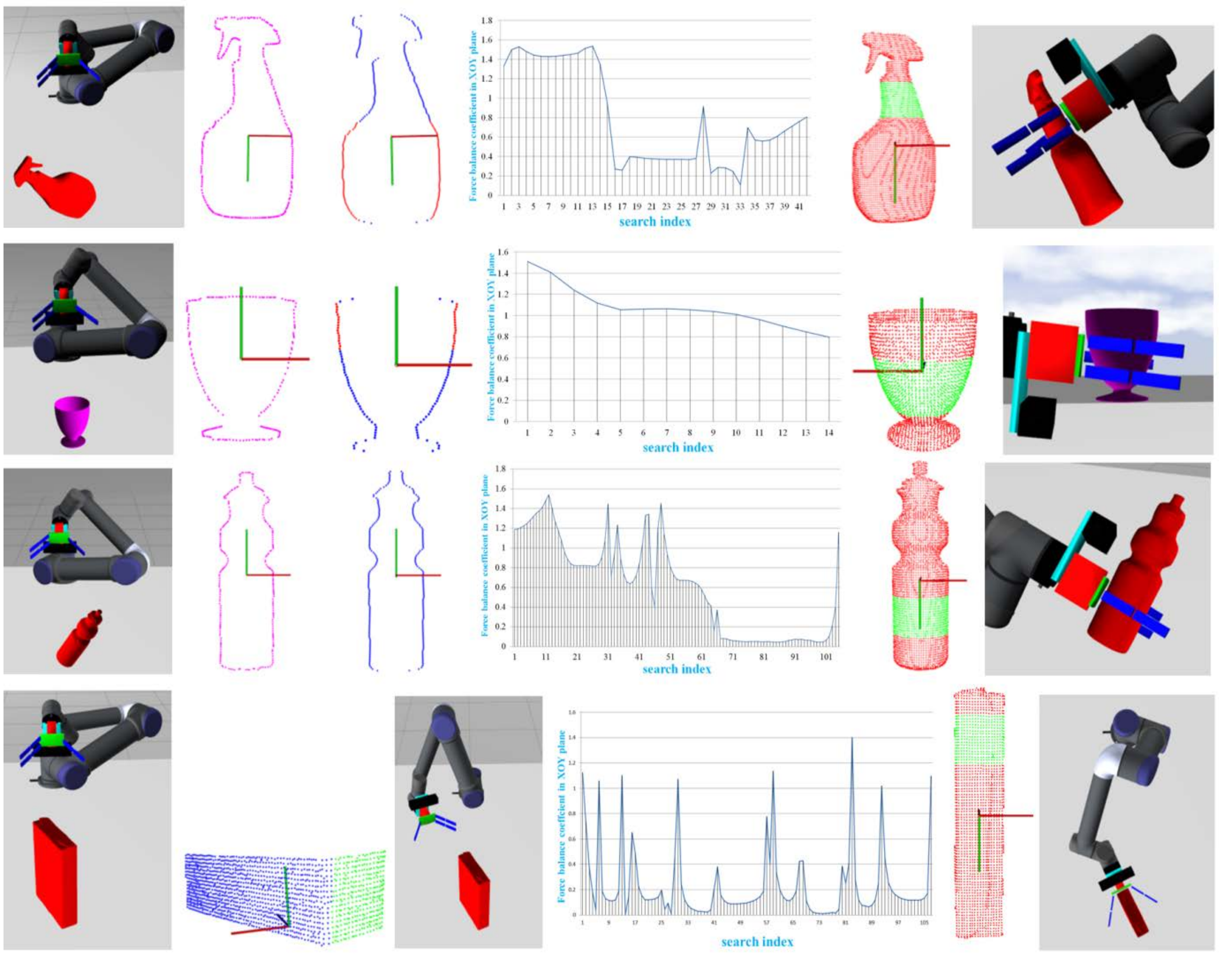


Figure 3.13: Simulation results of using a single-view partial point cloud as input.

Table 3.1 presents the detailed results of force balance computation on both the XOY plane and The XOZ plane. Based on force balance on the XOY plane, our grasping algorithm can be used to achieve stable grasping results for Spay bottle, Water bottle and Oatmeal box. For the selected cup, the force balance coefficient of the cup on the XOY plane is much larger due to its own geometrical character of no parallel sides. For force balance on the XOZ plane, results for all objects are quite satisfying. The maximum difference is 2.185 millimeter which cannot lead to massive movement of the target object when the robot tries to grasp it. As for the computing time, time spent to process a single-view partial point cloud is within one second.

Table 3.1: Simulation results of force balance computation using a single-view partial point cloud as input

	Spray bottle	Cup	Water bottle	Oatmeal box
XOY (radian)	0.108467	0.797198	0.0446418	0.000365332
XOZ (mm)	2.185	0.4709	0.01	1.56499

In summary, using the principle axis can reasonably determine the essential feature of an unknown target. It demonstrated that this grasping algorithm can achieve a greater grasping speed to obtain a suitable grasping areas compared to others, thus the proposed algorithm is numerically verified. In order to validate this novel algorithm in practice, the experimental tests are conducted as follows.

3.4 Experimental validation

In order to demonstrate the applicability of our proposed algorithm in reality, experimental tests are carried out using four different unknown objects of different shapes. The experimental tests are designed based on the references of the simulations and the availability of unknown objects.

3.4.1 Experimental description

An eye-in-hand system consisting of a Universal Robot arm UR5, an under-actuated Lacquey Fetch gripper and an Xtion pro live sensor is used to conduct experiment tests. The Xtion pro live sensor and the under-actuated gripper are installed at the end of the robot arm. With the reference of simulation models, four experimental tests are carried out as shown below as shown in Figure 3.14.

In the Figure 3.14, the first column presents the whole experiment setup. The second and third columns of Figure 3.14 show the single-view partial point cloud. The second column corresponds to the single-view partial point cloud obtained from the perspective of the front direction and the third column corresponds to the single-view partial point cloud from the back direction. The final grasps returned from our grasping algorithm of using a single-view partial point cloud as input are shown as the fourth column of Figure 3.14.

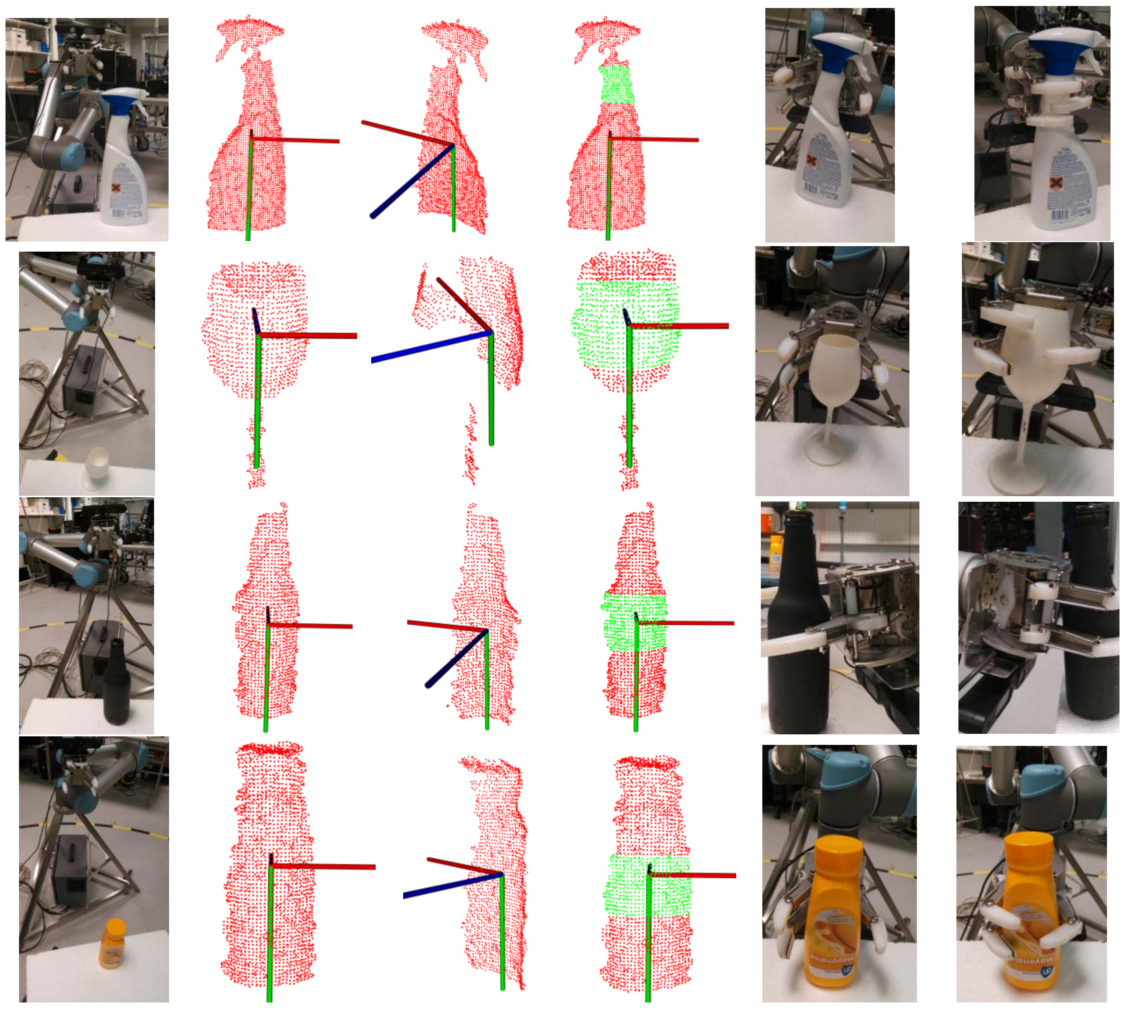


Figure 3.14: Snapshots from the experimental tests.

3.4.2 Results

Table 3.2 shows the experiment results of force balance computation on the XOY plane and the XOZ plane. We can find all final grasps are with a good force balance which can ensure the grasp stability. The fifth column of Figure 3.14 shows the under-actuated Lacquey Fetch gripper arrives at the grasping point. The last column demonstrates the successful grasp execution of the target unknown objects. Our grasping algorithm can quickly process the single-view partial point cloud of the target unknown object to output the final grasp within one second.

Table 3.2: Experiment results of force balance computation on the XOY plane and the XOZ plane

	Spray bottle	Wine glass	Beer bottle	Mayonnaise bottle
XOY(radian)	0.117	0.218	0.017	0.326
XOZ(mm)	2.635	0.504	0.363	0.002

Thus far, our algorithm has been validated using experimental tests. It demonstrated that our fast grasping algorithm can promote successful grasping results using a single-view partial point cloud as input. In order to facilitate this novel algorithm to practice, solutions for minimizing grasping uncertainty is illustrated in the next section.

3.5 Minimizing grasping uncertainty by using two 3D cameras

As was illustrated in section 3.2, using geometric properties of the target unknown object to obtain suitable grasps may bring about the risk of grasping failure as a result of the uncertainties induced from occlusions. For these scenarios, the robot has to move to obtain new perspective to obtain single-view point cloud data. In practice, the robots that have two cameras can be used to overcome grasp uncertainty resulting from occlusions. This can be achieved by following the six (A-F) procedures.

3.5.1 Building a “big” partial point cloud using two 3D cameras

Figure 3.15 shows a Baxter robot and PR2 robot which have two camera sensors separately installed at the robot head and robot hand. Using these types of robots, more completed contour of the target unknown object can be formed compared to single-view partial point cloud. After that, virtual exploration is carried out on the “big” partial point cloud to carry out searching of principle axis. Graspable candidates are allocated between the two camera sensor points, which can ensure the grasp candidates are allocated on the seen part of the target object. While constructing the “big” partial point cloud cannot result in much computing time, the increase of grasp security is achieved.

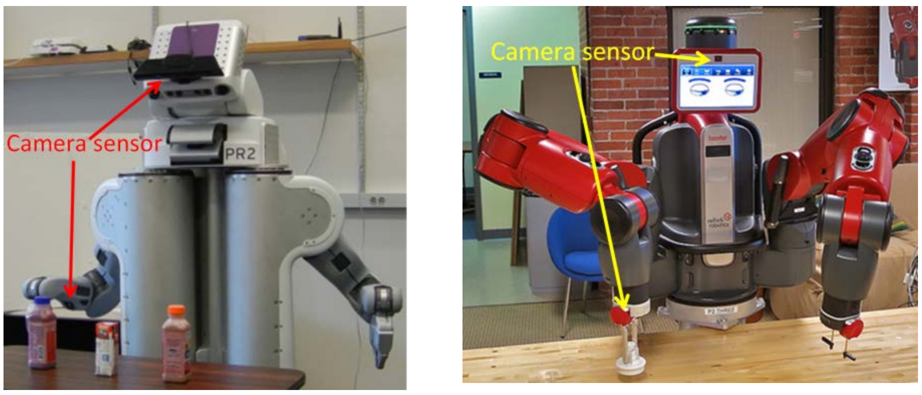


Figure 3.15: Two example robots which use two camera sensors.

The PR2 and Baxter robots in Figure 3.15 can be simplified as a head sensor and a hand sensor. The two 3D sensors are used to obtain the “big” partial point cloud. The green and blue point cloud in Figure 3.16 (b) respectively stand for a single-view partial point cloud acquired from the head sensor and the hand sensor. Next, the two single-view partial point clouds are fused together to obtain the “big” partial point cloud (shown as the brown point cloud in Figure 3.16 (b)). When we obtained the “big” partial point cloud, virtual viewpoints (shown as the black points in Figure 3.16 (c)) can be incrementally allocated around the principal axis between the head sensor point and the hand sensor point. Figure 3.16 (d) shows the corresponding point clouds for all virtual viewpoints.

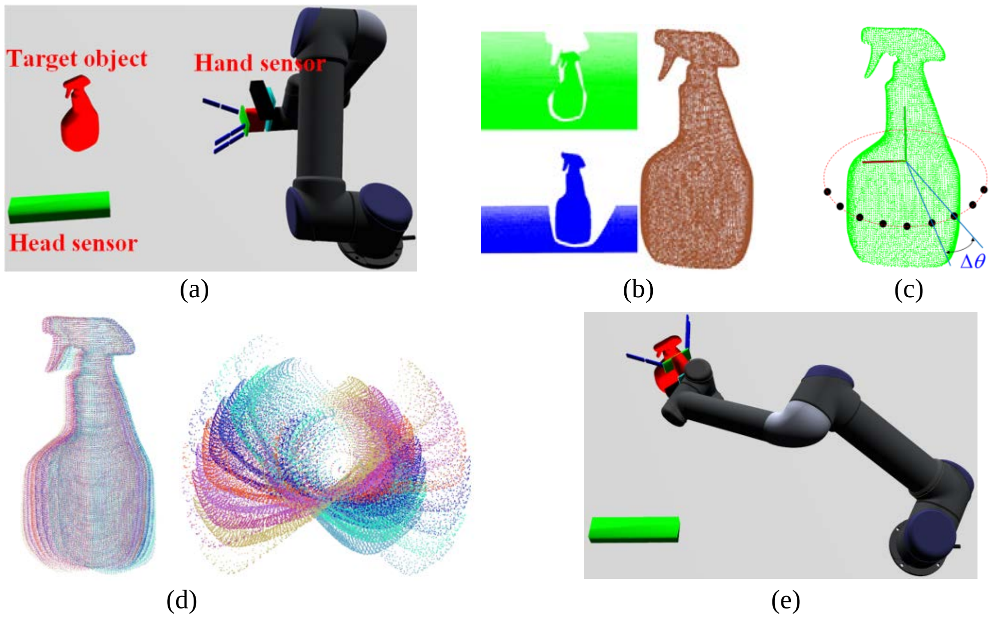


Figure 3.16: The method to overcome grasp uncertainty resulting from occlusions. (a) PR2 and Baxter robots are simplified as a head sensor and a hand sensor (b) two single-view partial point cloud from the two 3D sensors fused together (c) Virtual viewpoints incrementally allocated around the principal axis of the “big” partial point cloud. (d) the corresponding point clouds for all virtual viewpoints (e) an example of successful grasp execution.

Figure 3.17 illustrates the distribution of the camera sensors of the Baxter robot and the PR2 robot to obtain “big” partial point cloud. The green cuboid stands for the camera on the robot head and the black cuboid at the end of the robot arm represent the camera on the robot arm. The coordinate transformation between the head sensor coordinate system (HESCS) and the base link coordinate system (BCS) is defined as (T_{head}). The coordinate transformation between the hand sensor coordinate system (HASCS) and the base link coordinate system (BCS) is defined as (T_{hand}). T_{head} and T_{hand} can be obtained by Equation (3.7).

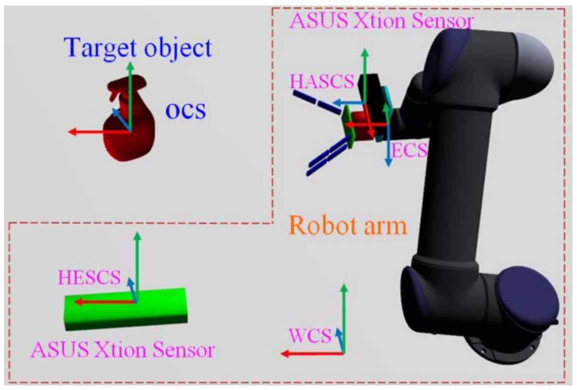


Figure 3.17: Coordinate transformation between two camera sensors and the word coordinate system (WCS).

$$\begin{cases} T_{head} = T_{\text{HESCS\_BCS}} \\ T_{hand} = T_{\text{HASCS\_ECS}} * T_{\text{ECS\_BCS}} \end{cases} \tag{3.7}$$

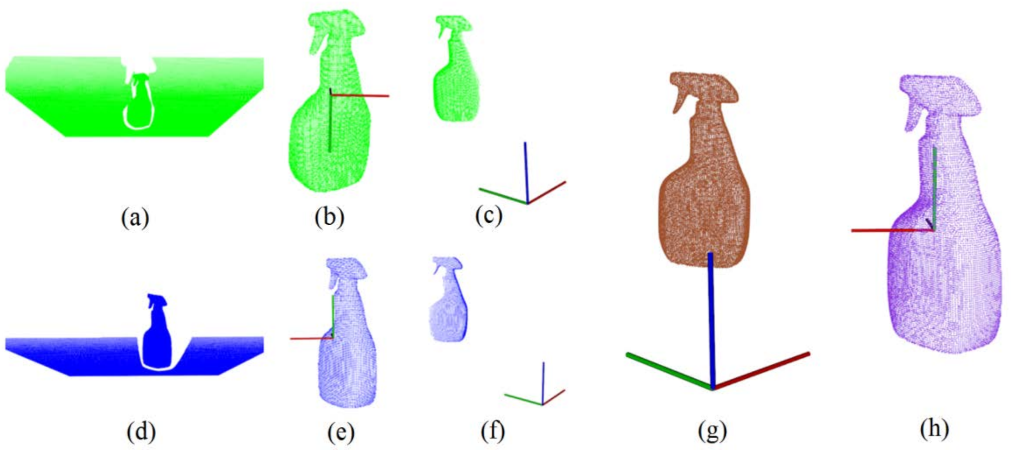


Figure 3.18: Acquisition of the "big" partial point cloud and transfer it from the world coordinate system (WCS) to the Object coordinate system (OCS). (a) and (d) are the single-view partial point cloud obtained from the head camera sensor and hand camera sensor. (b) and (e) are the two single-view partial point clouds of the target object in the camera coordinate system after removing the table plane. (c) and (f) are the two single-view partial point clouds transferred into world coordinate system (WCS). (g) is the registered "big" partial point cloud in the WCS. (h) is a "big" partial point cloud transferred from WCS to object coordinate system (OBS).

Figure 3.18 shows the procedures to acquire the "big" partial point cloud and transfer it from the world coordinate system (WCS) to the Object coordinate system (OCS). Figure 3.18 (a) and (d) respectively stand for the single-view partial point cloud obtained from the head camera sensor and hand camera sensor. Figure 3.18 (b) and (e) stands for the two single-view partial point clouds of the target object in the camera coordinate system after removing the table plane. Figure 3.18 (c) and (f) stand for the two single-view partial point cloud transferred into the world coordinate system (WCS). Figure 3.18 (g) is the registered "big" partial point cloud in the WCS. Figure 3.18 (h) shows the "big" partial point cloud transferred from WCS to object coordinate system (OBS).

3.5.2 Grasp allocation between the two camera sensor points

Figure 3.19 illustrates the searching strategies between the two camera sensor points. In this figure, each black point stands for a virtual viewpoint to carry out virtual exploration. The virtual viewpoints are allocated around the principal axis with a searching step ($\Delta\theta$). $q_{\theta 0}$ and $q_{\theta 1}$ stand for the hand and the head camera viewpoints, respectively. $q_{\theta i}$ means the i_{th} virtual viewpoint and $q_{\theta i} = q_{\theta 0} + i * \Delta\theta$.

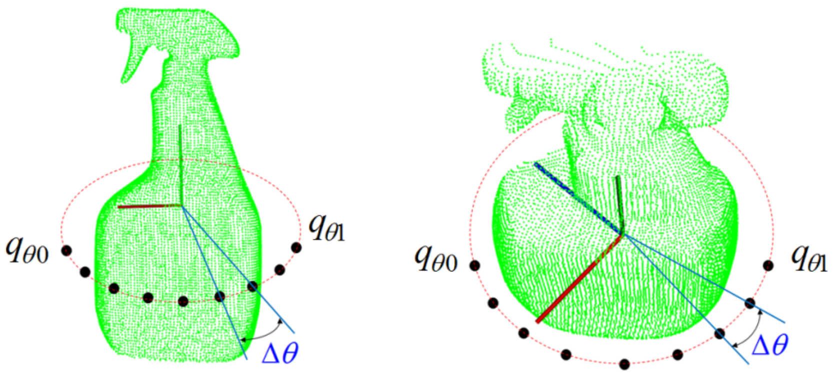


Figure 3.19: Searching strategies between the two camera sensor points.

3.5.3 Constructing virtual object coordinate systems

At each virtual viewpoint, a virtual object coordinate system (VOCS) is constructed for further analysis. Figure 3.20 illustrates the construction of the virtual object coordinate system. Specifically, the principal axis of the "big" partial point cloud in WCS is used to work as the Y axis of the object coordinate system (OCS). In Figure 3.20, Rp stands for a random point on the principal axis and Vp means a random virtual viewpoint. The X and Z axis can be obtained using Equation (3.8).

$$\begin{cases} \overrightarrow{O_cY} = \overrightarrow{O_cRp} \\ \overrightarrow{O_cX} = \overrightarrow{O_cVp} \times \overrightarrow{O_cRp} \\ \overrightarrow{O_cZ} = \overrightarrow{O_cX} \times \overrightarrow{O_cY} \end{cases} \tag{3.8}$$

Using the above method to go through every virtual viewpoint of Figure 3.19, we can obtain all VOCSs for all virtual viewpoints. Then the "big" partial point cloud in WCS is transferred to every local VOCS, all the transferred point clouds can be seen in Figure 3.21. Each color in Figure 3.21 corresponds to a transferred point cloud at the local virtual object coordinate system.

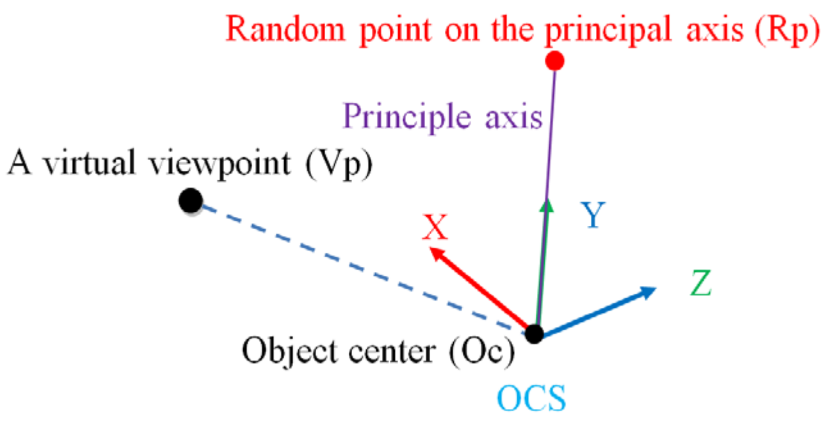


Figure 3.20: Schematical illustration of constructing a virtual object coordinate system (VOCS).

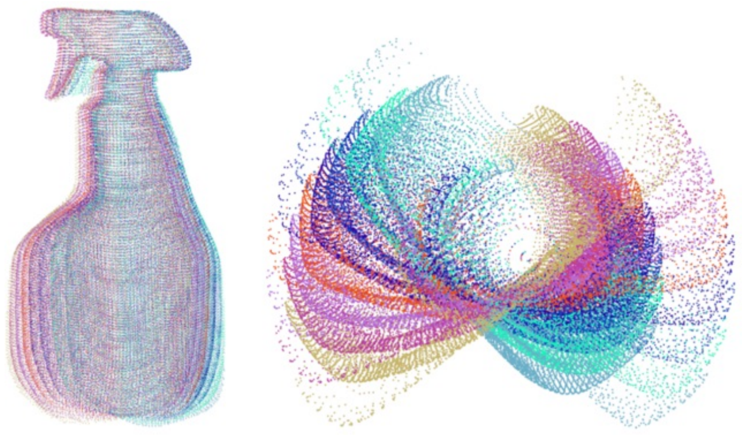


Figure 3.21: Transferred point clouds for all the virtual coordinate system (VOCSs).

3.5.4 Grasp allocation for a virtual viewpoint

The grasp configurations in the SE(3) group means many possibilities. To reduce grasp possibility to accelerate grasp searching, principal axis of the "big" partial point cloud is used to direct the grasp configurations. In Figure 3.22, the grasp configurations in the SE(2) group is simply a rotation around the principal axis. Each configuration can be expressed as $\Gamma[q_{yi} \mid q_{\theta_i}]$, in which y_i denotes grasp searching from the top of the "big" partial point cloud to the bottom of the "big" partial point cloud; θ_i means grasp searching between the two camera sensor points.

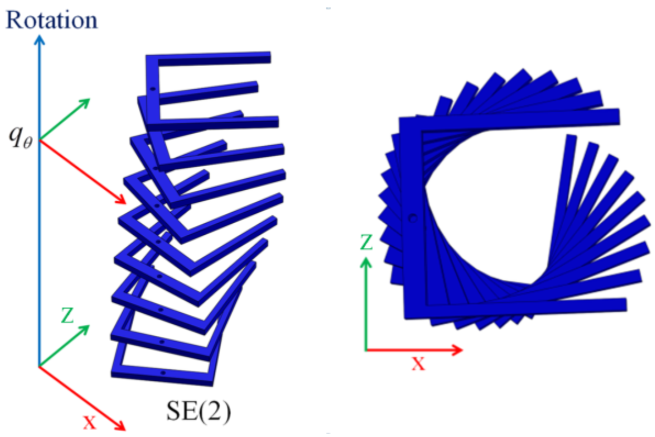


Figure 3.22: Grasping configuration in the SE(2) group.

All transferred point clouds for all VOCSs were already obtained shown in Figure 3.21. For each transferred point cloud $\mathrm{O}[q_\theta]$, grasp candidates are allocated from the top to the bottom, as can be seen in Figure 3.23. Let $\mathrm{O}[q_{yi} \mid q_\theta]$ $(i = 1, 2, \ldots, n)$ stand for the point cloud covered by i_{th} hand configuration, such that the red, blue and green rectangles respectively stand for three hand configurations. The corresponding point cloud covered by the three hand configurations can be expressed as $\mathrm{O}[q_{yi} \mid q_\theta]$, $\mathrm{O}[q_{yi+1} \mid q_\theta]$ and $\mathrm{O}[q_{yi+2} \mid q_\theta]$.

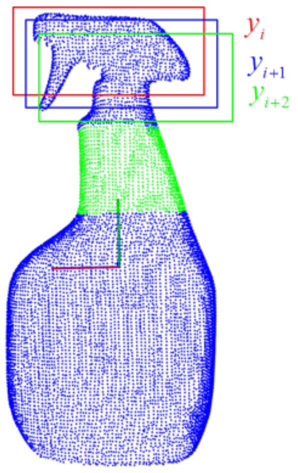


Figure 3.23: Hand configuration for a virtual viewpoint.

3.5.5 Force balance computation

For the point cloud $O[q_\theta]$ which corresponds to a virtual viewpoint, force balance computation can be divided into two parts, namely, force balance computation on the XOY plane (F_{xoy}) and on the XOZ plane (F_{xoz}). The application of using force balance to obtain the best grasping areas consisting of the following two steps.

The first step is using force balance computation on the XOY plane to select the best grasp for each virtual point cloud $O[q_{\theta h}]$ from top to bottom, i.e., $F_{XOY} = MAX\{F_{XOY}\{O[q_{yi} \mid q_\theta] \wedge O[q_{yi} \mid q_\theta] \in O[q_\theta]\}\}$. For every virtual viewpoint θ , force balance computation on the XOY plane is used to find the best grasp, and $yi\,(i = 1,2,3...,n)$ stands for the ith grasp candidate allocated along the Y axis.

Next step is using force balance computation on the XOZ plane to compare the best grasping areas deduced from the first step from left to right, i.e., $F_{XOZ} = MAX\{F_{XOZ}\{O[q_{ym} \mid q_{\theta h}]\}\}$, in which $\theta h\,(h = 1,2,3...,n)$ stands for the hth virtual viewpoint. $O[q_{ym} \mid q_{\theta h}]$ stands for the best grasp candidate of the hth virtual viewpoint. Force balance computation on the XOZ plane is carried out on the best grasping areas of every virtual viewpoint to choose the best grasp as final grasp execution.

A. Force balance computation on the XOY plane for each virtual viewpoint

As explained previously, force balance computation is firstly carried out to determine the best grasping areas for each virtual viewpoint. For a virtual viewpoint, the parameter (θ) corresponds to a virtual point cloud $O[q_\theta]$. The method in the step 6 of section 3.2 is used to find the best grasp for the virtual point cloud $O[q_\theta]$ on the XOY plane. Using the above method for going through all the virtual viewpoints, we can obtain all the best grasping areas for all virtual viewpoints. These best grasping areas can construct a grasp vector $G=(g_1,g_2...g_n)$, and g_n stand for the best grasp for the nth virtual viewpoint, and n means the total number of virtual viewpoints. Best grasping areas for all virtual viewpoints are shown as Figure 3.24. In the next subsection, the selection of the final grasp from the grasp vector $G=(g_1,g_2...g_n)$ is illustrated.

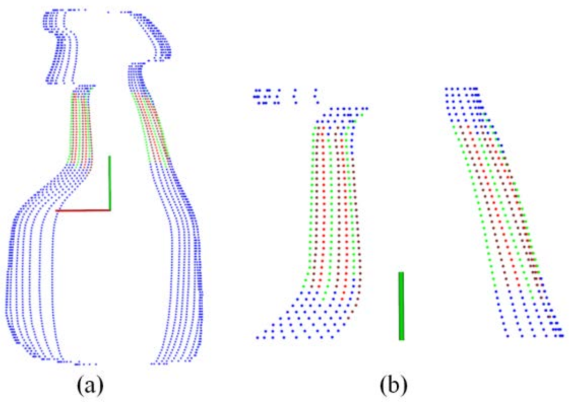


Figure 3.24: Results of force balance computation on the XOY plane for all virtual viewpoints. (a) best grasping areas for all virtual viewpoints (b) partial enlarged image.

B. Force balance computation on the XOZ plane for every virtual viewpoint

In order to choose the final grasp from the grasp vector $G=(g_1,g_2...g_n)$, the best grasping areas for every virtual viewpoint are extracted shown as Figure 3.25 (a) and (b). The colorful points in the black circle stand for all the grasps in the vector $G=(g_1,g_2...g_n)$. Figure 3.25 (c) shows an example grasp $g_i\,(1\le i\le n)$ in the vector G_j. The green points in Figure 3.25 (c) stand for points covered the g_i. To evaluate the grasp quality of g_i, force balance computation on the XOZ plane can be used. In order to compute force balance on the XOZ plane, the grasp g_i is first projected to the XOY plane to obtain the most left and most right contact area shown as the red points in Figure 3.25 (d). The left red points is numbered as $p_{l1},p_{l2},...,p_{lm}$ (m is the total number of left red points). The right red points is numbered as $p_{r1},p_{r2},...,p_{rn}$ (n is the total number of right red points).

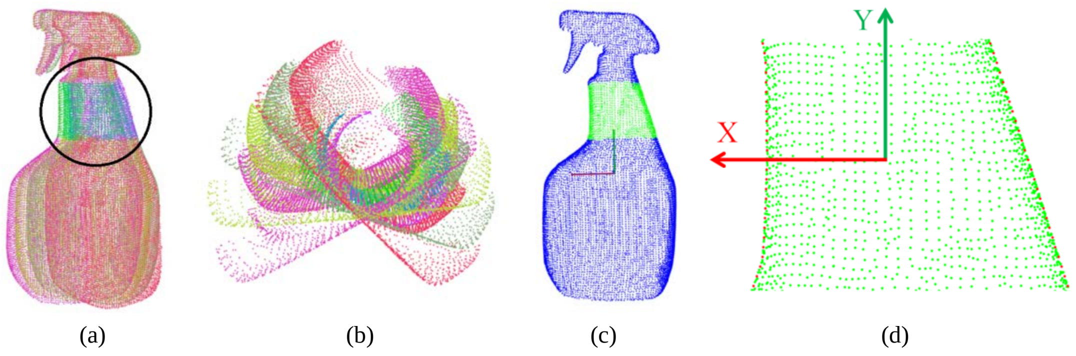


Figure 3.25: Best grasping areas for every virtual viewpoint are extracted to do force balance computation on the XOZ plane. (a) and (b) are the best grasping areas of all virtual viewpoints are extracted. Green points in (c) and (d) stand for an example of the achieved grasping area.

$$\begin{cases}(X_{lk} \le X_i \le X_{l(k+1)}) \,\|\, (X_{l(k+1)} \le X_i \le X_{lk}) \\ (X_{rk} \le X_i \le X_{r(k+1)}) \,\|\, (X_{r(k+1)} \le X_i \le X_{rk})\end{cases} \tag{3.9}$$

Using Equation (3.9), we can obtain the left and right contact area between the robot hand and the target object. By so doing a point of the grasp g_i is extracted when the X value of this point is located between two adjacent red points as shown in Figure 3.25 (d). The extracted left and right contact regions between the robot hand and the target object are shown as green points and purple points in Figure 3.26 (a) and (b). Then average Z values of the left and right contact region are worked out shown as Z_1 and Z_2 shown as Figure 3.26 (c). The difference (ΔZ) between Z_1 and Z_2 is used to evaluate the stability of this grasp (shown as Equation (3.10)). It is obvious to find that large difference (ΔZ) will lead to rotation of the object around the Y axis, which may lead to grasp failure. Using above method goes through all the grasp of vector $G = (g_1, g_2 ... g_n)$, we can obtain the force balance on the XOZ plane for all grasp candidates of vector G, that is vector $\Delta Z = (\Delta Z_1, \Delta Z_2 ... \Delta Z_n)$. A line graph (Figure 3.27) is drawn according to the vector $\Delta Z = (\Delta Z_1, \Delta Z_2 ... \Delta Z_n)$. We can predict that force balance on the XOZ plane reaches the best when virtual viewpoint θ_i comes to θ_4. Therefore, the fourth grasp g_4 of the vector $G = (g_1, g_2 ... g_n)$ is chosen as final grasp execution.

$$\Delta Z = |\overline{Z}_{left} - \overline{Z}_{right}| = |\frac{1}{m}\sum_{i=1}^{m} z_i - \frac{1}{n}\sum_{i=1}^{n} z_i| \tag{3.10}$$

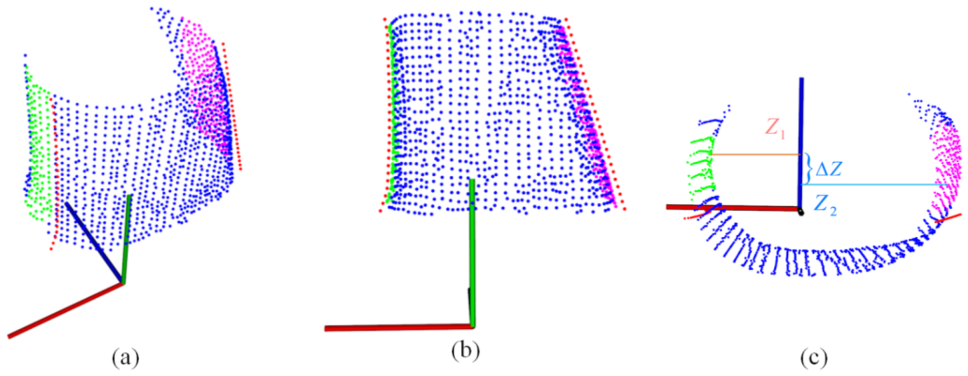


Figure 3.26: The difference between the left and the right contact region of the grasp is used to evaluate the grasp stability. (a) and (b) extracted left and right contact regions. (c) the average Z values of the left and right contact region.

3.5.6 Simulation for the “big” partial point cloud obtained by using two 3D sensors

In this subsection, simulations of using two 3D cameras to construct a “big” partial point cloud are conducted to obtain suitable grasping areas. Inspired by the Baxter and PR2 robots, we set two 3D cameras in the simulation setup. The black and green cuboids in the first column of Figure 3.28 respectively stand for the hand camera and head camera. Objects used to conduct simulations can be seen in the first column of Figure 3.28. Two single-view partial point cloud of the target object from the two 3D cameras are used to construct a “big” partial point cloud. Force balance computation is conducted on the “big” partial point cloud and the results of force balance computation are visualized as the second column and the third column of Figure 3.28.

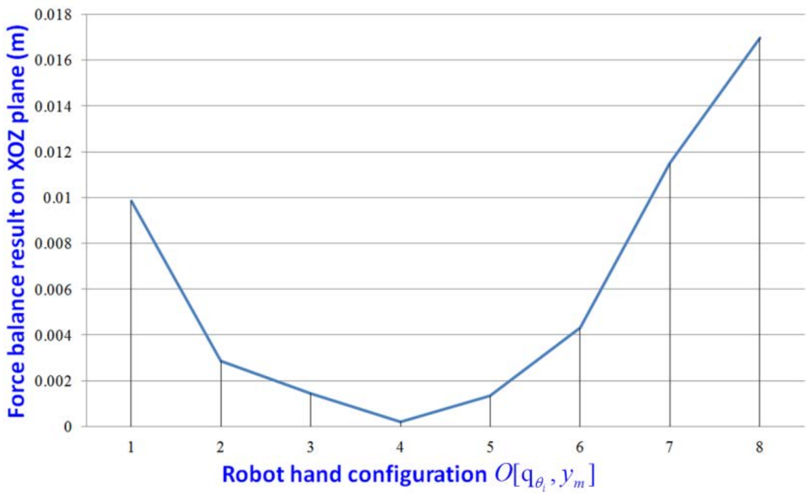


Figure 3.27: Results of force balance computation on the XOZ plane for all best grasping areas of every virtual viewpoint.

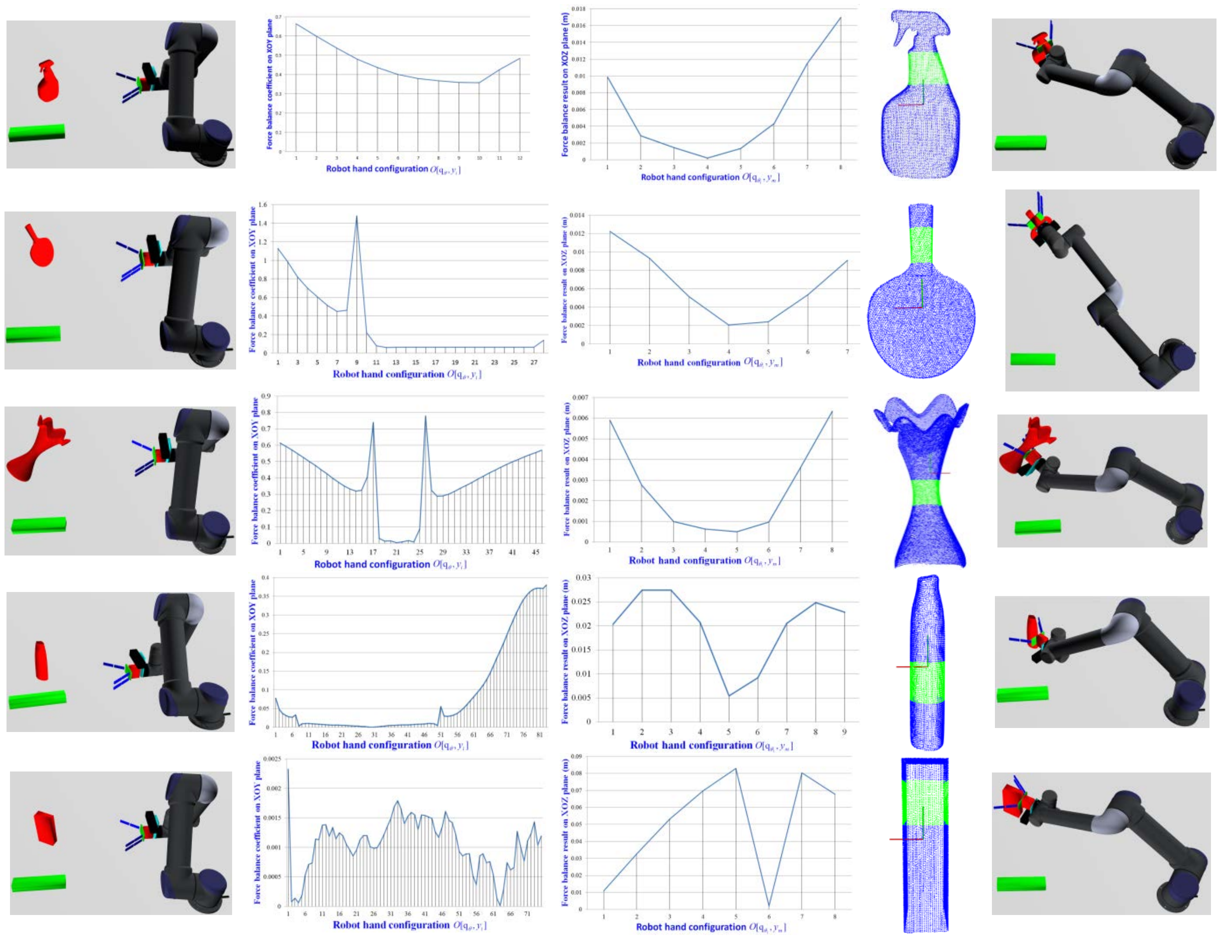

Force balance coefficient on XOY plane
Force balance result on XOZ plane (m)
Robot hand configuration $O[q_{\theta}, y_i]$
Robot hand configuration $O[q_{\theta_i}, y_m]$

Figure 3.28: Simulation results of “big” partial point cloud obtained using two 3D cameras.

Detailed results of force balance computation can be seen in Table 3.3. It demonstrates that appropriate force balances for all tested objects are obtained on both the XOY plane and the XOZ plane. The fourth column of Figure 3.28 shows the final grasp returned from our grasping algorithm. The last column demonstrates successful grasp execution for all target unknown objects. The fourth row of Table 3.3 shows the final grasps for all tested unknown objects. The third row of Table 3.3 shows the simulation results of force balance computation on both the XOY plane and the XOZ plane. We can find all the final grasps of the target unknown objects show expected force balance performances on box the XOY plane and XOZ plane. Thus, it ensures the stability of final grasp execution for all the tested unknown objects.

Table 3.3: Final grasp configuration and the corresponding force balance coefficient for the "big" partial point cloud obtained by using two 3D cameras.

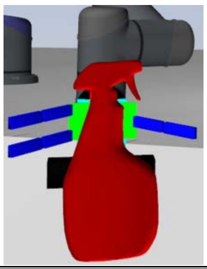
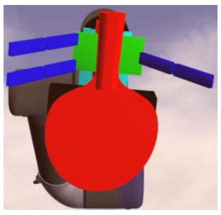
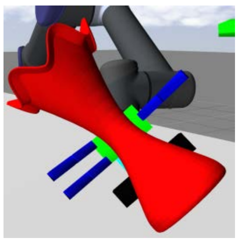
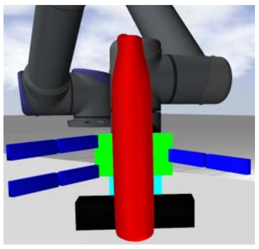
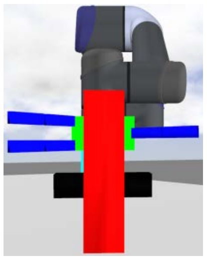
Force balance 1		Force balance 2		Force balance 3		Force balance 4		Force balance 5	
XOY	XOZ	XOY	XOZ	XOY	XOZ	XOY	XOZ	XOY	XOZ
0.3576	0.0002	0.0618	0.0021	0.0034	0.0005	0.0003	0.0054	8.5e-6	0.0021
									

Table 3.4 shows the grasp computing time of our grasping algorithm using a "big" partial point cloud. Even though the point cloud is based on a large number of points, our grasping algorithm can quickly process the "big" partial point cloud and output the final grasp within two seconds. This result demonstrates a faster grasping speed compared to the other researchers that use multi cameras [62] ore multi views [132-134].

Table 3.4: Grasp computing time of using two 3D cameras to construct a "big" partial point cloud.

Unknown Objects	Spray bottle	Table tennis racket	Vase	Shampoo bottle	Oatmeal box
points	9801	1358	19583	7059	11214
Time (s)	1.068	0.515	1.653	0.994	1.198

To sum up, the simulation tests to predict the grasping effectiveness in realistic situations are carried out using two 3D cameras. It demonstrated that the risk of grasping failures can be highly reduced in comparison with the application of single-view partial point cloud. While the grasping efficiency is maintained, our algorithm shows the improvements for robots to fast grasp unknown objects under unpredictable environments.

3.6 Conclusion

A novel algorithm of using principal component analysis for fast grasping unknown objects is proposed. For a single-view partial point cloud, graspable candidates are allocated along the principal axis from the top of the target object to the bottom of the target object. Force balance computation on both the XOY plane and the XOZ plane ensures the stability of the final grasping action. To illustrate the efficiency of our grasping algorithm, objects with different geometric shapes are used to conduct simulations and experiments. Both simulation and experimental tests demonstrated favorable performances of applying the algorithm. In addition, it shows that using our grasping algorithm the speed of grasping is greater than other algorithms for steady grasping.

In order to facilitate this algorithm in practice, the grasping uncertainty is minimized by taking the advantage of two cameras of the robot hardware. Virtual exploration on the “big” partial point cloud is carried out to determine the final grasp with the best force balance. The simulation results demonstrated that our grasping algorithm can quickly accomplish virtual exploration with steady grasping result. Therefore, this research demonstrates practical significance for increasing grasping speed and thus increasing robot efficiency under unpredictable environments.

Acknowledgement

This research was supported by the project: Factory in a day, from European Community’s Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 609206.

4

Object grasping by combining caging and force closure

This Chapter was published at the 2016 ICARCV conference:

Qujiang Lei, Martijn Wisse. 2016 IEEE International Conference on Control, Automation, Robotics and Vision (ICARCV), Phuket, Thailand.

Abstract

The current research trends of object grasping can be summarized as caging grasping and force closure grasping. The motivation of this chapter is to combine the advantage of caging grasping and force closure grasping to enable under-actuated grippers like the Lacquey gripper and the parallel grippers like the PR2 gripper to quickly grasp the flat unknown objects. Inspired by the idea that caging grasping generates finger points along the object's boundary and considering the geometry property of the grippers, we propose to allocate a discrete set of finger candidates along the object's boundary. Any two of the finger candidates can form a grasp candidate, which is analyzed by using force closure to choose the best grasp candidate as the final grasp execution. The grasp quality during the manipulation of the object is guaranteed by considering the gravity of the object. Simulations and experiments on an Universal arm UR5 and an under-actuated Lacquey Fetch gripper are used to examine the performance of this algorithm, and successful results are obtained.

4.1 Motivation

The motivation of this chapter is to quickly find suitable grasp for flat objects (shown as the Figure 4.1 (a)), specifically, this grasping algorithm is specially designed for under actuated grippers like the Lacquey gripper (shown as the Figure 4.1 (b)) or parallel grippers like the PR2 gripper (shown as the Figure 4.1 (c)). In order to enhance grasping stability, force balance and torque balance are taken into consideration. The stability is divided into two parts: one is the stability when the grasp action is being executed; the other is the stability while the object is being transported. These two parts of stability can ensure that the object is securely grasped during the whole process when the object is being grasped and manipulated. Inspired by [135] and [136], a novel grasping algorithm is proposed for flat unknown objects. [135] and [136] only concentrate on the objects themselves without considering the geometry property of the gripper. We are illuminated to combine the force closure grasping and caging grasping. In this chapter, we propose to consider both force and torque balance, as well as the geometry property of the robot gripper, for example, hand width and grasping range, when the robot tries to execute the grasp. Then the gravity of the object is considered when the robot tries to manipulate the object after it is grasped. This grasping algorithm has several advantages. First, it is simple to implement, which can lead to sound computational efficiency. Second, considering both force balance and torque balance and the geometry property of the gripper can ensure the grasp is executed successfully. Third, the grasping quality during the manipulation of the object is also guaranteed by considering the gravity of the object.

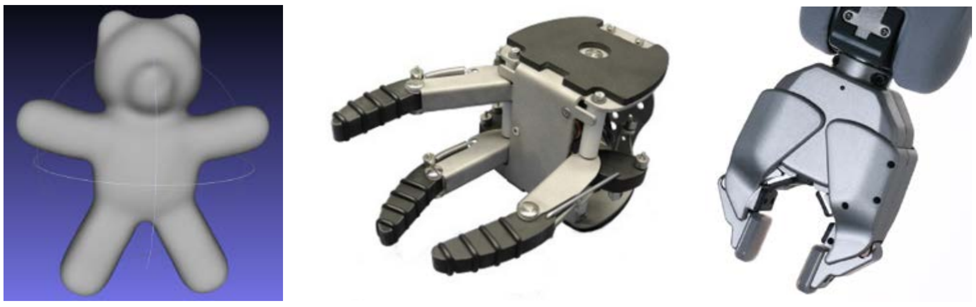


Figure 4.1: The motivation of this chapter, (a) shows an example of a flat object, (b) and (c) show the Lacquey gripper and the PR2 gripper respectively. The motivation of this chapter is to quickly find suitable grasp on flat objects for under-actuated grippers like the Lacquey gripper or parallel grippers like the PR2 gripper.

4.2 Introduction

Caging grasping is becoming increasingly popular in recent years. Since caging grasping was first introduced by [137] and [138], the analysis and synthesis of caging grasps has become an active research area. The basic idea of caging grasping is that the manipulators or fingers constitute a set of constraints in the object's configuration space that prevent it from escaping arbitrarily far. [139] proposed the first two-finger caging grasping for polygonal objects. Since the early works, many algorithms have been invented for finding two or three finger caging grasp for polygonal planar objects. [140] and [141] present comprehensive two-finger caging synthesis algorithms by formulating the caging grasping problem in the four dimensional configuration space of the two-finger hand. Afterwards, [135] formulates the caging set synthesis problem in two dimensional contact space which parameterizes the finger locations along the object's boundary. Several papers go further to consider the problem of planning and controlling the caging manipulation of an object by a team of mobile disc robots [142, 143, 144]. All above caging grasping algorithms by two/three-finger hand or by a team of mobile disc robots illuminate us to use a discrete of finger candidates allocating along the object's boundary to generate the grasp candidates. Figure 4.2 (a) and (b) show our inspiration of using caging grasping to generate grasp candidates.

Force closure grasping is a popular approach in the field of robotic grasping. Vast amount of research has been conducted in the domain of force closure grasping [99, 145]. Given the 3D meshed model of the target object and the friction coefficients, force closure grasping employs a grasp quality scoring function defined in terms of contact points and surface normal on the object to generate force stable grasp candidates [108, 146]. Force closure grasping confirms well with human's grasping synthesis. If given the 3D model of the target object, human can synthesize suitable grasp candidates by using the geometry information of the 3D model and the force closure requirement. Therefore, force closure grasping is a very promising method to solve the problem of unknown object grasping. Our previous works [123, 124] create a new method to compute force balance grasp directly on the partial point cloud of the target object.

[123, 124] do not require the 3D meshed model and the friction coefficients of the target object, which makes it more practical for unknown object grasping. The advantage of force closure grasping sheds illumination on using force balance and torque balance on robot grasping. Figure 4.2 (c) shows the inspiration of using force balance analysis on object grasping, and the idea of force balance searching from [123, 124] will be used in this chapter.

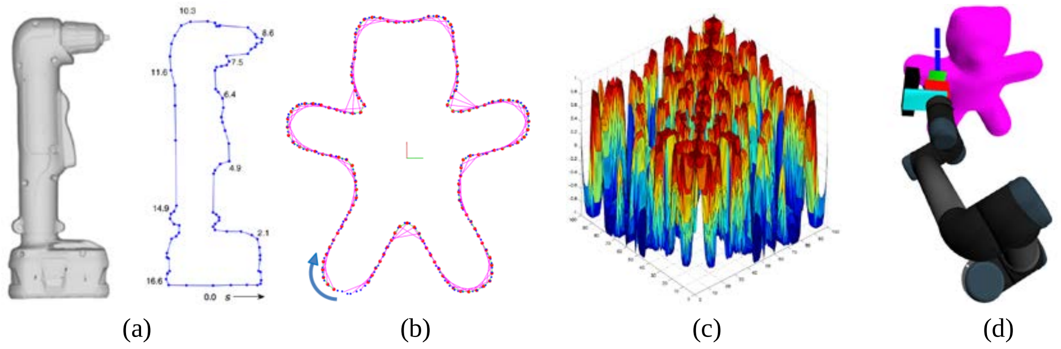


Figure 4.2: Inspiration of this chapter, (a) shows an image cited from [135] which uses caging method by introducing a discrete set of finger points allocating along the object's boundary to grasp the target object. (b) shows our inspiration. Inspiration from [135] promotes us to generate finger candidates (the purple lines) along the object boundary. (c) shows the result of force balance computation for all grasp candidates (one grasp candidate can be obtained by combining any two finger candidates in (b)). (d) shows the final grasp execution.

Inspired by the advantage of caging grasping and force closure grasping, we propose to use the method that caging grasping adopts to generate finger candidates along the object's boundary. After that, force closure analysis is employed to do force balance and torque balance computation. Specifically, force and torque balance computation is divided into two parts: one is the balance during the grasping execution; the other is the balance during the object manipulation after it is grasped. The purpose of the force balance and torque balance during the grasp execution is to assure that big movement and rotation will not occur. The aim of considering the force balance and torque balance during the manipulation of the object is to ensure that the possibility of the object sliding from the gripper is minimized. Grasping quality during the manipulation of the object is guaranteed by considering the gravity of the object.

This chapter is organized as following: section 4.3 contains a detailed explanation of our algorithm, section 4.4 shows the simulation results, section 4.5 demonstrates the experiment results, section 4.6 discusses the comparison between our algorithm and [135], section 4.7 is a conclusion of this chapter.

4.3 Detailed algorithm

This section contains a detailed explanation of the whole grasp algorithm. Subsection 4.3.1 shows how to borrow the idea from caging grasping to generate finger candidates. Subsection

4.3.2 demonstrates the details about how to use force closure analysis to work out good grasp candidates. Subsection 4.3.3 shows the gravity analysis.

4.3.1 Grasp candidates generation

The existing work about flat polygonal object grasping usually have a hypothesis, that is, the polygonal object is on the desk. In this chapter, we make a progress to enable the robot to find the main plain by employing the oriented bounding box (OBB), which can ensure that the robot finds the main plane on which to project the point cloud of the object. Even if the object is held in the air, the robot can find the polygonal contour of the object.

A. Construct the oriented bounding box

A teddy bear is used to explain our algorithm. Figure 4.3 (a) shows a virtual setup in a simulation environment. An eye-in-hand system is established by installing an Asus Xtion sensor at the end of the robot arm, which is used to capture the point cloud of the target object. After the point cloud of the target object (shown as Figure 4.3 (b)) is obtained, the Oriented Bounding Box (OBB) algorithm is used to find the main plane to project the point cloud.

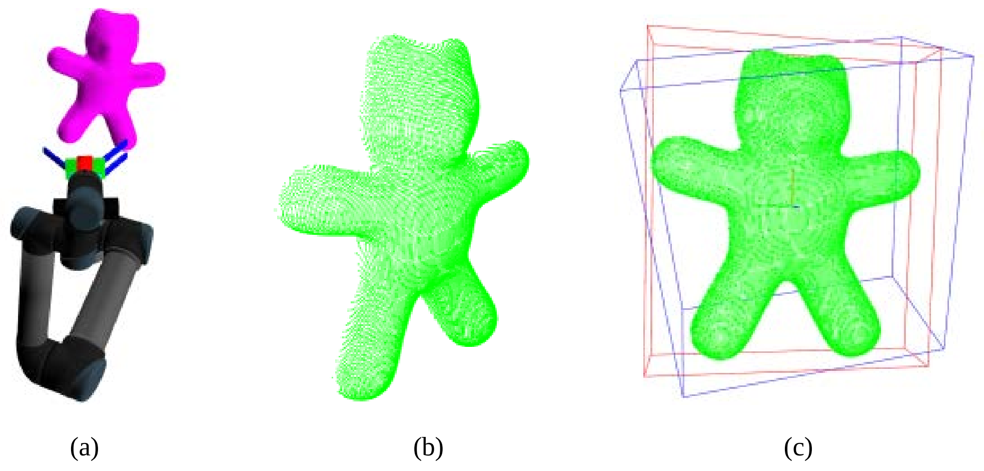


Figure 4.3: Construction of the oriented bounding box (OBB), (a) shows the virtual setup in simulation environment, an Asus Xtion sensor is installed at the end of the robot arm. (b) shows the point cloud acquired by the Asus Xtion sensor. (c) shows the OBB box, the red rectangular frame stands for the OBB box, the blue rectangular frame represents the axis-aligned bounding box (AABB).

There are two ways to obtain a bounding box, that is the axis-aligned bounding box (AABB) and the oriented bounding box (OBB). The axis-aligned bounding box for a given point set is its bounding box subject to the constraint that the edges of the box are parallel to the Cartesian coordinate axes. The oriented bounding box is the bounding box calculated subject to no constraints as to the orientation of the result. By using the eccentricity and moment of inertia, a position vector and a rotation transform matrix can be obtained. And then, each vertex of the given AABB must be rotated with the given rotation transform matrix and then positioned to get the OBB. The blue and the red rectangular frames in Figure 4.3 (c) respectively stand for the Oriented Bounding Box and the axis-aligned bounding box. We can easily find that the oriented bounding box is more generous and better suitable for the grasping purpose.

B. Project the point cloud to the main plane of the OBB

A local object coordinate system can be established by using the Oriented Bounding Box shown in the Figure 4.4 (a). The red, green and blue lines respectively stand for the X, Y and Z axis of the local object coordinate system. Then the point cloud is projected to the XOY plane (the main plane) to obtain the main silhouette of the object shown as Figure 4.4 (b). The concave hull (Figure 4.4 (c)) of the projected point cloud is extracted to work as the main silhouette of the target object. The points making up the concave hull are conveniently stored in serial order for later processing. As can be seen from Figure 4.5 (b), which is an enlarged image of the red rectangle in Figure 4.5 (a), the points are stored in serial order.

C. Finger candidate generation

After the main silhouette of the target object is obtained, finger candidates need to be first generated to do further analysis. The specific procedures to generate finger candidates are as follows.

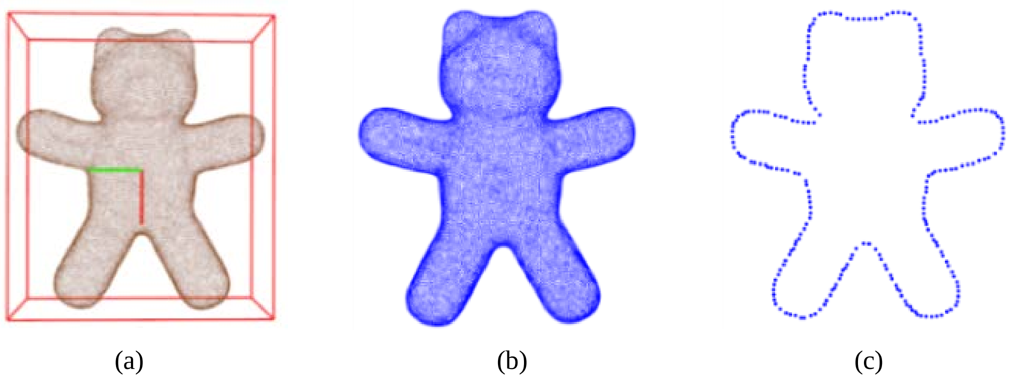


Figure 4.4: Abstraction of the object contour. (a) shows the point cloud in the OBB box. (b) shows the point cloud projected to the main plane of the OBB. (c) shows the object boundary acquired by abstracting the concave hull of (b).

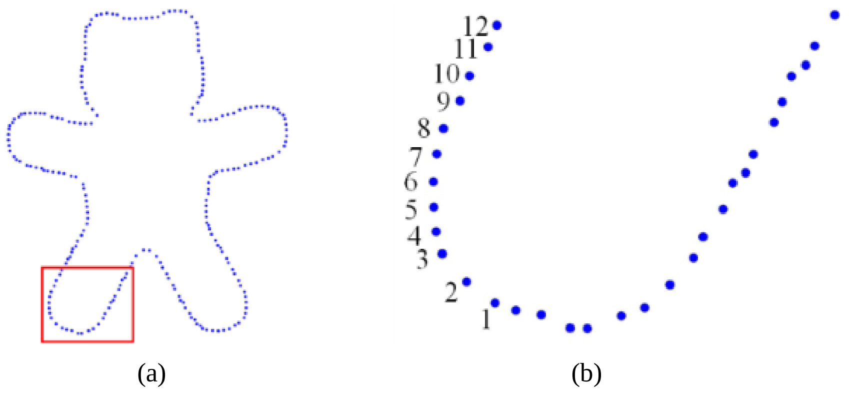


Figure 4.5: The points on the concave hull of the object is stored in serial order.

C.1 Search step points

Employing the property that all the boundary points are in serial order, two adjacent boundary points can be connected to form a polygon. Figure 4.6 (a) shows the corresponding partial polygon for the boundary points in Figure 4.5 (b). Two adjacent points in Figure 4.6 (a) are connected by an orange line. As it is can be seen from Figure 4.6 (b), a step distance (r) is used to work as the search radius. The distance between point 1 to point n is defined as d_{1_n}, the distance between point 1 to point n+1 is defined as d_{1_n+1}. if d_{1_n} and d_{1_n+1} satisfy one of the Equation (4.1), an intersection point can be found to work as the step point. If several intersection points are found at the same time, the point with the minimum serial number is chosen as the step point. In another word, if there are m intersection points, the $\min(n_1, n_2, ..., n_m)$ will be chosen as the step point. Figure 4.7 shows the result of step point searching. The blue and red points respectively stand for the boundary points and step points.

$$\begin{cases} d_{1\_n} < r < d_{1\_n+1} \\ d_{1\_n+1} < r < d_{1\_n} \end{cases} \tag{4.1}$$

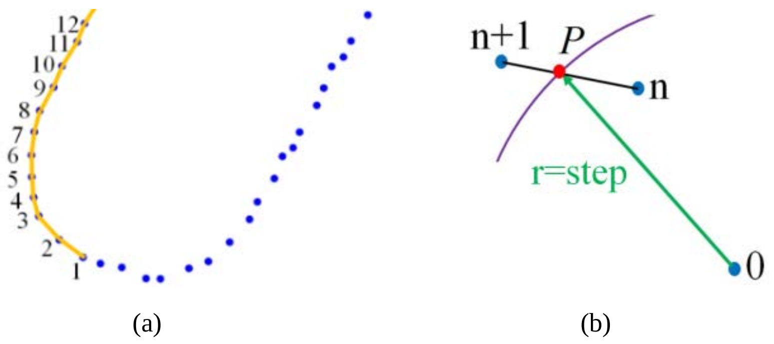


Figure 4.6: The step point searching process. (a) shows the orange polygon formed by connecting two adjacent boundary points. (b) shows how to compute the step points.

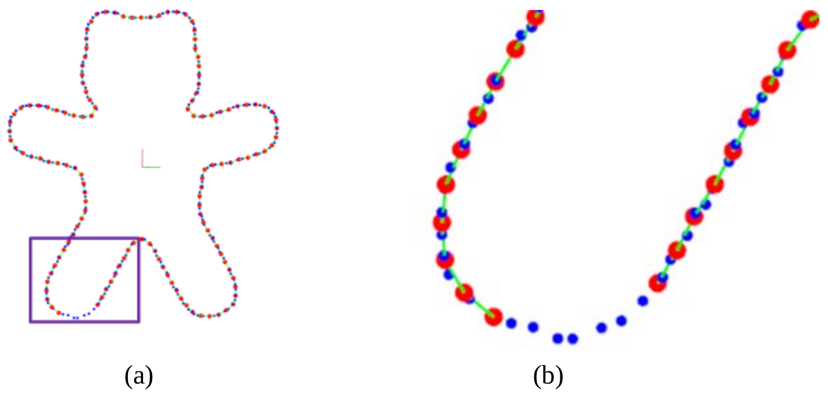


Figure 4.7: The result of step point searching. The blue points are the boundary points and the red points are the step points. (a) shows all the step points and (b) is an enlarged image of (a).

C.2 Obtain the finger candidates

After all the step points are obtained, the finger width is taken into consideration. In subsection C.1, if the step point is located on the line between point n (P_n) and point n+1 (P_{n+1}), a point cloud Ω can be constructed by adding P_n, P_{n+1} until the last point of the concave hull boundary in Figure 4.5 (a). w_f is used to describe the width of the finger, and the method of finding step points in Figure 4.6 can be employed to find the end point of the finger candidate. The step point works as the start point of the finger candidate and w_f works as the searching radius. An intersection point can be found on Ω by using the start point and the searching radius w_f. The line between the start point and the end point stands for a finger candidate. Figure 4.8 shows all the finger candidates. Every purple line in Figure 4.8 represents a finger candidate.

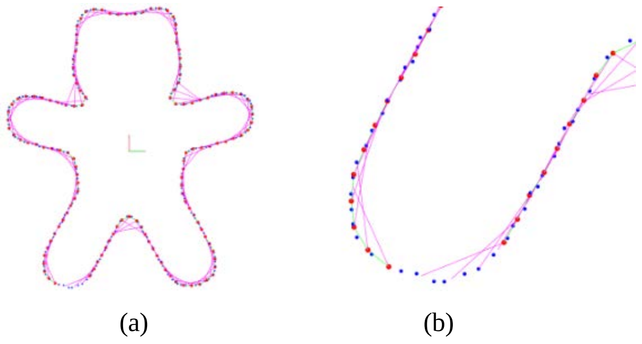


Figure 4.8: The result of finger candidate computation, every purple line stands for a finger candidate. (a) shows all the finger candidates, (b) is an enlarged image of (a).

C.3 Obtain the grasping direction for finger candidates

After the finger candidates are obtained, the first thing need to be done is to find the grasping direction. For every finger candidate, there are two possible grasping directions shown as Figure 4.9 (a). The blue line and the orange line respectively demonstrate the inside and outside grasping direction. Figure 4.9 (b) shows random grasping directions for all finger candidates, some lines are toward inside the object, some others are toward outside the object. How to find all the inside grasping direction?

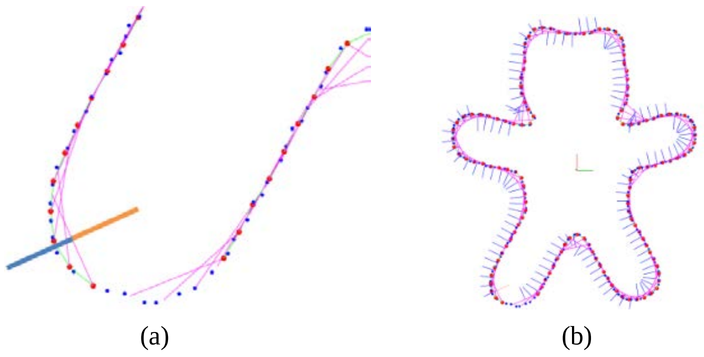


Figure 4.9: There are two possible grasping directions for a finger candidate. (a) shows two grasping direction (the orange line and the blue line) for a finger candidate. (b) shows all the random grasping direction for all the finger candidates.

In order to solve the problem of finding correct grasping direction, let's first look at how to judge whether a given point is inside or outside the contour of the object. First, the contour of the target object is used to construct a polygon (Φ). An effective way to find whether a given point is inside or outside a polygon is to cast many random rays from the given point to any direction. The intersects between the casting ray and the polygon are used to judge whether the given point is inside or outside the polygon. If the number of intersects is an odd number, the given point lies inside the polygon, otherwise, it lies outside the polygon. Figure 4.10 shows the idea of how to judge whether a give point is inside or outside of a polygon. The red point and the orange point are two given points. The green and the purple points are two random points. The line between the red point and the green point has two (even number) intersects with the polygon. The line between the orange point and the purple point has three (odd number) intersects with the polygon. Specifically, a given point (P_g) is first given, and then, a controlled number of random points (P_{rn}, $n = 1,2,...,m$). m is the total number of the random points) are generated by system, A straight line (l_{g_r1}) can be constructed by connecting the given point (P_g) and the first random point (P_{r1}). l_Φ is used to represent one side of the polygon (Φ). The intersect point between l_{g_r1} and the l_Φ will be found. If the intersect point is on the polygon, it means there is a real intersect point. A for-loop is used to go through all the sides of the polygon (Φ) to find all the intersection points. The number of the intersects between l_{g_r1} and the polygon (Φ) is defined as n_1. Then, the second line l_{g_r2} can be constructed by connecting another random point (P_{r2}) and the given point (P_g). The number of intersects between l_{g_r2} and the polygon (Φ) is defined as n_2. The line between P_g and P_{rm} is defined as l_{g_rm}, the number of intersects between l_{g_rm} and the polygon (Φ) is defined as n_m. If all the numbers ($n_1, n_2 \dots\ n_m$) are odd numbers (that is, $(n_1, n_2,..., n_m)\%2 = 1$), the given point is inside the polygon (Φ), otherwise it is outside the polygon.

After we known how to judge whether a given point is inside or outside of a polygon, we can use it to find the grasping direction for the first finger candidate. As shown in Figure 4.11 (a), a step value (δ) is used to do step searching along the middle vertical line of the first finger candidate. A pair of step points are set along the green arrow and the black arrow with the step of δ. Then, above method can be used to judges whether the two step points are inside or outside of the object contour. If the two step points are both inside the object boundary, then the algorithm continues to search along the direction of the green arrow and the black arrow. The step searching process stops until one step point is inside the object boundary and the other step point is outside the object boundary. The direction from the middle point of the first finger candidate to the step point inside the object boundary is used to work as the grasping direction.

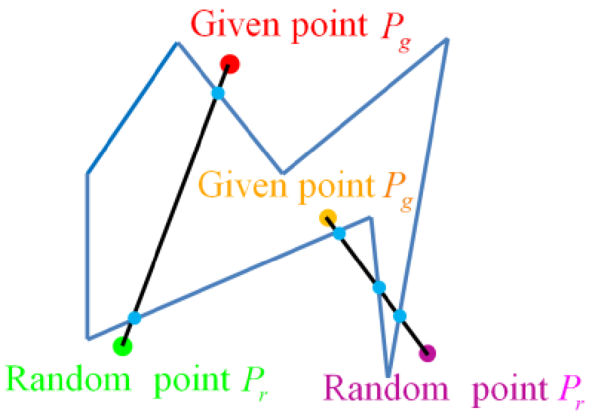


Figure 4.10: The method to judge whether a given point is inside or outside a polygon. If a given point is outside a polygon, the line between the given point and the random point has even number of intersects with the polygon. If a given point is inside a polygon, the line between the given point and the random point has odd number of intersects with the polygon.

After the grasping direction of the first finger candidate is obtained, a coordinate system can be established (seen as Figure 4.11 (b)). The middle point of the finger candidate works as the origin, the direction from the start point of the finger candidate to the end point of the finger candidate works as the X axis. The Z axis is vertical to the main plane. If Y_1 is the grasping direction, the grasping direction is the cross product of X and Z, that is, $Y_1 = X \times Z$. If Y_2 is the grasping direction, the grasping direction is the cross product of Z and X, that is, $Y_2 = Z \times X$. The grasping direction of other finger candidates can be worked out by using the same cross product. Figure 4.11 (c) shows inside grasping directions for all finger candidates.

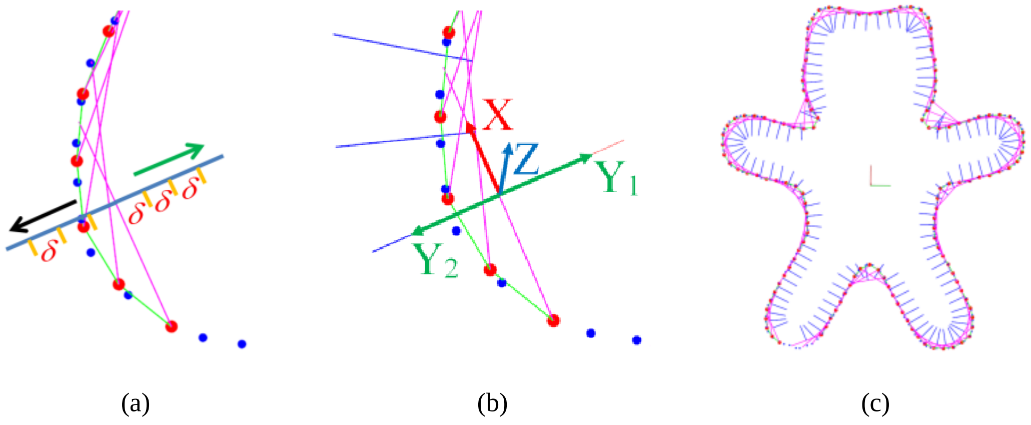


Figure 4.11: Grasping direction searching process. (a) shows how to find the grasping direction for the first finger candidate. (b) shows how to use the grasping direction of the first finger candidate to build a cross product, which can be used to work out the grasping direction of the rest finger candidates. (c) shows grasping direction for all the finger candidates.

4.3.2 Force closure analysis

After the grasping directions for all finger candidates are worked out, any two finger candidates can form a grasp candidate. Force closure analysis is used to do further analysis. Specifically, force balance and torque balance are used to do balance computation to choose the stable grasp candidates. Then, grasping range is considered to remove grasp candidates of which the distances between the two grasp sides are bigger than grasping range. Afterwards, the local geometry property of the grasp candidates is considered to remove those grasp candidates with big variance, which may lead to grasp failure. Then, the operability analysis is used to remove those grasp candidates of which the robot gripper may collide with the object when the robot tries to grasp it.

A. Force balance computation

After the grasping directions for all finger candidates are worked out, any two finger candidates can form a grasp candidate. For every grasp candidate, force balance computation is used to analyze the resultant force applied on the object. If the total number of the finger candidates is m, f_i ($i = 1, 2, ..., m$) is used to represent the i_{th} finger candidate and F_i is used to stand for the force applied on the object by f_i. If the force along the grasping direction for every finger candidate is a unite force, the angle between the two unite forces can represent the intensity of the resultant force. In Figure 4.12, the orange line and the red line respectively stand for the grasping direction for two example finger candidates (f_i and f_j). The angle (γ_{ij}) is used to describe the intensity of the resultant force of F_i and F_j. γ_{ij} is used to evaluate the stability of the grasp candidate consisting of f_i and f_j. If i and j go from 1 to m, the results of force balance computation for every grasp candidate can be obtained (as Figure 4.13 (a) shows). The red and blue areas of Figure 4.13 (a) respectively represent the maximum resultant force and the minimum resultant force. Figure 4.13 (b) is the projected image of Figure 4.13 (a), where we can clearly see that the resultant force is maximum when it satisfies $i = j$, that is the area between the two green parallel lines. The bigger the resultant force is, the more unstable the grasp is. The centers of the blue circles in Figure 4.13 (b) stand for the minimum resultant force. At these center points, the resultant force is almost zero, which means these center points correspond to the most stable grasp candidates.

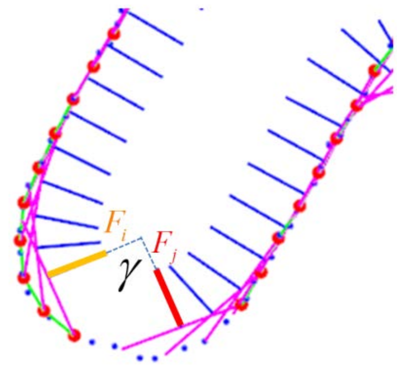


Figure 4.12: Force balance computation, the orange line and the red line respectively stand for the grasping direction for the finger candidate f_i and f_j. The angle (γ_{ij}) is used to describe the intensity of the resultant force of F_i and F_j. The smaller γ_{ij} is, the more stable the grasp is.

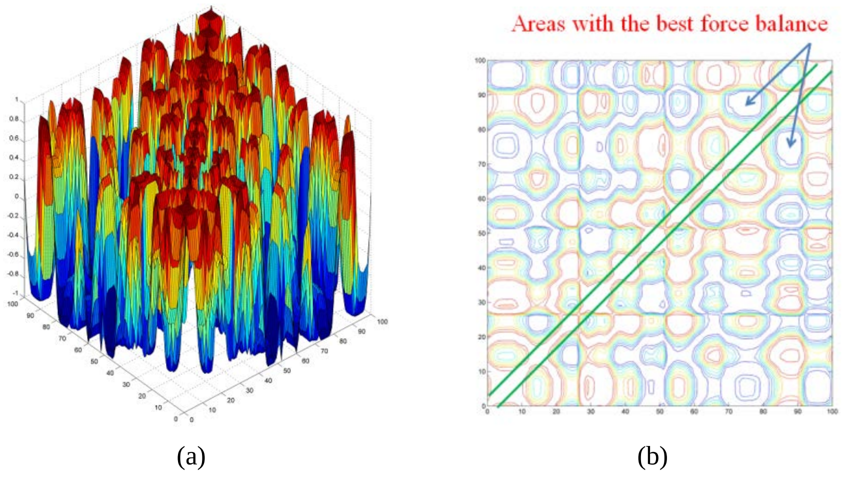


Figure 4.13: The result of force balance computation. (a) shows the result of force balance computation, the red areas mean the maximum resultant force and the blue areas stand for the minimum resultant force. (b) is the projected image of (a).

B. Torque balance computation

After the above steps, the grasps candidates $g_{i,j}$ (consisting of f_i and f_j) satisfying the force balance requirement set by the system are chosen out. However, only use of force balance cannot make sure the grasp stability. Figure 4.14 shows an example grasp on the bear's head, which satisfies the force balance requirement. However, if the robot tries to grasp the bear using this grasp configuration, the bear would rotate around the green point, which may lead to grasp failure. Therefore, the torque of every grasp $g_{i,j}$ should be taken into consideration. $T_{i,j}$ is used to stand for the torque of a grasp candidate $g_{i,j}$. A function is used to represent the relation between $T_{i,j}$ and $g_{i,j}$, that is $T_{i,j} = f(g_{i,j})$. If $T_{i,j}$ is bigger than the threshold (T_s) set by the system, then the grasp $g_{i,j}$ is removed, otherwise, $g_{i,j}$ is kept. All the grasp candidates left are used to do following analysis.

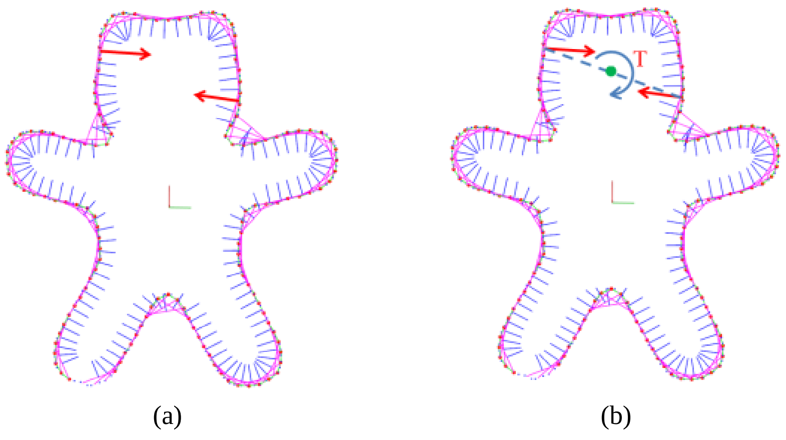


Figure 4.14: Torque balance analysis. (a) shows a possible grasp candidate satisfying the force balance requirement. (b) shows the torque balance analysis. Big torque may lead to big rotation of the object, which may result in grasping failure.

C. Grasping range computation

After the force balance and torque balance computation, grasping range of robot hand should be considered. If the distance $d_{i,j}$ between the two grasp sides of a grasp candidate $g_{i,j}$ is bigger than the grasp range, the robot cannot grasp the object. Therefore, if the distance $d_{i,j}$ of the two grasp sides of every grasp $g_{i,j}$ is smaller than the grasping range, the grasp is remained, otherwise it is removed.

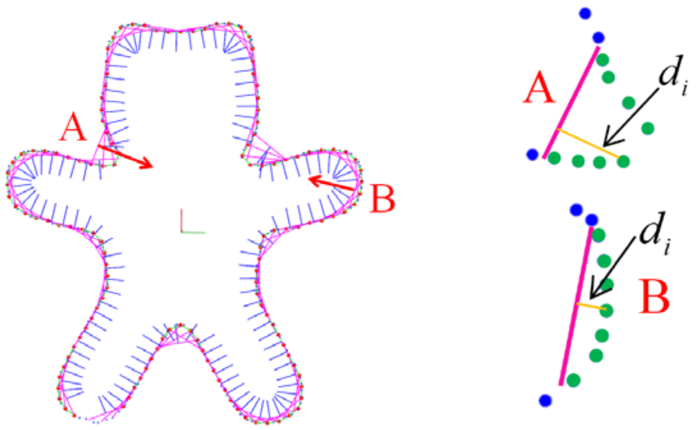


Figure 4.15: Variance analysis. The left image shows a possible grasp candidate. The right images are enlarged images of the two grasp sides. Variance of the points of the two grasp sides is used to evaluate the grapping quality.

D. Variance analysis

After finishing all steps mentioned above, the grasps left satisfy the force requirement, torque requirement and grasping range requirement. However, the local geometry property of grasp candidates is not yet considered. Figure 4.15 shows a grasp candidate, right up and right down are the enlarged images of the two grasp sides (the green points). The distance between one green point to the purple line is defined as d_i, $0 < i \le n$, n is the total number of the green points. All the distances are added together to get the variance v of the grasp, $v = \sum_{i=1}^{i=n} d_i$. The bigger the variance is, the larger possibility of grasp failure is. If the variance is smaller than the threshold set by the system, the grasp is saved, otherwise, it is removed.

E. Operability analysis

Operability in this subsection means whether the grasps found in above subsection can be executed or not. Not all grasp candidates obtained by above steps can be executed successfully, Figure 4.16 shows an example, the two purples lines stand for a grasp candidate, the two orange lines represent the biggest open width of the robot hand. For the example grasp candidate, the robot finger will collide with the bear at the red circle. Therefore, it is necessary to analyze the operability of grasp candidates. In order to simplify computation, a local coordinate system is established and the concave hull boundary of the object is transferred into the local coordinate system. The biggest open width of the robot hand is defined as w_0 and the hand width is defined as w_h, the distance between the two grasp sides is defined as d. If the points on the concave hull boundary satisfy Equation (4.2), then it means there is collision when the robot try to execute this grasp, otherwise, there is no collision. Using the above steps repeatedly, we can find all the grasp candidates satisfying the operability requirement.

$$\begin{cases} -0.5 * w_h \le x \le 0.5 * w_h \\ -w_o \le y \le -0.5 * d \,||\, 0.5 * d \le y \le w_o \end{cases} \tag{4.2}$$

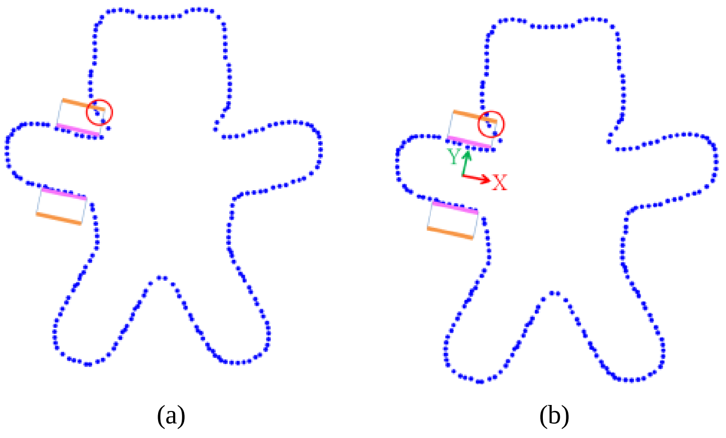


Figure 4.16: Grasping operability analysis. (a) shows an example grasp candidate that the robot finger will collide with the object. (b) shows the local coordinate system which is used to do point cloud transformation.

4.3.3 Gravity analysis

When an object is under manipulation after it is grasped, its gravity inevitably brings instability to the grasp. How to take the gravity of the object into consideration is a key problem which can decide whether a grasp action is reliable or not. In this chapter, we propose to use the distance between the gravity center and the grasping line (between the two grasping points) to evaluate the grasp candidates. For example, if the robot already grasped the teddy bear and the robot wants to move the bear in the air. At this moment, the gravity needs to be considered to prevent the object falling from the robot gripper. Figure 4.17 (a) shows an example grasp, specifically, the two purple lines stand for the grasp and the red point represents the gravity center. If the robot grasps the teddy bear and moves in the air, the object may rotate around the red line, the corresponding torque is defined as T, T can lead to instability which may result in grasp failure. The distance (d) between the gravity center and the grasping line is used to evaluate the grasp quality after the object is grasped. The shorter the distance is, the smaller the torque is. The smaller the torque is, the more stable the grasp is. Figure 4.17 (b) shows the result of gravity analysis, the grasp with the smallest gravity torque is chosen as the final grasp (shown as the two bold red lines in Figure 4.17 (b)).

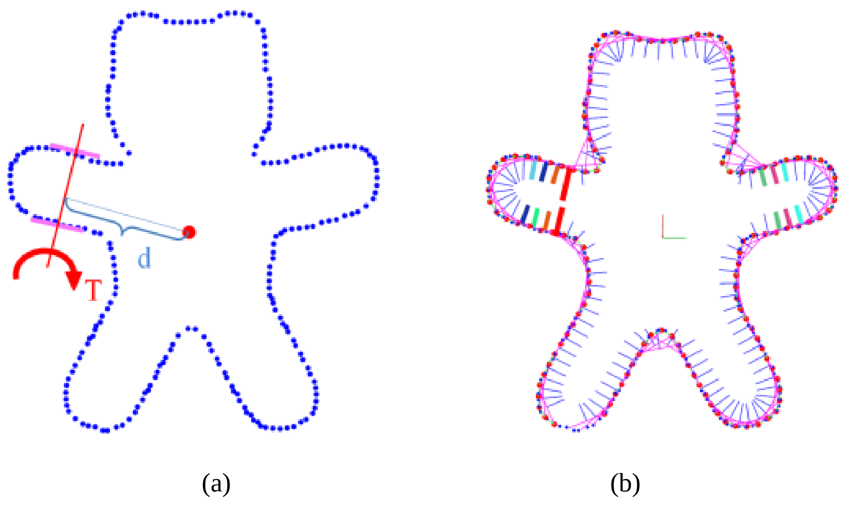


Figure 4.17: Gravity analysis. (a) the distance (d) between the gravity center and the grasping line is used to evaluate the effect of the object's gravity. (b) the final grasp obtained.

4.4 Simulation

In order to test the algorithm, various objects are chosen to conduct simulation to determine the grasping performance. The simulation system consists of Robot Operating System (ROS), Gazebo (a Standalone Open Dynamics Engine based simulator) and MoveIt! (a state of art software for mobile manipulation, incorporating the latest advances in motion planning, manipulation, 3D perception, kinematics, control and navigation). In the Gazebo simulation environment, A Lacquey under-actuated gripper and an Asus Pro Live sensor are installed at the end of the Universal arm (UR5). The Asus Pro Live sensor is used to acquire the point cloud of the target object in the simulation environment. The Lacquey under-actuated gripper is used to execute the final grasp found by the algorithm.

Five objects with different geometry shapes are used to do simulations. These objects are a teddy bear, an electric drill, a pistol, a spray bottle and a pan. Figure 4.18 shows the simulation results. The first column shows the simulation setup. The second column shows the OBB box to process the point cloud. The third column shows the finger candidates and the grasping directions. The fourth column shows the final grasp found by the algorithm. The two bold red lines stand for the final grasp. The fifth column shows the grasp area on the point cloud of the target object. The sixth column shows the grasp execution. The algorithm can find good grasp for all these tested objects, which proved the effectiveness of this algorithm.

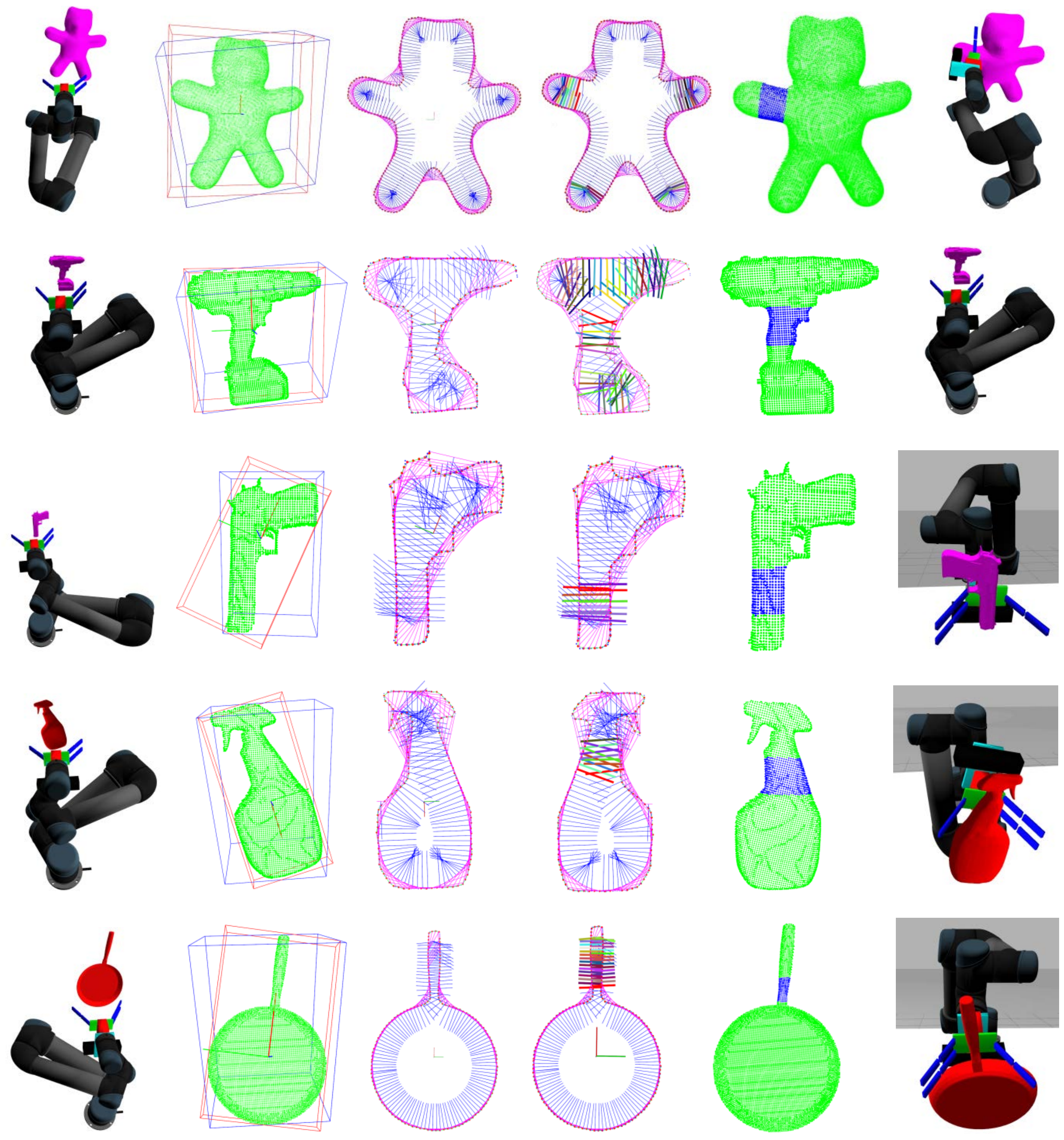


Figure 4.18: Simulation results.

4.5 Experiment

The experiments are conducted using a six degrees of freedom Universal arm UR5 and an under-actuated Lacquey Fetch gripper. An Xtion pro live sensor is installed on the tool tip of the robot. The whole experiment setup can be seen in Figure 4.19. Five objects with different geometry shapes are used to do experiment. These objects include an electric drill, a spray bottle, a hammer, a pan and a juice box. Figure 4.20 shows some snapshots of the grasping process of these objects. The first column is the initial state of the robot and the target objects. The second column is result of grasping computation, the two red lines stand for the final grasp found by this algorithm. The third column shows the grasp area on the point cloud of the target object. The fourth column shows the gripper arriving at grasping point. The fifth column shows objects grasped by the gripper.

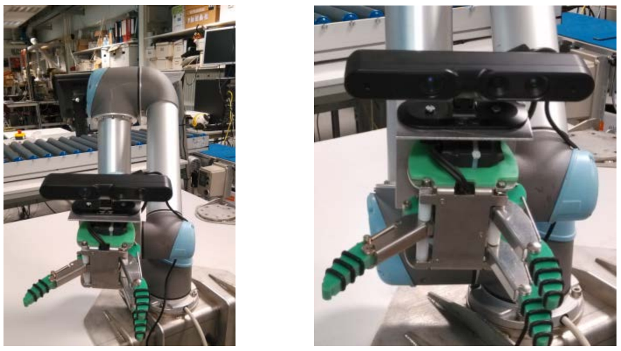


Figure 4.19: Experiment setup. A Lacquey under-actuated gripper and an Asus Pro Live sensor are installed at the end of the Universal arm (UR5).

From this experiment, authors can safely draw three conclusions. The first is that this grasping algorithm is very fast. Grasping computation for the tested objects can finish within one second. The second one is that this grasping algorithm is reliable. All the grasps found for these objects have good force balance and torque balance, as well as the gravity optimization. The third is that this grasping algorithm has a good tolerance. Point clouds of the electric drill, the spray bottle and the pan missed a lot of pixels because of the restriction of the Asus Xtion pro live sensor. However, the grasping algorithm can still work out good grasps for the target objects. The experiment also proved the effectiveness of our algorithm.

4.6 Comparison

As mentioned in the first part of motivation, inspiration of this chapter comes from [135]. Let’s look at the outcomes of [135] and our algorithm. Figure 4.21 shows the comparison between [135] and our algorithm. We made several improvements over [135]. The first one is [135] did not tell how they get the boundary of the object. We propose to use Oriented Bounding Box to obtain the boundary of the object, which is proved to be efficient in our experiments. The second is [135] did not consider the geometry property of the gripper.

Actually, the grasps found by [135] in the red circle are impossible to be executed by grippers like the PR2 gripper, because [135] did not consider the geometry property of the gripper. [135] just uses one single point to represent the finger. On the contrary, we consider the geometry shape of the finger from the beginning of our algorithm. The third is [135] did not consider gravity of the object, which plays an important role in object grasping. On the contrary, we choose the nearest grasp to the gravity center to work as the final grasp. This grasp can not only make sure the grasp can be executed successfully, but also ensure the grasp quality during the manipulation of the object after it is grasped. To sum up, our algorithm combines the advantage of caging grasping and force closure grasping, which is much more practical for flat object grasping than [135].

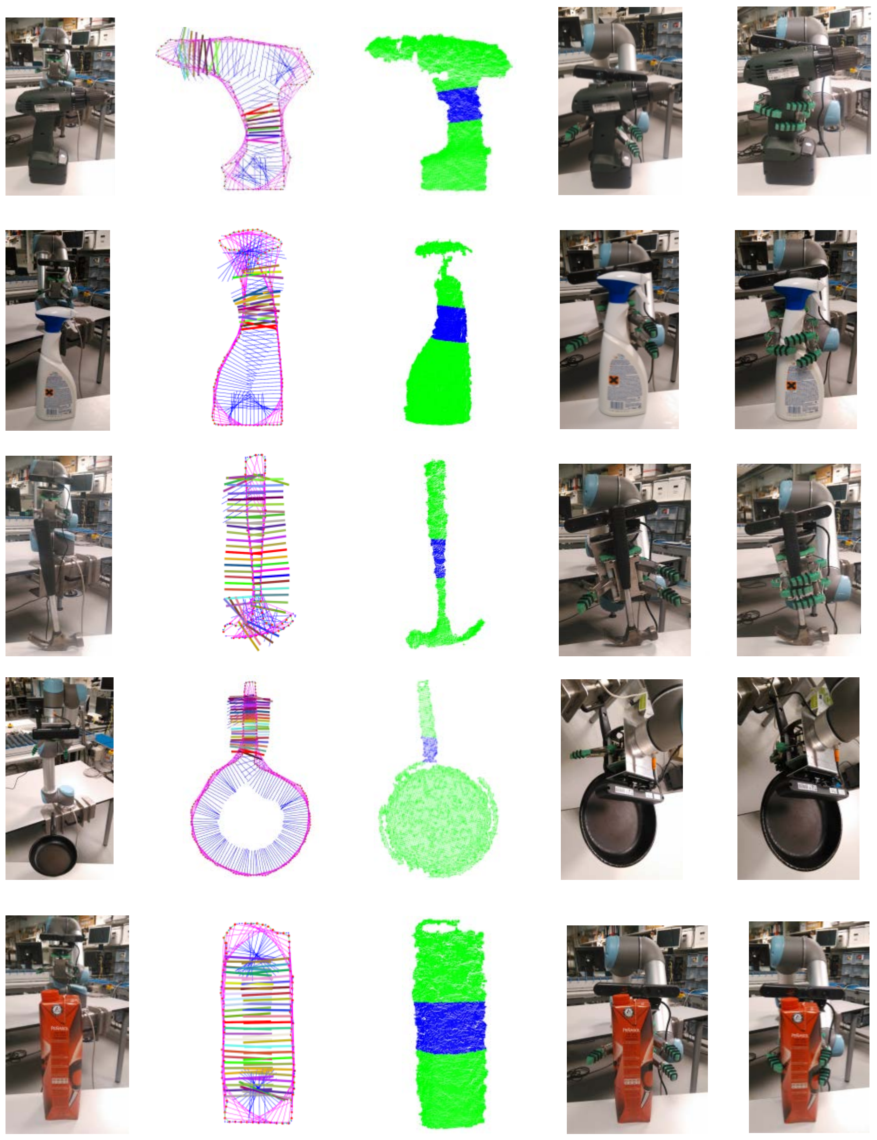


Figure 4.20: Snapshots from the experiments: Fist column is the initial state of the robot and the target objects. Second column is the result of grasping computation. Third column shows the grasp area on the point cloud of the target object. Fourth column shows the gripper arriving at grasping point. Fifth column shows objects grasped by gripper.

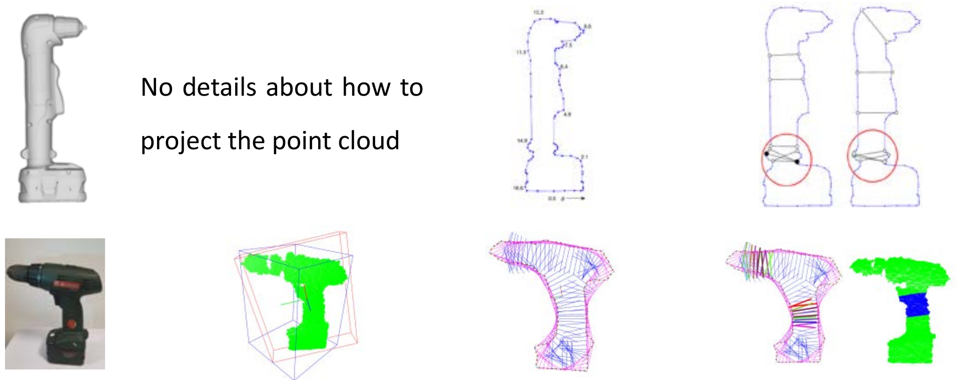


Figure 4.21: Comparison between [135] and our algorithm. The top is the algorithm form [135]. The bottom is our algorithm. Three improvements are made. The first is that [135] did not give details about how to project the point cloud. We use to OBB to find main plane to project point cloud. The second is that some grasps found by [135] are not practical for two-finger gripper because [135] did not consider the geometry property of the robot hand. We consider the geometry property of robot hand like the hand width, grasp range and the local geometry of every finger candidate on the boundary. The third is that [135] did not consider gravity of the object. Our algorithm considers gravity to make it more reliable.

4.7 Conclusion

In this chapter, a novel grasping algorithm is presented for flat object grasping by combining the merits of caging grasping and force balance grasping. The idea of caging points is borrowed to generating grasp candidates. After that, force balance computation is carried out to find out suitable grasps by considering the gripper geometry properties, for example, the grasping range and the hand width. Gravity of the target object is also considered to ensure the grasping quality during the manipulation of the object after it is grasped. This algorithm can quickly work out the best grasp with good force balance and torque balance. In order to prove the validity of our grasping algorithm, several objects with different geometry shapes are used to do simulations and experiments. And good results are obtained.

Acknowledgement

The work leading to these results in this chapter was supported by EU through the project: Factory in a day, which received funding from the European Community’s Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 609206.

5

Unknown object grasping by using concavity

This Chapter was published at the 2016 ICARCV conference:

Qujiang Lei, Martijn Wisse. 2016 IEEE International Conference on Control, Automation, Robotics and Vision (ICARCV), Phuket, Thailand.

Abstract

Reducing the grasp candidates for unknown object grasping while maintaining grasp stability is the goal of this chapter. In this chapter, we propose an efficient and straight forward unknown object grasping method by using concavities of the unknown objects to significantly reduce the grasp candidates. Shortest path concavity is first employed to work out the concavity value for every vertex of the unknown objects followed by concavity extraction to obtain the most salient concave areas. Grasp candidates are then generated on the most salient concave areas and evaluated by using force balance computation. Grasp candidates are ranked according to the result of force balance computation and the manipulability of every grasp candidate. The grasp with the best force balance and manipulability is chosen as the final grasp. In order to verify the effectiveness of our algorithm, some unknown objects commonly used by other papers about unknown object grasping are used to do simulations and favorable performance is obtained.

5.1 Introduction

Grasping of unknown objects with neither appearance data nor object models given in advance is very important for robots that work in an unfamiliar environment. Vast research has been conducted on the problem of unknown object grasping and many achievements have been obtained in the previous years. However, unknown object grasping is still a challenging task that has not yet been solved in a general manner.

[35] gives a profound survey about unknown object grasping. The existing unknown object grasping algorithms can be divided into two main categories, that is, using partial model and using full model.

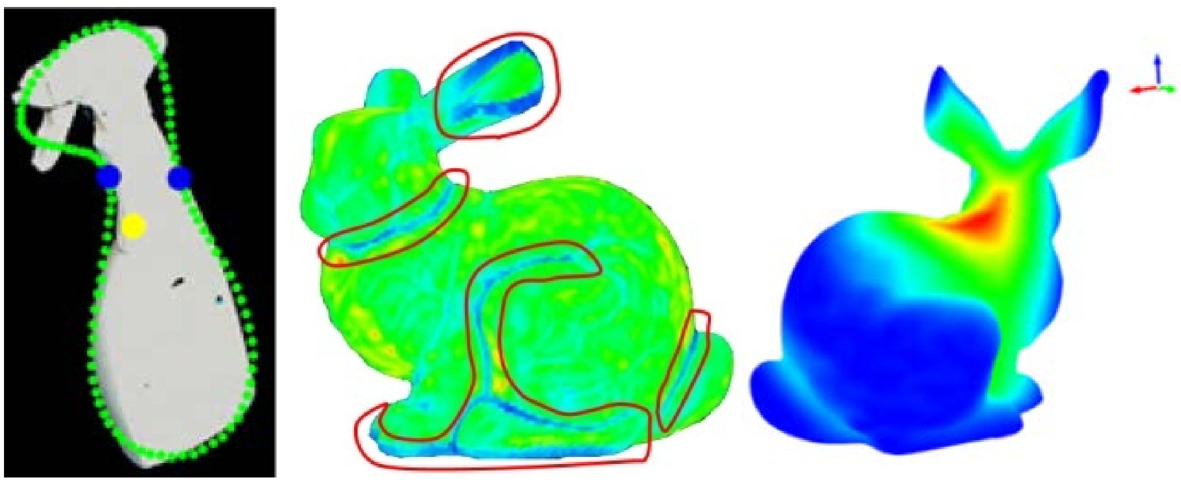
In order to accelerate the grasping process of unknown objects, partial information of an object may be used to realize the grasping. [55] uses partial object geometry to achieve a semantic grasp. This algorithm needs predefined example grasps and cannot deal with the grasping task of symmetric objects since multiple views of a symmetric object could have the same depth images. [80] proposes a data-driven grasp planner that requires partial sensor data. Matching and alignment methods were used for grasping after obtaining the Columbia Grasp Database. [82] uses local descriptors from several images to construct the 3D model of an object. Object registration was conducted by using a set of training images. [147] installs a 2D range sensor on the robot at an inclined angle to acquire partial shape information of the unknown objects. Two straight lines are extracted directly from this partial shape information as the two grasp sides for a parallel jaw gripper. [148] uses binocular vision to recover the partial 3D structure of unknown objects. Then process the incomplete 3D point clouds searching for good grasp candidate for a three finger robot hand according to a function that accounts for both the feasibility and the stability.

The second method is building a full 3D model using many images or point clouds of the target object. In [74], the full 3D model is fit and split into many minimum volume bounding boxes and a grasp is found on these bounding boxes. In [75], two flat, parallel surfaces are found on the 3D model to realize the grasping task with a gripper. In [76], the center of mass and axes of inertia of the target object are calculated from the 3D model, and then a grasp on the center or along the axes is found. [77] uses a genetic algorithm to search for grasping points on a 3D model of the target object. [78] uses a cost function to analyze the 3D model to obtain grasping points. In [79], the 3D model is simplified into some shape primitives (boxes or cylinders). Then grasping points which are assigned offline to these shape primitives are selected for the corresponding shape.

The best way to find a good grasp is said to use full 3D model to do grasp candidate simulation [108]. GraspIt which was first introduced in [108] established a benchmark for the object grasping community. However, there is a big problem we must face if we want to use GraspIt on unknown object grasping, that is how to deal with large number of grasp candidates promptly. After GraspIt, OpenGRASP [146] is invented on the base of the OpenRAVE [149], which made a progress comparing with GraspIt. OpenGRASP uses the normal of the object as the approaching vector of the robot hand, which can greatly reduce the number of grasp candidates. However, [116] states that depending on the choice of the parameters, the time of using OpenGRASP to simulate all the corresponding grasp candidates for a common object can vary from a few minutes to more than an hour. And then [116] uses [120] to do sampling to further reduce the grasp candidates. However, it still needs about one minute to find a good grasp for the unknown object, which is pretty time consuming. That is why the above 3D model based grasping algorithms try to use shape primitive or boxes to simplify the unknown objects.

The reason that using partial information for unknown object grasping is becoming popular is that it requires less data comparing with using full 3D model. In another word, using partial model can be quicker than using full 3D model. However using full 3D model can achieve a better stable grasp, especially when GraspIt or OpenGRASP is used even though it is time consuming. Can we combine the merits of these two methods together? What if we have a 3D model first and then we just use part of the 3D model to synthesize a stable grasp.

In our previous works [123, 124], we employed the principal axis of the unknown object to accelerate the grasp searching process and good results are obtained. In this chapter, inspired by using curvature on unknown object grasping in [86] and using downsampling to reduce the number of grasp candidates in [116], we propose to use concavity of the unknown object to reduce grasp candidates to accelerate the generation of grasp candidates for the unknown objects. The concavity we said is different from the 2D curvature used in [86]. There is a significant difference between 2D curvature and 3D curvature regarding geometry properties. As can be seen from Figure 5.1 (a), the two blue points mean the maximum curvatures of the green silhouette of the spray bottle. Apparently, 2D curvature has a good performance to help searching a good grasp for robots. However, the situation for 3D curvature is different. As can be seen from Figure 5.1 (b), the blue in the red circles stands for the Gaussian curvature (3D curvature), which actually cannot give much clue for robot grasping. Fortunately, [150] and [151] give us inspiration about using concavity on unknown object grasping. Figure 5.1 (c) shows an example of the concavity that we mentioned. Specifically, the red area of the bunny represents the most concave area. Obviously, the concavities of the unknown object have a higher possibility to form a more stable force closure grasp. Our motivation is to generate grasp candidates on concavities and choose the best grasp by evaluating every grasp candidate.



(a) (b) (c)

Figure 5.1: Curvature versus concavity: (a) 2D curvature (the two blue points stands for the maximum curvature of the silhouette of the spray bottle), (b) 3D curvature (the blue in the red circles mean the Gaussian curvature of the bunny), (c) an example of concavity (the red represents the most salient concavity).

The overview of our grasping algorithm is as follows. First, the 3D model (point cloud or meshed model) is input and the concavity of the unknown object is worked out as Figure 5.2 shows. Then the most salient concavities are extracted according to the concavity intensity. After that, grasp candidates are generated followed by the evaluation of every grasp candidate. Finally, the grasp with best force balance and manipulability will be chosen as the final grasp to execute.

This chapter is organized as follows. Section 5.2 contains a detailed explanation of our algorithm, Sections 5.3 shows the simulation results, and Section 5.4 is the conclusion of this chapter.

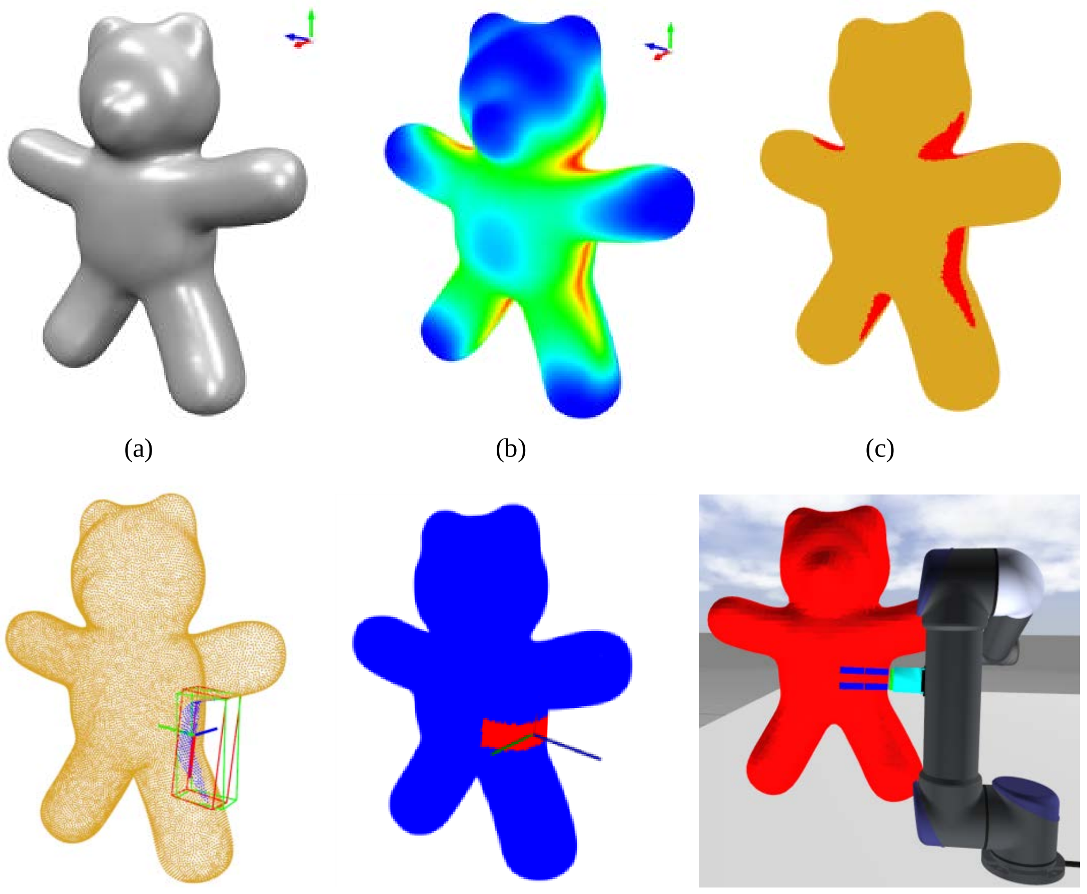


Figure 5.2: An over view of our algorithm: (a) the input model, (b) concavity computation, (c) the most salient concavities are extracted (the red stands for the most salient concavities), (d) concavity analysis, (e) the returned best grasp, (f) the execution of the best grasp.

5.2 Detailed algorithm

5.2.1 Concavity calculation

Concavity calculation is carried out directly on the meshed model of the target object. The meshed model can be obtained by fast triangulating the full point cloud of the target object. Take the bear as example, Figure 5.3 (a) is its full point cloud and Figure 5.3 (b) is its meshed model. Then a proper outer convex hull is computed (shown as the Figure 5.3 (c)). The free space between the meshed model and the proper outer convex hull is used to compute the concavity. First, the stable Constrained Delaunay Tetrahedralization is used to discretize the free space. The corresponding discretized space is shown as Figure 5.3 (d). Then Fast Marching Method is employed to compute the shortest path distance by using the discretized free space. The shortest path distance between the meshed model and the convex hull is returned as the concavity value of every vertex of the meshed model. Figure 5.3 (e) shows the concavity computation result, as can be seen, the original concavity value is in disorder. In order to have a better view of the concavity computation result, the concavity value is rendered by color according to its intensity from the maximum concavity value to the minimum concavity value (the red means the maximum concavity and the blue means the minimum concavity). Figure 5.3 (f) shows the result of the rendered concavity.

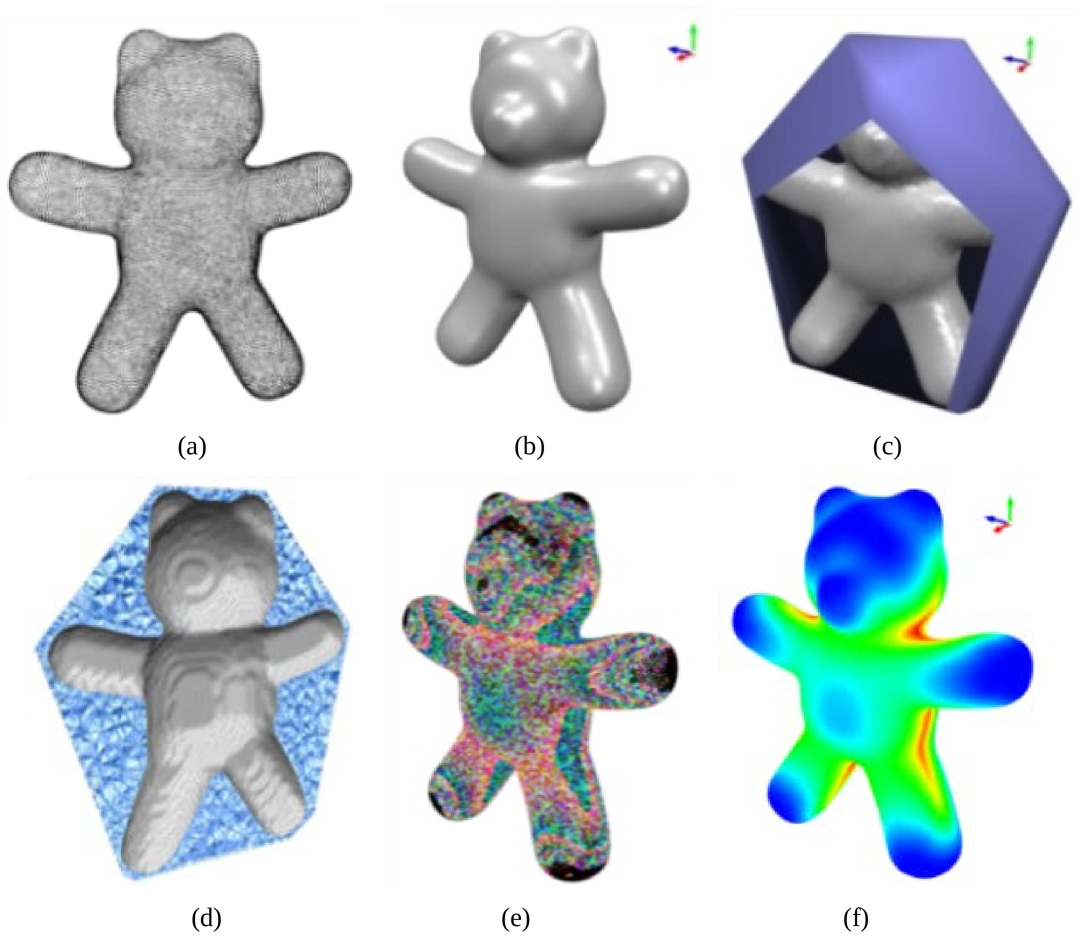


Figure 5.3: The process to compute the concavity for a bear, (a) The point cloud of the bear, (b) the meshed model of the bear, (c) the convex hull of the bear, (d) discretization of the free space between the bear and the convex hull, (e) the original concavity computation result for every point in (a), (f) the result of the rendered concavity.

5.2.2 Concavity extraction

As can be seen from Figure 5.3 (f), the red stands for the most concave areas and the blue represents the least concave areas. In order to make it convenient to do further analysis on the point cloud with the concavity value of every point, the most concave areas are extracted. The red in Figure 5.4 (a) means the most concave areas. It is extracted out from the whole point cloud to be shown as the Figure 5.4 (b). After that, the Euclidean cluster extraction is employed to separate the concavity with each other (shown as the Figure 5.4 (c)). Further grasping analysis will carried out on the separated concavity point clouds. Specifically, the separated point clouds of the concavity will be used to generate grasp candidates.

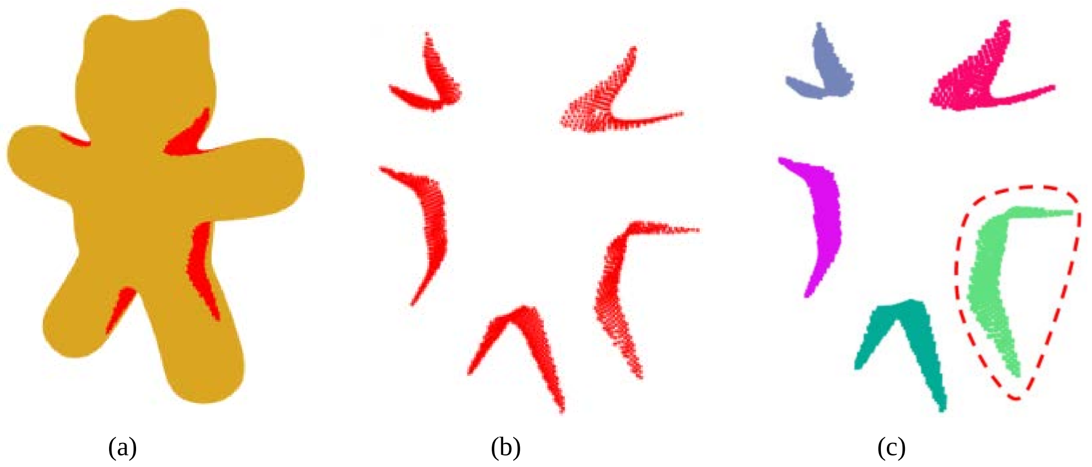


Figure 5.4: The extraction and separation of the bear concavity. (a) and (b) demonstrate the extraction of the most salient concavities (the red areas), (c) the most salient concavities are separated into every single concavity.

5.2.3 Construct the concavity coordinate system

The concavity in the red dashed circle in Figure 5.4 (c) is used to explain our following algorithm. After we get the point cloud of the every single concavity, we can use the principle of OpenGRASP. The grasp candidates can be generated by using the normal of the point cloud. But one important fact we must notice is that there are still a lot of grasp candidates even though we downsampled the normal of the point cloud. The reason is that the normal can only decide the approaching direction of the robot hand. The robot hand can rotate around the normal, which means a lot of grasp candidates can be generated. In order to effectively decrease the number of grasp candidates. We propose to use oriented bounding box to reduce the grasp searching, which will be explained later.

Let’s first look at what the oriented bounding box is and how to get the oriented bounding box. There are two common ways to obtain a bounding box, that is the axis-aligned bounding box (AABB) and the oriented bounding box (OBB). The axis-aligned bounding box for a given point set is its bounding box subject to the constraint that the edges of the box are parallel to the Cartesian coordinate axes. It is simply the Cartesian product of N intervals each of which is defined by the minimal and maximal value of the corresponding coordinate for the points. The oriented bounding box is the bounding box calculated subject to no constraints as to the orientation of the result. By using the eccentricity and moment of inertia, a position vector and a rotation transform matrix can be obtained. And then, each vertex of the given AABB must be rotated with the given rotation transform matrix and then positioned to get the OBB. Therefore, the OBB is much more generous and better suitable for concavities analysis. The green and red rectangular parallelepiped frames in Figure 5.5 respectively demonstrate the axis-aligned bounding box and the oriented bounding box. The blue, green and red straight lines in Figure 5.5 (b) respectively stand for the X, Y and Z axis of the concavity coordinate system.

After the oriented bounding box is obtained, the algorithm will analyze which two parallel planes of the rectangular parallelepiped frames have the higher possibility to be grasped by robot hand without collision with the object. For the concavity shown in Figure 5.5 (a), an oriented bounding box is obtained as Figure 5.5 (b) shows. The oriented bounding box can be divided into three pairs of parallel planes. The pair of the front and the back planes has the less possibility to collide with the bear. Therefore, the following grasp analysis will be carried out in XOY plan of the concavity coordinate system.

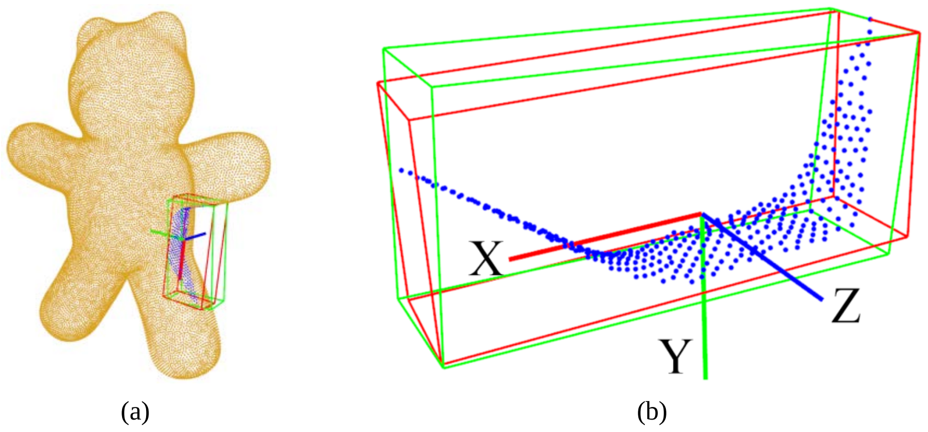


Figure 5.5: The oriented bounding box and its corresponding Cartesian coordinate system. The blue and the red rectangular parallelepiped frames respectively stand for the axis-aligned bounding box and the oriented bounding box.

5.2.4 Analyze concavity and generate grasp candidates

In the above subsection, the oriented bounding box and its corresponding Cartesian coordinate system have been obtained. Actually, the three axis of the Cartesian coordinate system respectively sand for the major eigenvector, the middle eigenvector and the minor eigenvector. According to the analysis in subsection 5.2.3, the major and middle vector will be used to generate the grasp candidates. First, the point cloud of the concavity is projected to the XOY plane of the OBB coordinate system (shown as the Figure 5.6).

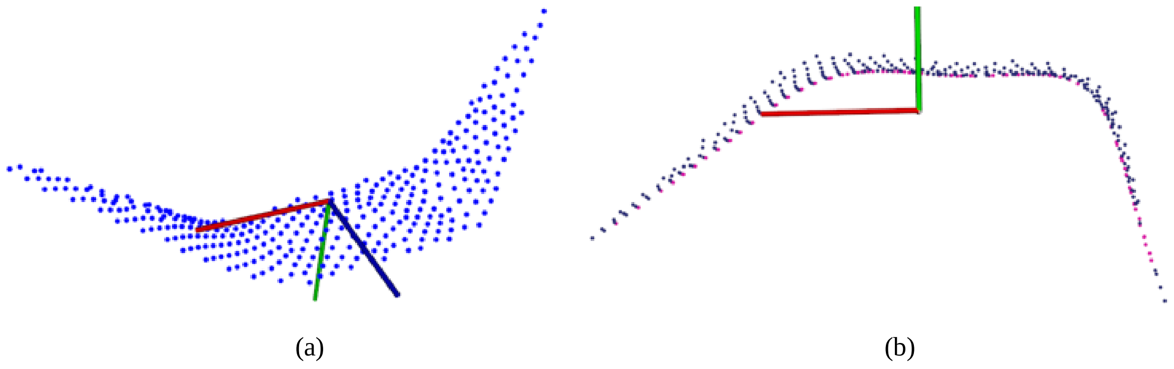


Figure 5.6: The point cloud of concavity in the OBB coordinate system and the projected point cloud, (a) the point cloud of the concavity in the OBB coordinate system, (b) the projected point cloud of the concavity.

After that, we need to extract the concavity outer layer on the projected point cloud. The outer layer boundary (shown as the purple points in Figure 5.6 (b)) of the projected point cloud will be used to configure the grasp candidates. We first obtain the concave hull of the projected point cloud (shown as the Figure 5.7). The concave hull can be divided into two parts, one part is inside the object, and the other part is on the boundary. Usually, the point density is employed to judge whether a two dimensional point is on the boundary or not, therefore, an enlarged point cloud is extracted by adding more less concave areas (which are shown as the red points in the Figure 5.8 (a)). Then the enlarged point cloud is down-sampled shown as the red points in the Figure 5.8 (b).

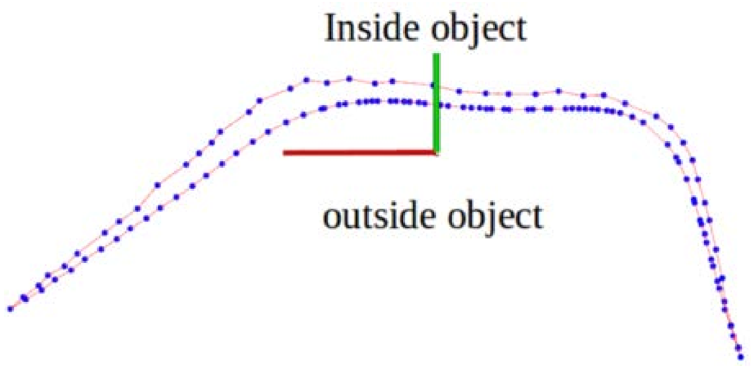


Figure 5.7: Concave hull of the projected point cloud of the concavity.

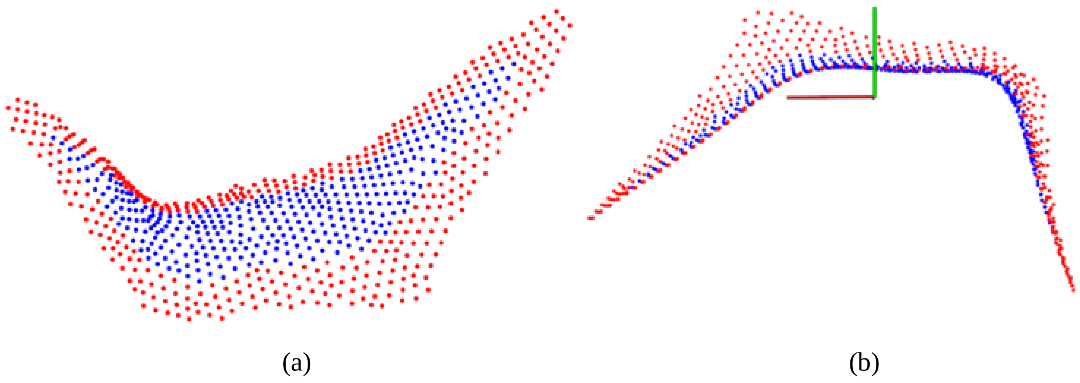


Figure 5.8: An enlarged point cloud and its projected point cloud, (a) the blue points mean the concavity point cloud shown in Figure 5.6 (a), the red points mean the enlarged point cloud by add more less concavity areas, (b) the enlarged point cloud is projected and down-sampled shown as the red points.

Originally, we tried to compare the point density of every point (the blue points in Figure 5.9) on the concave hull of the concavity point cloud. By introducing a circle with the radius of R (the green and the black circle in Figure 5.9), we can get the number of the points within the circle. Apparently, the point on the boundary will have a low point density. However, all the blue points are pretty close to the boundary and the down-sampled enlarged point cloud is sparse, comparing the density of every point on the concave hull of the projected concavity point cloud is not robust, especially for the blue points in the green rectangular.

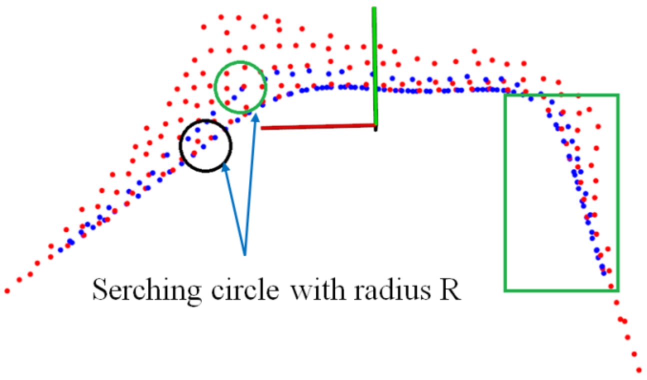


Figure 5.9: Extract the boundary points by compare the point density of every blue point.

Considering the instability of comparing point density, we come up with a method to extract boundary point more stably. Specifically, two concave hulls of the concavity point cloud and the enlarged point cloud are used. Through observing the two concave hulls visualized in Figure 5.10 (a), it is pretty obvious that the boundary points on the concave hull of the concavity point cloud are almost the same or pretty near to the points on the concave hull of the enlarged point cloud. Therefore, the distance between every blue point and the green line (shown as the Figure 5.10 (b)) is used to judge whether a point is on the boundary or not. One important fact we can use is that points on concave hull is generated in sequential order, as can be seen, the blue point on the upper red line of Figure 5.10 (b) is in order as n-1, n and n+1. A vector can be used to store all the straight lines between the two adjacent red points of the concave hull of the enlarged point cloud. For every blue point, a vertical straight line to the straight lines constructed by the two adjacent red points can be obtained. If the intersection point between the vertical line and the straight line is lying between the two red points, then that intersection point is recorded. The orange point in Figure 5.10 (b) is an example of the intersection point and it satisfies the Equation (5.1). A distance threshold ($d_{threshold}$) is given by the system to decide whether the blue point (point n) is on the boundary or not, if the distance $d < d_{threshold}$, the point is considered as the boundary point. Using above method to go through all blue points can get the boundary point cloud shown as the orange points in Figure 5.11.

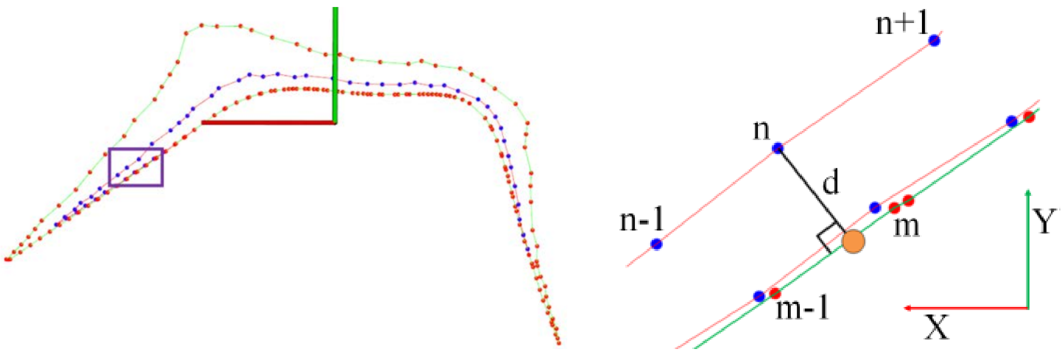


Figure 5.10: (a) is the two concave hulls of the projected concavity point cloud and the enlarged point cloud, (b) is the enlarged image of the points in the purple rectangular in (a).

$$\begin{cases} x_m < x_{point\_orange} < x_{m-1} \\ y_{m-1} < y_{point\_orange} < y_m \end{cases} \tag{5.1}$$

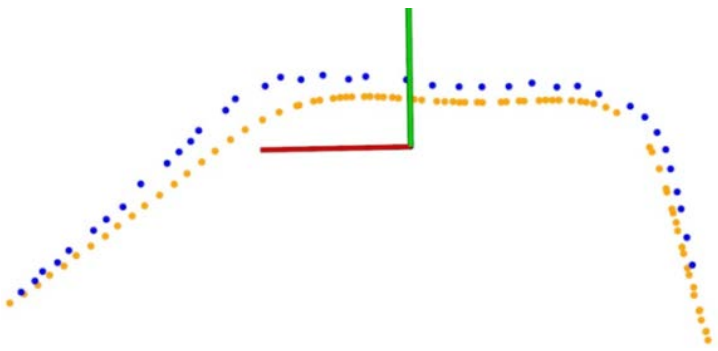


Figure 5.11: Orange points mean the boundary points of the concavity point cloud.

5.2.5 Generate grasp candidates

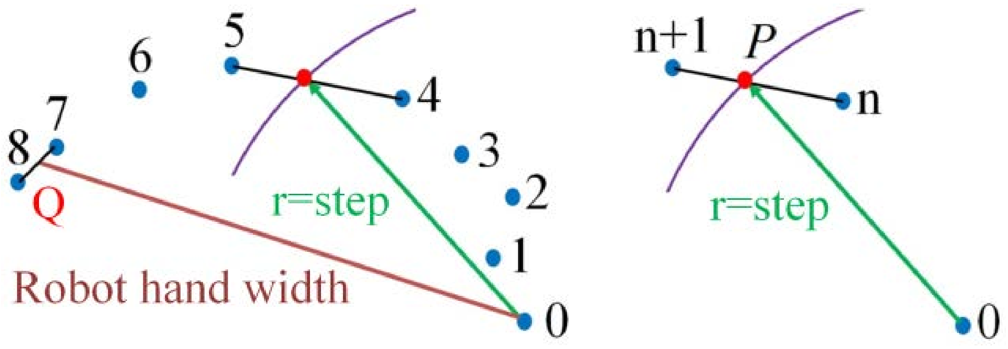
After we get the boundary point cloud, the points are not in order. The first thing to do is to make all the boundary points into order. We first find the point with $x_{\min}$ and store it into a vector ($V_{boundary\_in\_order}$). Then sequentially find the closest point and add it into the vector. Figure 5.12 shows an example of the boundary points, which are in sequential order as 0, 1, 2, 3, 4, 5, until 8.

After the boundary point cloud in order is obtained, the next to do is to calculate a series step points. Step points are used to configure the grasp candidates. It means the robot will search along the boundary with a step. Search process of the step points will start from the point 0 ($V_{boundary\_in\_order}$ [0]). And then search along the boundary to find any two adjacent points, which can construct a straight line and have intersection point with a circle with a radius equal the searching step. If the step is given as r and the two adjacent points are point n and point $n+1$, the distance between the point 0 and point n is defined as d_{0_n} and the distance between the point 0 and point $n+1$ is defined as d_{0_n+1}.

If the distance between the start point and the two adjacent points satisfies any of the Equation (5.2), there must be an intersection point between the boundary and the step circle. Figure 5.12 shows an example of searching the step points. The purple curve stands for the searching circle with the radius equaling the searching step. The purple curve has an intersection point with the straight line between point 4 and point 5. Giving the coordinate value of point 0, point n, point $n+1$ and the step length (r), using Equation (5.3) can get the coordinate value of the intersection point P (shown as the red point in Figure 5.12 (b)). After we get the first step point, all the points from the point 0 to point n will be removed from the current boundary point cloud to form a new point cloud. The new point cloud is shown as the Figure 5.13 (b). Then, the $d_{\max}$ is checked to see whether it is bigger than the robot hand width. If yes, then the algorithm will repeat the way of finding the first step point to find the second step point. The above steps are repeated until $d_{\max}$ is smaller than the robot hand width (shown as the Figure 5.13 (a)).

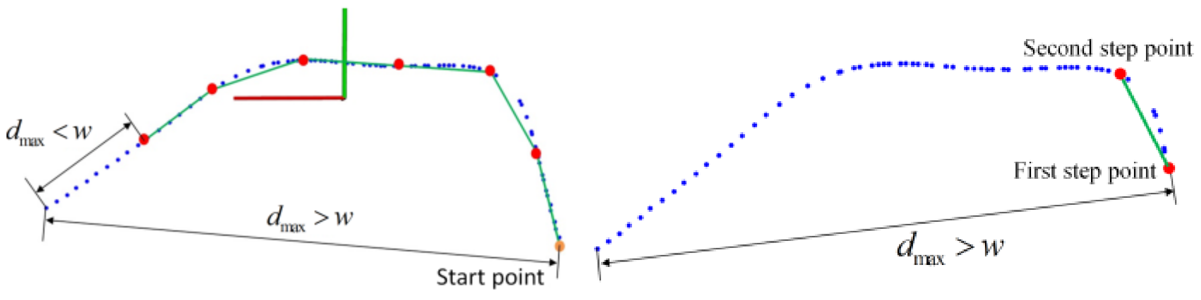
$$\begin{cases} d_{0\_n} < r < d_{0\_n+1} \\ d_{0\_n+1} < r < d_{0\_n} \end{cases} \tag{5.2}$$

$$\begin{cases} m = \dfrac{p_n.x - p_{n+1}.x}{p_n.y - p_{n+1}.y} \\ w = p_{n+1}.x - mp_{n+1}.y \\ p.y = \dfrac{2p_0.y + 2mw \pm \sqrt{(2p_0.y + 2mw)^2 - 4(m^2+1)(p_0.y^2 + w^2 - r^2)}}{2(m^2+1)} \\ p.x = my + p_{n+1}.x - mp_{n+1}.y \end{cases} \tag{5.3}$$



(a) (b)

Figure 5.12: The step point searching.



(a) (b)

Figure 5.13: The way to find all the step points, if a step point is found, the points out of the searching circle will form a new point cloud as (b).

Figure 5.14 shows the step searching result with different step length. The blue points stands for the boundary point cloud of the concavity. The red points represent all the step points. Figure 5.14 (a) and (b) respectively show the result of step searching with a small and a big step length.

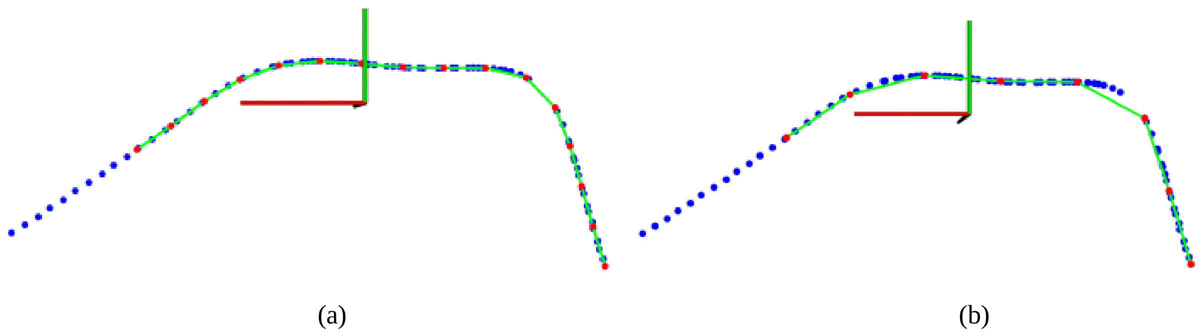


Figure 5.14: The step points obtained by using different steps length. The blue points are the boundary points of the concavity. Every green line stands for a step. Every red point represents a step point. (a) shows the result of step searching with a small step. (b) demonstrates the result of step searching with a big step.

After all the step points are obtained, the way to work out the step point can be used to work out the hand configuration. In Figure 5.12, if point 0 is a step point and the length of the brown line equals the robot hand width, using method shown in Figure 5.12 (b) can work out the intersection point Q. A grasp candidate can be configured between point 0 and point Q (in Figure 5.12 (a)). The purple lines in Figure 5.15 stand for all the grasp candidates. Specifically, Figure 5.15 (a) and (b) respectively show all the grasp candidates corresponding to Figure 5.14 (a) and (b).

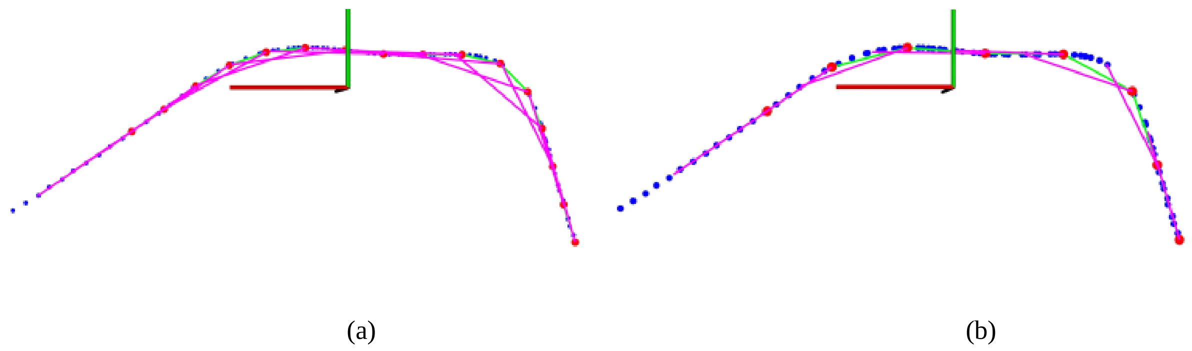


Figure 5.15: The grasp candidates obtained by using a small step and a big step. The purple lines stand for the grasp candidates. Blue points, green lines and red points are the same as Figure 5.14.

After all the grasp candidates are obtained, middle vertical lines to every grasp candidate are work out shown as blue lines in the Figure 5.16. On every grasp candidate, a local Cartesian coordinate system is established by using purple lines as the X axis and the blue lines as the Z axis. Using cross product of the Z axis and the X axis can get the Y axis. Then point cloud for every grasp candidate is extracted and transformed to the local coordinate system. Figure 5.17 shows 3 example grasp candidates in their own local coordinate system.

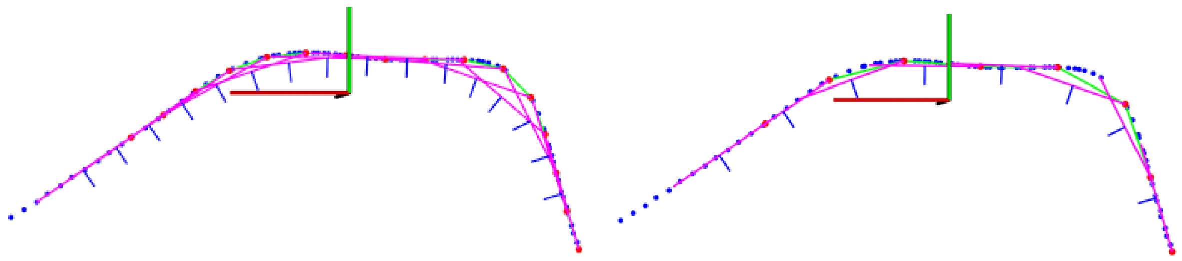


Figure 5.16: The construction of local coordinate system, the purple lines represent the grasp candidates and work as the X axis of the local coordinate system. The middle vertical lines (blue lines) work as the Z axis.

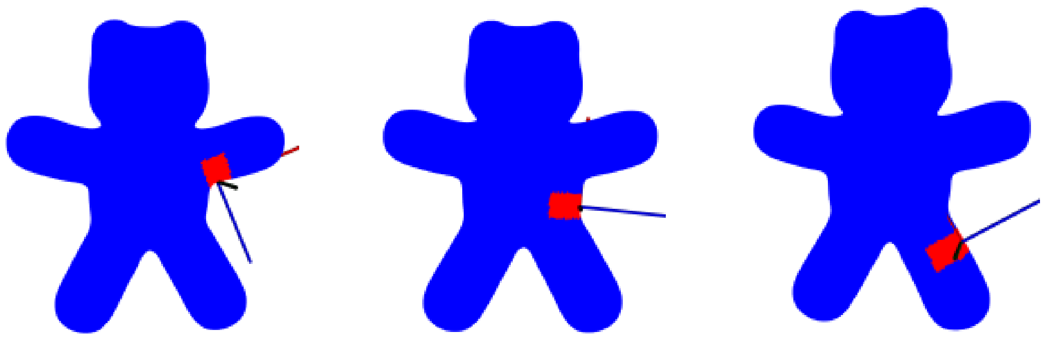


Figure 5.17: Three grasp candidates and their coordinate system.

5.2.6 Force balance computation and manipulability analysis

Force balance analysis is carried out on every grasp candidate to evaluate the stability of the grasp candidate. Figure 5.18 is one grasp candidate from the 16 grasp candidates shown in Figure 5.15 (a). This grasp candidate will be used to explain the way of doing force balance analysis.

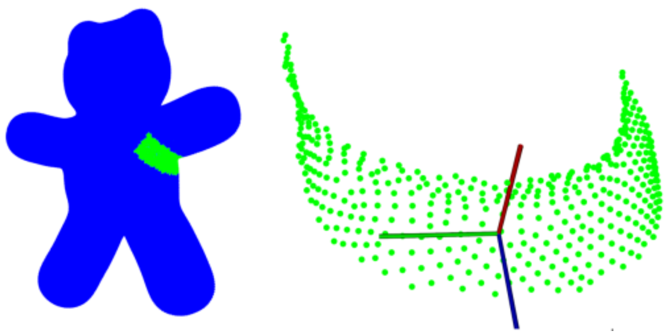


Figure 5.18: One example grasp candidate.

At first, the point cloud of the grasp candidate is projected to the XOY plane (made of the red and green lines in Figure 5.18). The projected point cloud can be seen as the green points in Figure 5.19 (a). Then the concave hull (the blue points in Figure 5.19 (a)) of the projected point cloud is abstracted. The two grasp sides are extracted shown as the red points and the green points in the Figure 5.19 (b). After we get the points of the two grasp sides, a straight line fitting method is employed to do line fitting for the two grasp sides. If a straight line is defined as $y = kx + b$, Equation (5.4) can be used to work out k and b. The purple line on the left in Figure 5.19 (b) stands for the fitting line for the red points. Correspondingly, the red line on the right side of Figure 5.19 (b) represents the fitting line for the green points. The angle (β) between the purple line and the red line is used to evaluate the grasping stability of this grasp candidate. Figure 5.20 shows the result of force balance computation of the 16 grasp candidates.

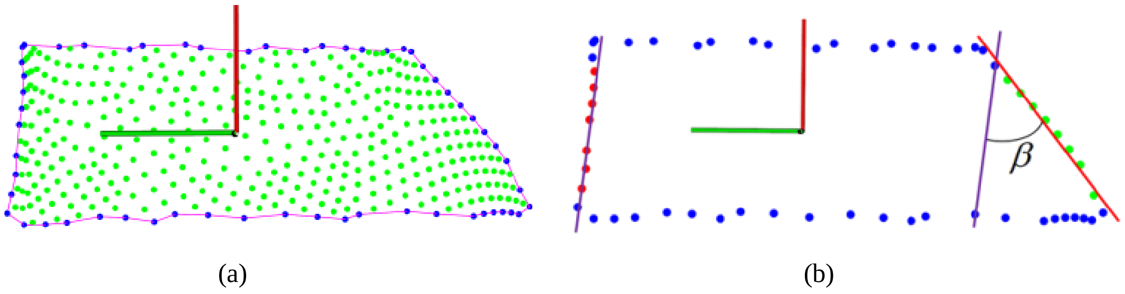


Figure 5.19: (a) shows the projected point cloud (green points) of the grasp candidate, the blue points mean the concave hull of the projected point cloud. (b) is force balance analysis of the two grasp sides by using line fitting.

$$\begin{cases} k = \dfrac{\sum_{i=1}^{n} x_i y_i - \dfrac{1}{n}\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{\sum_{i=1}^{n} x_i^2 - \dfrac{1}{n}\sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i} \\ b = \dfrac{1}{n}\sum_{i=1}^{n} y_i - k\dfrac{1}{n}\sum_{i=1}^{n} x_i \end{cases} \tag{5.4}$$

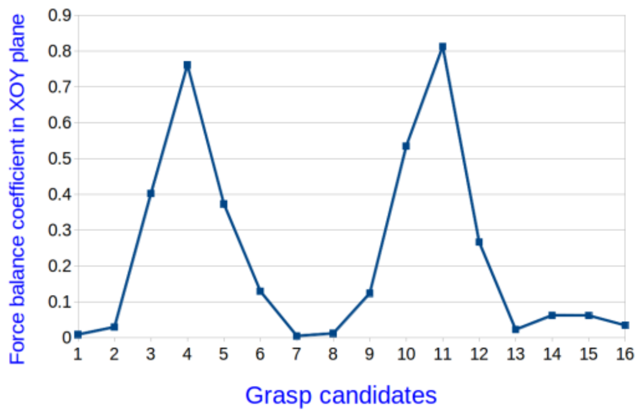


Figure 5.20: The result of force balance computation of the 16 grasp candidates shown in Figure 5.15 (a).

As can be seen from Figure 5.20, both grasp 1 and 7 have good force balance performance. How to select the best one for this concavity? Because all grasp candidates are generated on the concavity of the target object, the robot may collide with the target object when the robot tries to execute the grasp action. Therefore, it is necessary to analyze the manipulability of every grasp candidate that has good force balance. Figure 5.21 shows the grasp 1 and grasp 7. When the robot executes these two grasps, the gripper will approach the object along the red arrow. The length and the width of the gripper are considered to do collision check. As visualized in Figure 5.21, if a grasp leads to no collision between the gripper and the target object, this grasp is saved; otherwise, this grasp is abandoned. Among all grasp candidates that passed collision check, the grasp candidate with the best force balance is chosen as the best grasp for this concavity. Here, grasp 7 is chosen as the best grasp for this concavity because there is no collison between grasp 7 and the bear. After finishing the grasping analysis of the example concavity shown in dashed circle in Figure 5.4 (c), the algorithm repeats the above concavity analysis on other four concavities shown in Figure 5.4 (c) to acquire the best grasp for every concavity. Then, the best grasps from every concavity are further compared with each other in the aspects of force balance, the best one is chosen as the final grasp execution.

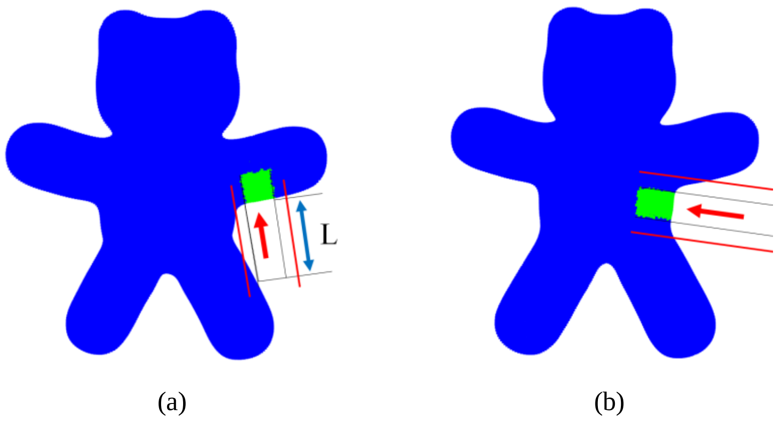


Figure 5.21: The comparison of two grasps with good force balance.

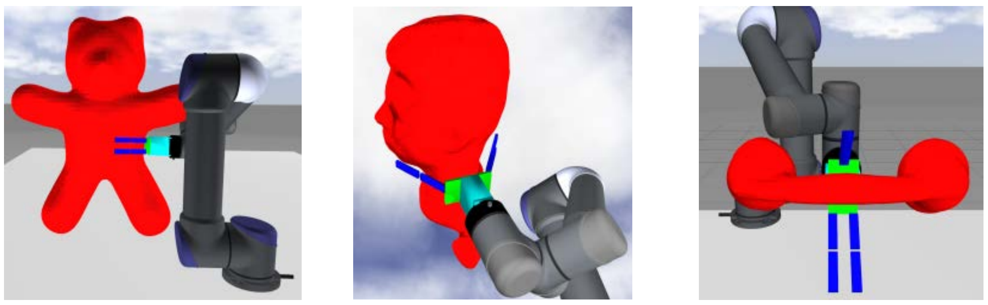


Figure 5.23: Execution of the best grasp returned from the algorithm.

5.3 Simulation

In order to verify our grasping algorithm, several objects in different geometry shapes are chosen to do simulations. All the tested objects can be seen in the first column in Figure 5.22. The second and third column shows the results of concavity computation and concavity extraction. The fourth column is the results of force balance computation of grasp candidates on one concavity of the object. The best grasps for the objects of the first column is shown in the fifth column. Figure 5.23 shows the execution of the best grasp returned from the algorithm. The returned best grasp for each object has good force balance and manipulability. The concavity computation for these objects can be finished within ten seconds. Concavity analysis, grasp candidate generation and force balance analysis can be completed within 2 seconds. Therefore, the whole grasping computation from computing the concavity to obtaining the best grasp is within 12 seconds, which is much faster than [116]. [116] uses OpenGRASP and down-sampling of the grasp candidates, but the time for an common object still needs about one minute. In summary, the simulations demonstrated our improvement over other grasping algorithms that also use full 3D model.

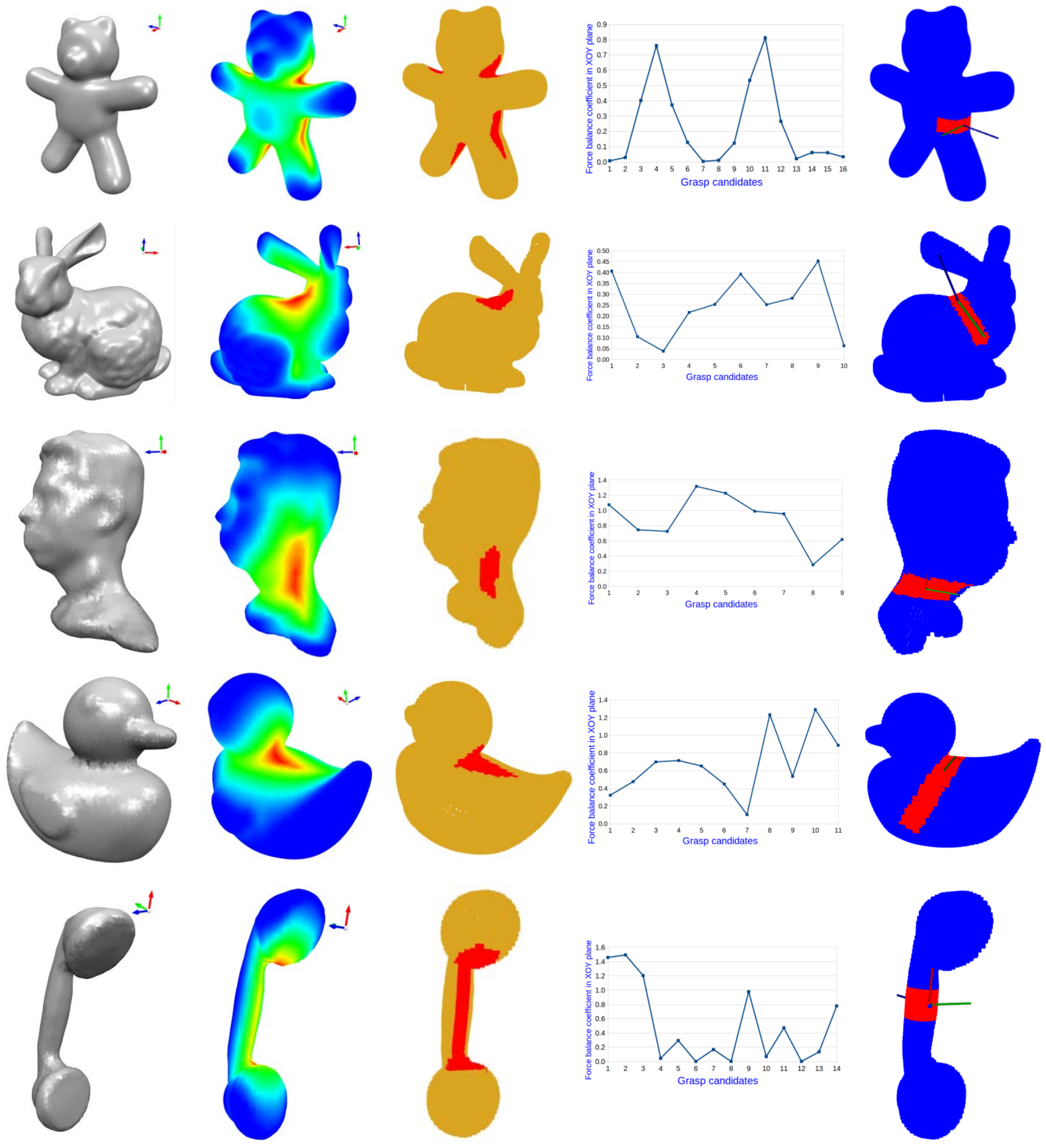


Figure 5.22: Simulation results

5.4 Conclusion

In this chapter, a novel grasping algorithm for unknown objects is presented. Concavity is first introduced to unknown object grasping in this chapter. Oriented bounding box is used to construct the coordinate system for every concavity followed by grasp candidate generation on every concavity. Force balance analysis and manipulability analysis are employed to evaluate every grasp candidate and the grasp with best force balance and manipulability is returned as the final grasp. In order to verify the effectiveness of our algorithm, several objects commonly used by other grasping algorithms with different geometric shapes are used to do simulations and successful results are obtained. Our algorithm can quickly finish the whole grasping computation from calculating concavity to obtaining the best grasp within 12 seconds, which is much faster than [116]. [116] uses OpenGRASP and downsampling of the grasp candidates, however, the time for a common object grasping needs about one minute. In summary, our algorithm shows improvement over other grasping algorithms which also use full 3D model. Our algorithm has a much better performance in time efficiency and grasping stability.

Acknowledgement

The work leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 609206.

6

Fast grasping of unknown objects using C-shape configuration

This Chapter is a journal paper accepted by Journal of AIP Advances, which is based on two earlier conference papers:

1. Qujiang Lei, Martijn Wisse, “Fast C-shape grasping for unknown objects”, 2017 IEEE International Conference on Advanced Intelligent Mechatronic (AIM 2017), Munich, Germany.

2. Qujiang Lei, Martijn Wisse, “Fast grasping of unknown objects using cylinder searching on a single point cloud”, 2016 9th International Conference on Machine Vision (ICMV 2016). Nice, France.

Abstract

Increasing the grasping efficiency is very important for robots to grasp unknown objects especially subjected to unfamiliar environments. To increase grasping efficiency, a new algorithm for fast grasping of unknown objects is proposed based on the usage of C-shape configuration. Specifically, the geometric model of the used under-actuated gripper is approximated as a C-shape. To obtain an appropriate graspable position, this C-shape configuration is applied to fit geometric model of an unknown object. The geometric model of an unknown object is constructed by using a single-view partial point cloud. To test the algorithm using simulations, a comparison of the commonly used motion planners is made. The motion planner with the highest number of solved runs, lowest computing time and the shortest path length is chosen to execute grasps found by the grasping algorithm. The simulation results demonstrate that excellent grasping efficiency is achieved by adopting our algorithm. To validate this algorithm, experiment tests are carried out using a UR5 robot arm and an under-actuated gripper. The experimental results show that steady grasping actions are obtained. Hence, this research provides a novel algorithm for fast grasping of unknown objects.

6.1 Introduction

An unknown object can be defined as an item that has neither apparent information nor geometric model. Fast grasping of unknown objects is quite important for robots efficiently perform missions especially under unfamiliar environments. Due to the fact that various robots are increasingly dependent in contemporary society, improving grasping speed emerges as one essential challenge to achieve fast grasping of unknown objects.

A literature study reports five dominant fast grasping algorithms [84, 94, 100, 113, 114]. Among them [110] is a well acknowledged fast grasping algorithm using Hough transformation to visualize the edges of objects in a 2D image. It can detect whether the edges are sufficiently long and whether the parallel edges suit the width of the used grippers. In the work of Eppner and Brock [94], the point cloud is transformed into shape primitives (cylinder, disk, sphere and box). A pre-grasp (configuration of the hand) is chosen according to their shape primitives. This type of shape primitive can significantly reduce the scope of grasp searching to achieve a fast grasping algorithm. However, this may result in lots of grasp uncertainty, which may lead to grasp failure.

[84] applies the contact area of the grasping rectangle to determine the suitable grasps. When the contact area is too small, the grasp is likely to fail, and thus has to be replaced by another one. [114] utilizes principal axis and centroid of the object to synthesize a grasping action.

Pas [113] tries to fit the shape of the parallel gripper on the point cloud of the objects. They use a detailed segmentation to be able to pick objects from dense clutters. This algorithm promotes quite efficient grasping action, however, the parallel gripper is not advocated on the respect of flexibility comparing with dexterous hands and under-actuated grippers. These three fast grasping algorithms have a common character of using the normal of the table plane as the grasp approaching direction, which can accelerate grasp searching. However, due to the limitation that grasping from top is inapplicable for many objects that are placed in enclosed spaces, e.g., fridges and shelves, this type of simplification cannot be widely accepted.

In summary, it shows that except [94], the other four fast grasping algorithms [84, 110, 113, 114] are designed for parallel grippers. In addition, excluding [110] that uses RGB images as input of the grasping algorithm, the rest four grasping algorithms employ a partial point cloud as input. To design a faster grasping algorithm than the above five fast grasping algorithms, it is necessary to create a more general and faster grasping algorithm for simple grippers by using a partial point cloud.

In fact, four of the five dominant fast grasping algorithms choose to use parallel grippers, because parallel grippers have simpler geometry shape and are easier to control. In addition, the parallel grippers are cheap such that the grasping algorithms specially designed for parallel grippers can be widely used. Nevertheless, all of them ignore a kind of excellent robot hands, that is, under-actuated grippers.

The under-actuated gripper is one of the three kinds of popular robot hands. Other two kinds of robot hands are, dexterous hands and parallel grippers. Even though dexterous hands are very good at flexibility, the high complexity and high price stop them from being widely used in the research field of fast grasping of unknown objects. However, between the dexterous hands and the parallel grippers, there is a kind of grippers with high flexibility, low complexity and low price, which is under-actuated gripper. Under-actuated grippers are a very good tradeoff between dexterous hands and parallel grippers. Figure 6.1(a-c) shows three popular cheap under-actuated grippers. In order to achieve a cheap and general grasping algorithm, we will adopt the under-actuated grippers shown as Figure 6.1. All the three types of under-actuated grippers can be described as a C-shape with radii and as particularly shown in Figure 6.1(d).

Typically not a lot of grasping algorithms give details about the actual motion planning of the robotic arm towards the object. Grasping algorithms seem to only focus on finding grasps on the object itself. Researchers and users that want to implement grasping algorithms have to fill the gap of motion planning. They have to study on many different available motion planning methods before implementing it, which is time consuming. In order to help future researchers and users quickly choose a suitable motion planner to execute grasp action, we will make a comparison of different online motion planners available in Moveit!. The motion planner with the highest number of solved runs, lowest computing time and the shortest path length will be chosen to execute grasps found our grasping algorithm.

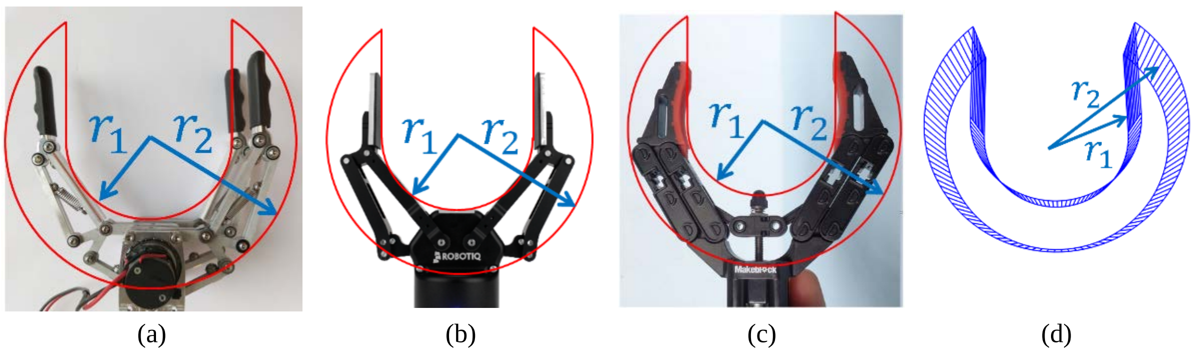


Figure 6.1: Three widely used commercial under-actuated grippers and the approximation of C-shape.

The goal of this chapter is to design a fast and general grasping algorithm for unknown objects. To achieve this goal, the rest of this chapter is organized as follows: Section 6.2 illustrates our fast grasping algorithm. Section 6.3 compares different online motion planners. Section 6.4 presents the simulation results. Section 6.5 gives the experimental results and Section 6.6 gives a discussion about our fast grasping algorithm and the other aforementioned five popular fast grasping algorithms. Finally, the conclusions are provided in Section 6.7.

6.2 Fast C-shape grasping for unknown objects

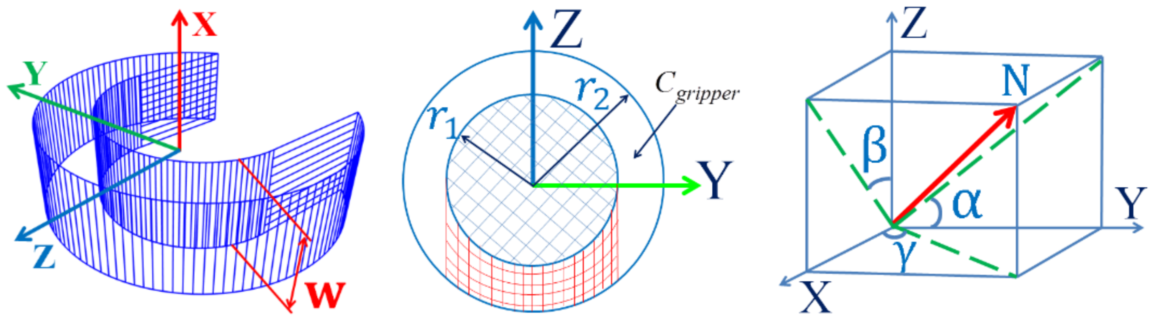
This section firstly presents the mathematical description of the C-shape configuration. Furthermore, our fast grasping algorithm is illustrated, which consists of eight steps.

6.2.1 Mathematical description of the C-shape configuration

The algorithm will direct C-shape searching on the single point cloud of the target object to quickly synthesize an executable grasp. Figure 6.2 (a) shows the C-shape of the under-actuated grippers in Figure 6.1, w is the width of the griper. From Figure 6.2 (b), we can find the space of the C-shape (C_c) equals the outer cylinder space (C_{out}) minus the inner cylinder space (C_{in}) and the red space (C_{red}), shown as Equation (6.1). C_{red} can be approximated as $C_{red} = \{(-0.5w \le x \le 0.5w) \wedge (-r_1 \le y \le r_1) \wedge (-r_2 \le z \le 0)\}$.

$$C_c = C_{out} - C_{in} - C_{red} \tag{6.1}$$

In order to calculate the outer cylinder space (C_{out}) and the inner cylinder space (C_{in}), we must know how to obtain the parametric equation of an arbitrary circle on an arbitrary plane in 3D space. If $P(x_0, y_0, z_0)$ is the center of an arbitrary circle, the radius is r and its unit normal vector is $N = (n_x, n_y, n_z)$ shown as the red arrow in Figure 6.2 (c). If the normal vector is projected to the XOY plane, XOZ plane and YOZ plane, we can obtain three project lines (shown as the three green lines). γ, β and a are used to respectively stand for the angles between the projected lines and the coordinate axes. Then the arbitrary plane can be obtained by transforming the XOY plane through the following transformation: rotating around the X axis by a; rotating around the Y axis by β, then moving along the vector N to P (x_0, y_0, z_0). The whole transformation can be expressed as Equation (6.2).



(a) (b) (c)

Figure 6.2: Mathematical description of the C-shape.

$$T=\begin{bmatrix}1&0&0&0\\0&\cos a&\sin a&0\\0&-\sin a&\cos a&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}\cos\beta&0&\sin\beta&0\\0&1&0&0\\\sin\beta&0&\cos\beta&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\x_0&y_0&z_0&1\end{bmatrix}=\begin{bmatrix}\cos\beta&0&-\sin\beta&0\\\sin a\sin\beta&\cos a&\sin a\cos\beta&0\\\cos a\sin\beta&-\sin a&\cos a\cos\beta&0\\x_0&y_0&z_0&1\end{bmatrix} \qquad (6.2)$$

Assuming that $(x(t), y(t), z(t))$ are used to stand for an arbitrary points on the arbitrary circle, then the parametric equation of the circle can be obtained by the Equation (6.3).

$$\begin{pmatrix}x(t)\\y(t)\\z(t)\\1\end{pmatrix}=\begin{pmatrix}r\cos t\\r\sin t\\0\\1\end{pmatrix}^T *T=\begin{cases}x(t)=x_0+r\cos t\cos\beta+r\sin t\sin a\sin\beta\\y(t)=y_0+r\sin t\cos a\\z(t)=z_0+r\sin t\sin a\cos\beta-r\cos t\sin\beta\end{cases} \qquad (6.3)$$

Where t should satisfy $0\le t\le 2\pi$. If $\{x(s,t), y(s,t), z(s,t)\}$ is an arbitrary point on the cylinder, and the axis vector of the cylinder is $N=(\cos a', \cos\beta', \cos\gamma')$, then parametric equations for an arbitrary cylinder in 3D space can be obtained using Equation (6.4).

$$\begin{cases}x(s,t)=x_0+r\cos t\cos\beta+r\sin t\sin a\sin\beta+s\cos a'\\y(s,t)=y_0+r\sin t\cos a+s\cos\beta'\\z(s,t)=z_0+r\sin t\sin a\cos\beta-r\cos t\sin\beta+s\cos\gamma'\end{cases} \qquad (6.4)$$

In which $0\le s\le w$, w is the width of the griper. Using Equation (6.4), we can derive equations for C_{out} and C_{in}, then we can obtain the math description of the C-shape using Equation (6.1).

6.2.2 Outline of our fast grasping algorithm

This subsection presents a detailed explanation of our fast grasping algorithm for unknown objects. The outline of our fast grasping algorithm is shown in Figure 6.3. It can be seen that eight steps are required to execute fast grasping using our algorithm.

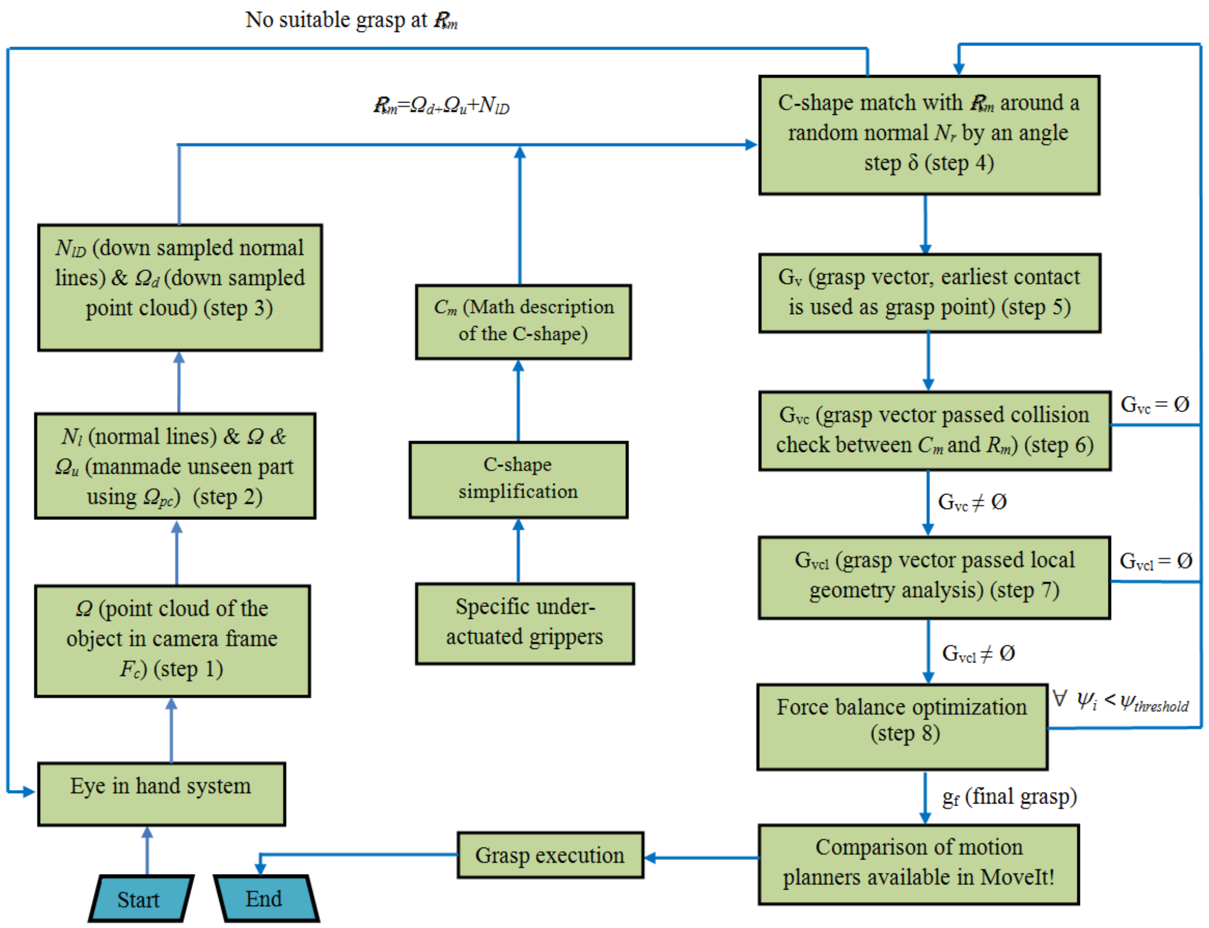
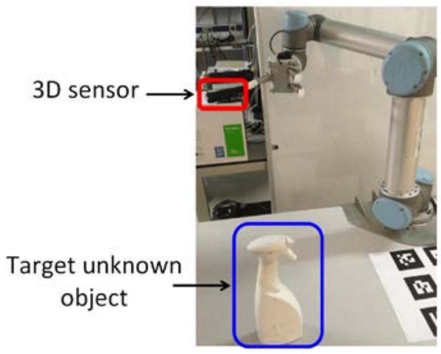

No suitable grasp at R_m
$R_m = \Omega_d + \Omega_u + N_{lD}$
C-shape match with R_m around a random normal N_r by an angle step δ (step 4)
N_{lD} (down sampled normal lines) & Ω_d (down sampled point cloud) (step 3)
C_m (Math description of the C-shape)
G_v (grasp vector, earliest contact is used as grasp point) (step 5)
N_l (normal lines) & Ω & Ω_u (manmade unseen part using Ω_{pc}) (step 2)
C-shape simplification
G_{vc} (grasp vector passed collision check between C_m and R_m) (step 6)
$G_{vc} = \emptyset$
$G_{vc} \neq \emptyset$
Ω (point cloud of the object in camera frame F_c) (step 1)
Specific under-actuated grippers
G_{vcl} (grasp vector passed local geometry analysis) (step 7)
$G_{vcl} = \emptyset$
$G_{vcl} \neq \emptyset$
Force balance optimization (step 8)
$\forall\ \psi_i < \psi_{threshold}$
Eye in hand system
g_f (final grasp)
Grasp execution
Comparison of motion planners available in MoveIt!
Start
End

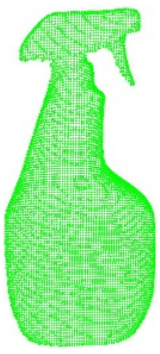
Figure 6.3: The outline of our fast C-shape grasping for unknown objects.

Step 1: Obtaining the point cloud of the target object

Figure 6.4 (a) shows a setup consisting of a robot arm, a 3D sensor and a target unknown object. The raw point cloud obtained from the 3D sensor contains the environment (for example the table plane). In order to quickly extract the point cloud of the target object, down-sampling and distance filtering are firstly applied on the raw point cloud from the 3D camera to reduce the computing time and remove the points out of the reach of the robot arm. Then Random Sample Consensus (RANSAC) method is applied to remove the table plane, resulting in the isolated point cloud of the target object (visualized as the green points in Figure 6.4 (b)).



(a)



(b)

Figure 6.4: Obtaining the point cloud of the target object.

Step 2: Generation of normals

Surface normals are important properties of a geometric surface, and are widely used in many areas such as computer graphics applications. In this chapter, normals are used to guide the configuration of the C-shape to accelerate grasp searching.

The problem of determining the normal to a point on the surface is approximated by the problem of estimating the normal of a plane tangent to the surface, which in turn becomes a least-square plane fitting estimation problem. The solution for estimating the surface normal is therefore reduced to an analysis of the eigenvectors and eigenvalues of a covariance matrix created from the nearest neighbors of the query point. Specifically, for each point P_i, we assemble the covariance matrix C as follows:

$$C = \frac{1}{k}\sum_{i=1}^{k}(P_i - \overline{P})\cdot(P_i - \overline{P})^T \quad C\cdot\vec{V}_j = \lambda_j\cdot\vec{V}_j \qquad j \in \{0,1,2\} \tag{6.5}$$

Where k is the number of points in the neighborhood of P_i, $\overline{P}$ represents the 3D centroid of the nearest neighbors, λ_j is the j-th eigenvalue of the covariance matrix, and $\vec{V}_j$ is the j-th eigenvector. The first eigenvector corresponding to least eigenvalue will be the normal at each neighborhood.

As one normal has two possible directions (the red and blue arrow lines) shown in Figure 6.5, it must be figured out which is the right direction of the normal. Since the point cloud datasets are acquired from a single viewpoint, the camera view point p_c is used to solve the problem of the sign of the normal. The vector from the point p_i to the camera view point p_c is $V_i = p_c - p_i$, To orient all normals $\vec{n}_i$ consistently towards the viewpoint, it must satisfy the equation: $\vec{n}_i \cdot V_i > 0$. Using this equation, we can constrain all the normals towards the camera viewpoint to obtain right normals (shown as all the red lines in Figure 6.5) of the target object.

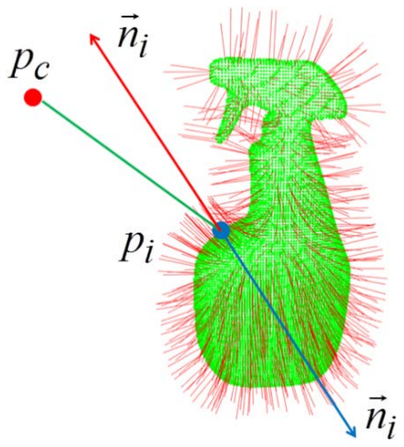


Figure 6.5: Generation of normals of the target object.

Step 3: Down-sampling of point cloud

Normals in Figure 6.5 are too dense. In order to accelerate the speed of grasp searching, the normals are required to be down-sampled. A K-d tree is used to down-sample the normals. The green points in Figure 6.6 (a) stand for the original point cloud (Ω) that is used to compute the normal, Ω is first down-sampled to obtain the down-sampled point cloud Ω_d (shown as the red points in Figure 6.6 (a)). At each red point (P_{di}) of Ω_d, we use KNN search to find the nearest neighbor point (P_i) in Ω (shown as Figure 6.6 (b)). Then the corresponding normal (n_i) of P_i can be looked up in the dense normals obtained in step 2. Eventually, all the corresponding normals are put together to form the down-sampled normals shown as Figure 6.6 (c).

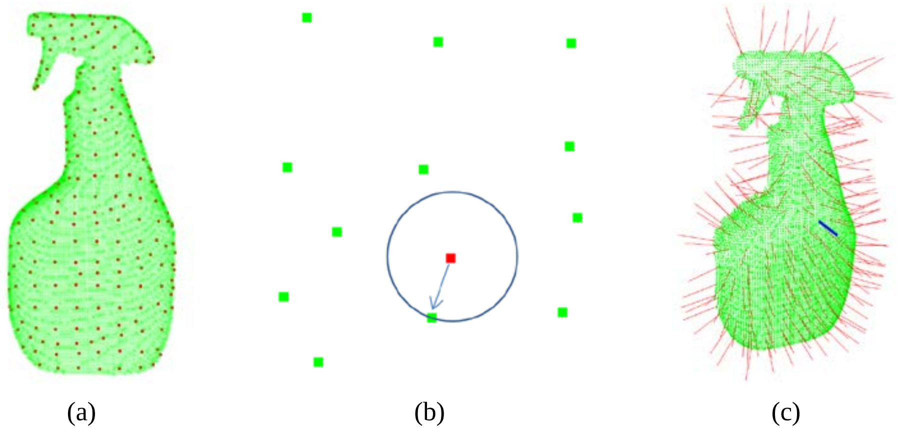


Figure 6.6: Down-sampling of normals of the target object.

Step 4: Effective configuration of a C-shape

In this step, we will explain how to configure the C-shape to find a suitable grasp and how to handle the unseen part of object because we cannot see the back side of the object when we only use a single-view point cloud to search grasps.

Problem of grasping unknown objects can be understood as finding a proper grasping configuration for the robot hand. From the perspective of motion planning, the grasping problem can be formulated as motion planning under the work space of the robot and the configuration space of the target object. In the 3D world, the configuration space (C space) of the target object (C_{obj}) actually follows a SE (3) group. If the object configuration in C_{obj} is q, $q=\{q_x,q_y,q_z,q_{\theta 1},q_{\theta 2},q_{\theta 3}\}$ where q_x, q_y, q_z and $q_{\theta 1}$, $q_{\theta 2}$, $q_{\theta 3}$ correspond to coordinates of position in the Cartesian frame and coordinates along the rotational orientation. $\mathrm{O}[q]$ ($\mathrm{O}[q_{xyz}]$ and $\mathrm{O}[q_\theta]$) is the corresponding target objects at configuration q.

Grasping could be regarded as the configuration of the fingers ($\Gamma_i, i=1,2,...,n$, n is the number of fingers) of the robot in the C_{obj}. Γ_i is formed by independent fingers and corresponds to $f_i, i=1,2,...,n$ in the work space (W space). The configuration of fingers Γ_i can be considered as configuration of obstacles in C_{obj}.

From the perspective of motion planning, the grasping algorithm needs to calculate a pre-grasp. The robot tries to approach the target object by using the pre-grasp. Usually, for grippers without tactile sensors, the grasping algorithm needs to work out the final grasp state. After obtaining this state, what the robot needs to do is just to close its gripper. This is the whole procedure of the unknown object grasping by using under-actuated gripper. The Equation (6.6) shows the final grasp configuration. This equation means the finger configuration belongs to the target object configuration and fingers should have intersection so that the gripper can grasp the target object.

$$\Gamma_i = \{q \mid q \in C_{obj} \wedge (\mathrm{O}[q] \cap f_i \neq \phi)\} \tag{6.6}$$

After obtaining the final grasp configuration, the robot will work out a collision free trajectory to drive the robot arriving at the grasping point. The trajectory planning is carried out in the configuration free space C_{free} (seen in Equation (6.7)).

$$C_{free} = \{q \mid q \in C_{obj} \wedge q \notin \bigcup_{i=1}^{n} \Gamma_i\} \tag{6.7}$$

According to the above analysis, the configuration of the C-shape actually follows a SE (3) group. If we want to locate a C-shape in 3D space, it means many possibilities. In order to reduce the possibilities to accelerate grasping searching, normals of the target object are used to work as the approaching direction of the C-shape. Then the configuration of the C-shape can be simplified from SE(3) to SE(2). Figure 6.7 shows how to configure the C-shape. (a) shows a random normal (the blue line). (b) is an enlarged image of (a). If a normal is chosen as the approaching direction of the C-shape, it means that the Z axis of the C-shape will align with the blue line in (a) and (b). Then the C-shape can only rotate around the normal, therefore the C-shapes are configured around the normal with an incremental angle δ (visualized as Figure 6.7 (b)). Every red line in (a) and (b) represents a possible axis for the C-shape. The X axis of the C-shape will match with every red line to construct a potential grasp candidate. Figure 6.7 (c) shows an example of a potential grasp candidate corresponding to the black axis in (b). The red points in (c) mean the points of the object covered by the C-shape.

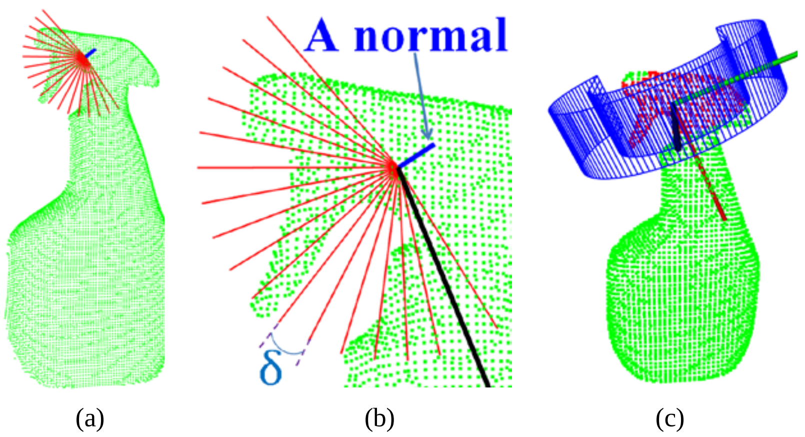


Figure 6.7: Configuration of the C-shape.

As mentioned before, the C-shape axis is allocated around the normal with an incremental angle δ. Then a question comes out, i.e., how to decide the first axis of the C-shape to increase the possibility to find a suitable grasp?

If δ is a big angle, for example 60^o in Figure 6.8 (a) and (b), we may get two totally different allocations of C-shape axis. In Figure 6.8 (a), the three cylinder axis will lead to no grasp found, because all the three C-shapes will collide with the object. However, the C-shape axis 1 in Figure 6.8 (b) corresponds to a very good grasp candidate (shown as Figure 6.8 (c)). The difference is generated because of the position of the first axis. In this chapter, we propose to use the principal axis of the local point cloud to work as the first C-shape axis.

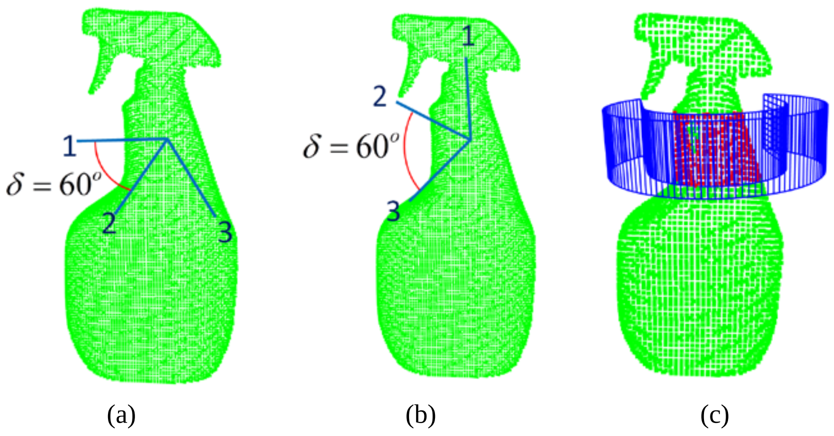


Figure 6.8: How to determine the first configuration of the C-shape.

If the C-shape is configured as Figure 6.9 (a), the gripper will collide with the target object. Because we simply use a single-view partial point cloud of the object in this chapter, the unseen part of the target object will inevitably result in grasp uncertainty. To overcome this, the boundary of the object is employed to eliminate the uncertainty. Specifically, the point cloud in the camera frame is utilized to work out the boundary points Ω_b (visualized in Figure 6.9 (b)). Figure 6.9 (c) shows our idea to deal with the unseen part. In detail, the two red points are on Ω_b, the two orange lines are obtained by connecting the origin point of camera frame and the two red points. The two orange dashed lines are obtained by extending the two orange lines. Using this method goes through all the points on the boundary, we can obtain a point cloud shown as Figure 6.9 (d). Then the configuration space (C space) of the target object (C_{obj}) is divided into two parts. $C_{obj}^{'}$ (the green points in (d)) and C_{unseen} (the orange points in (d)) are used to describe the configuration space after the unseen part is generated. It is shown as Equation (6.8).

$$C_{obj} = C_{obj}^{'} + C_{unseen} \tag{6.8}$$

Step 5: Determination of the center point of the C-shape

As mentioned in step 4, the under-actuated gripper will approach the target object along the normal direction. Then a question comes out, that is, where to stop?

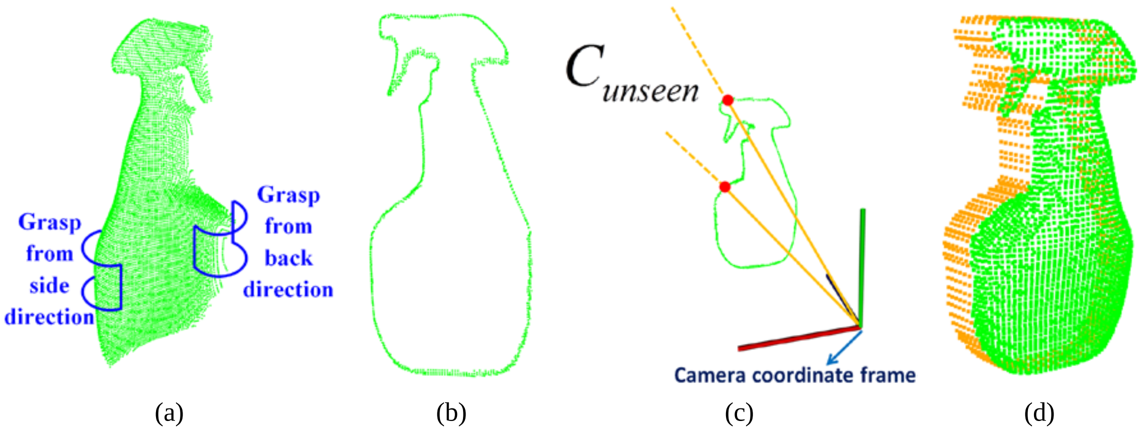


Figure 6.9: Illustration of the solution to deal with the unseen part of the target object to eliminate the grasp uncertainty.

Figure 6.10 illustrates how to determine the center point of the C-shape. Figure 6.10 (a) shows a possible grasp candidate, the green points stand for the points covered by the C-shape. Figure 6.10 (b) is the abstracted point cloud, and the red arrow stands for the approaching direction of the C-shape. The two red points in Figure 6.10 (b) are two example center points of the C-shape. The two blue circles stand for the corresponding C-shape. It is obvious to find that the two example center points of the C-shape are not the best ones. The center point can go down further. (c), (d) and (e) elaborate how to determine the center point of the C-shape. Specifically, the abstracted point cloud in (b) is first projected to the YOZ plane to get the projected point cloud (orange points shown as (c)). Then the convex hull of the projected point cloud is extracted shown as the green points in (c). The green point in Figure 6.10 (d) means one point of the convex hull obtained in (c). If we draw a circle with r_1 as radius (shown as the green circle), we can obtain two intersects with the Z axis (shown as the two purple points P_1 and P_2). $Z = \min(Z_1, Z_2)$ will work as the C-shape center. Using the method goes through all the green points in (c), we can get all the center points $Z_c = (Z_{c1}, Z_{c2}, \cdots, Z_{cn})$ (shown as (e)). The maximal Z_c is used as the final C-shape center (shown as the Equation (6.9)). The maximal Z_c means the earliest contact point with the object when the C-shape tries to approach the object.

$$Z_{c\_\max} = \max(Z_{c1}, Z_{c2}, \cdots, Z_{cn}) \tag{6.9}$$

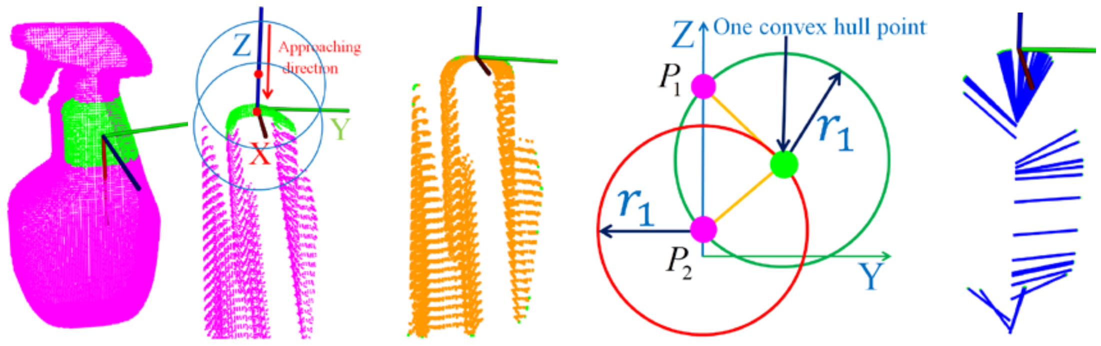


Figure 6.10: How to determine the center point of the C-shape.

Step 6: Collision analysis of the C-shape

Figure 6.11 (a) shows an example of C-shape configuration. After the configuration of the C-shape is obtained, we need to judge whether this configuration will collide with the object or not? If the C-shape will not collide with the object, then it means this configuration is possible to be an executable grasp candidate, otherwise this configuration should be abandoned.

In order to judge whether one configuration will collide with the object or not, points with X axis value between $-0.5w$ and $0.5w$ are abstracted to form a point cloud $\Omega_{[-0.5w,-0.5w]}$ (shown as the red points in Figure 6.11 (a), w is the width of the gripper). If any points p_i of $\Omega_{[-0.5w,-0.5w]}$ falls inside of the C-shape space, it means the C-shape will collide with the target object, then the grasp candidate g_i should be removed, otherwise g_i is reserved for following analysis. Applying this method to all the C-shape configurations, it leads to a vector $G = (g_1, g_2 ... g_n)$ which is used to store all grasp candidates without collision with the target object.

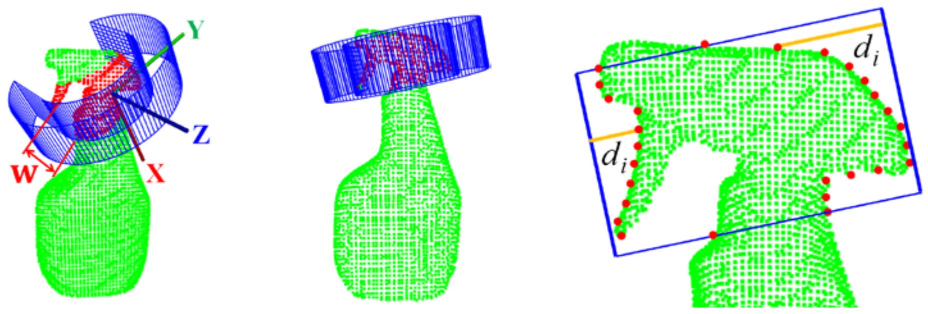


Figure 6.11: Analyzing one grasp formed by a C-shape.

Step 7: Local geometry analysis

After finishing all above steps, the grasps left can ensure that the C-shape will not collide with the object, it means that the C-shape can envelope the object at this configuration. In this step, we will consider the local geometry of the points enveloped by the C-shape. Specifically, a grasp candidate is shown as Figure 6.11 (b), the local geometry shape may lead to grasp uncertainty. Two grasp sides are abstracted shown as the red points in (c), then, the distance between one red point and the blue line is defined as d_i, $0 < i \le n$, n is the total number of the red points. All the distances are added together to get the variance v of the grasp, $v = \sum_{i=1}^{i=n} d_i$. If the variance is smaller than the threshold set by the system, the grasp is saved, otherwise, it is removed.

Step 8: Force balance optimization

All grasp candidates passed previous steps can form a new vector $G_j = (g_{j1}, g_{j2} ... g_{jn})$, all the grasps in this vector can be executed without collision with the target object. If the lines 1, 2, 3, 4, 5, 6 and 7 in Figure 6.12 (a) stand for the C-shape axis of the grasps in vector G_j, we can find that all the grasps from g_{j1} to g_{j7} can be executed. How to choose the best grasp as the final grasp?

We propose to use force balance optimization to select out the best grasp. Usually, the existing grasping approaches will employ the physical property to do force balance computation, for example, the friction coefficient. But in our case, we cannot know the physic property, because the objects for this chapter are unknown. We propose to use the local geometry shape to do approximate force balance computation. The blue points in Figure 6.12 (b) stand for the grasp candidate 1 (g_{j1}). It is projected to the XOY plane to get the projected point cloud shown as (c). The two grasp sides are abstracted to shown as the red points in (d). Two orange lines ($y = kx + b$) can be fit out for the tow grasp sides. The two angles between the two fit lines and X axis are defined as ξ and θ. (e) shows three cases of allocation of ξ and θ.

The sum (σ) of ξ and θ is used to evaluate the force balance quality of this grasp. σ can be obtained using $\sigma = fabs(\arctan(k_\theta)) + fabs(\arctan(k_\xi))$. The bigger σ is, the higher possibility that the grasp forces are vertical to the grasp sides, correspondingly more stable the grasp is. The vector $\psi = (\psi_1, \psi_2 ... \psi_7)$ is used to stand for all the force balance coefficients for the grasp vector $G_j = (g_{j1}, g_{j2} ... g_{j7})$. Figure 6.12 (f) is a line graph of the vector ψ, the grasp with the largest ψ is chosen as the final grasp. Figure 6.12 (g) shows the best grasp returned, which corresponds to the 4th grasp in (a) and (f).

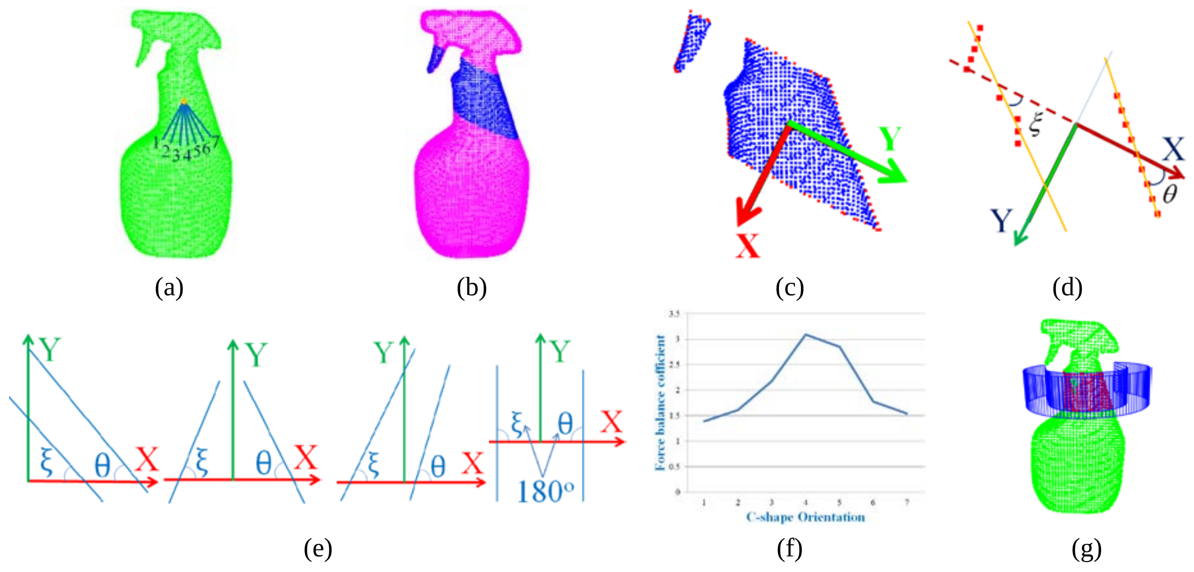


Figure 6.12: Choose the best grasp using force balance optimization.

The above steps from step 1 to step 8 illustrate how the grasping algorithm work to find a suitable grasp at one normal of the target object. If the grasping algorithm cannot find a suitable grasp at one normal, another random normal will be used to repeat above steps until a suitable grasp is found.

6.3 Selection of motion planners for grasping execution

Typically not a lot of grasping algorithms give details about the actual motion planning of the robotic arm towards the object. Grasping algorithms seem to only focus on finding grasps on the object itself. Researchers and users that want to implement grasping algorithms have to fill the gap of motion planning. They have to study on many different available motion planning methods before implementing it, which is time consuming. In order to help future researchers and users quickly choose a suitable motion planner to execute grasp action, we will conduct a comparison of different available online motion planners.

Motion planning is a very important part for grasp execution. However, typically not a lot of grasping algorithms give details about the actual motion planning of the robotic arm. MoveIt!

[152], a motion planning interface in ROS, is easy to use and therefor widely used for robot manipulation. In this section, we will discuss the choice of motion planner by looking at the available motion planning methods in MoveIt! and by evaluating benchmark data.

6.3.1 Motion planning using MoveIt!

Performance of motion planning depends on the chosen motion planner. MoveIt! itself does not provide motion planning, but instead it is designed to work with planners or planning libraries. Currently four main planners/planning libraries can be configured to use.

OMPL (Open Motion Planning Library) [153] is a popular choice to solve a motion problem. It is an open-source motion planning library that houses many state-of-the-art sampling based motion planners. OMPL is configured as the default set of planners for MoveIt!. Currently 23 sampling-based motion planners can be selected for use.

STOMP (Stochastic Trajectory Optimization for Motion Planning) [154] is an optimization-based motion planner. It is designed to plan smooth trajectories for robotic arms. The planner is currently partially supported in MoveIt!

CHOMP (Covariant Hamiltonian Optimization for Motion Planning) [155] mainly operates by using two terms. The dynamical quantity term describes the smoothness of the trajectory. The obstacle term is similar to potential fields. The planner is not yet configured in the latest version of MoveIt!.

Search-Based Planning Library (SBPL) [156] consists of a set of planners using search-based planning that discretize the space. The library is not yet configured in the latest version of MoveIt!.

Among these four, OMPL is chosen to perform motion planning in MoveIt! to compare the different motion planners. OMPL gives us a wide variety of choice to solve a motion planning problem since it contains 23 planners.

6.3.2 Overview of OMPL planners available in MoveIt!

Sampling-based motion planners in OMPL work by constructing roadmaps in the configuration space of the robot. This is done by connecting sampled configuration states with each other. The planners are widely used due to their success in finding feasible paths in high dimensional and geometrically constraint environments. Moreover, they are proven to be probabilistically complete [165]. Asymptotically optimal planners can refrain from potential high-cost paths and rough motions [166]. However, computational effort for finding an optimal path is increased.

The planners from Table 6.1 can be divided into multi-query and single-query planning methods. A well-known multi-query method is the Probabilistic Road Map (PRM) [165]. The planner attempts to find a path by using a roadmap in the configuration space of the robot. The construction of the roadmap is done by sampling valid nodes which resemble configuration states and connect these nodes to other nearby nodes by edges. Once the roadmap construction is finished a simple graph search is performed to find a motion plan in the roadmap from start node to goal node. Because the algorithm covers the total configuration space with a roadmap, it can be used again to find a different start-goal motion plan, this is why this planner is a multi-query method. Variants of PRM exist for use in MoveIt!. The LazyPRM [160] initially does not check for valid states when sampling states for roadmap construction in the configuration space. Once a path has been found from start to goal state, collision checking is performed along the nodes and edges of the roadmap. Invalid nodes and edges are removed and a new search is attempted. This process is repeated until a feasible path is found. PRM* [166] is the asymptotically optimal algorithm of the PRM planner. It rewires nodes to other nearby nodes if this is beneficial to the cost towards the node. LazyPRMstar [166] is a combination of the LazyPRM and PRM* algorithms.

Table 6.1: Available planners of OMPL in MoveIt!

Planner name	Reference	Asymptotically optimal	Time-invariant goal
SBL	[157]		√
EST	[158]		√
BiEST	Based on [158]		√
ProjEST	Based on [158]		√
KPIECE	[159]		√
BKPIECE	Based on [159]		√
LBKPIECE	Based on [159][160]		√
RRT	[161]		√
RRTConnect	[162]		√
PDST	[163]		√
STRIDE	[164]		√
PRM	[165]		
LazyPRM	[160]		
RRTstar	[166]	√	
PRMstar	Based on [165][166]	√	
LazyPRMstar	Based on [160][166]	√	
FMT	[167]	√	√
BFMT	[168]	√	√
LBTRRT	[169]	√	√
TRRT	[170]	√	√
BiTRRT	[171]	√	√
SPARS	[172]	√	
SPARStwo	[173]	√	

The SPARS which is similar to PRM* but adds another sparse subgraph. This subgraph is an asymptotically optimal roadmap that houses nodes which resemble multiple nodes in a dense graph. SPARStwo is a variant of this algorithm that has an infinite iteration loop.

Single-query methods create a new roadmap every time a new planning query has to be determined. The most common single-query planner is the Rapidly Exploring Random Tree (RRT) method [161]. It creates a tree from the initial configuration state in the direction of the unexplored areas of the bounded free space. This is done by picking a valid random node, the algorithm in turn checks if this node can be added to the tree by determining if the nearest node of the tree is within a specified distance. This process is done at every iteration until a tree is grown that reaches the goal node. The RRTConnect method [162] is a bi-directional version of the RRT method, meaning that two trees are grown. One tree is grown from the start node and one from the goal node when the two trees can be connected a path is found that solves the motion problem. The near-optimal variant of RRT called RRT* [166] checks whether the new sampled node can be connected to other nearby nodes so that the state space is more locally refined. The RRT* removes the connections of the new sample that are not beneficial towards the cost of the path. When the number of random samples is big enough it would result in a near-optimal path from the start-to-goal state. The RRT* planner implementation in within OMPL and MoveIt! keeps trying to optimize the trees by adding new nodes until specified time limit is met. Lower Bound Tree-RRT (LBT-RRT) [169] is a near-optimal planner, it uses a so-called lower bound graph which is an auxiliary graph and it uses a similar to RRT* method to maintain the tree. Transition-based RRT or TRRT [170] is a combination of the RRT method with a stochastic optimization method for global minima. It computes transition tests to accept new states to the tree. The algorithm computes a near-optimal path that is not tied to a user specified time limit like RRT*, meaning that the planner stops as soon as it found a connection between start and goal node. The Bi-TRRT [171] is a bi-directional version of this planner.

The EST method [158] stands for Expansive Space Trees, it was developed in the same period as RRT. Other than RRT the EST algorithm tries to determine the direction of the tree by looking at neighboring nodes and then grow in the less explored area in the configuration space. Bi-directional EST (BiEST), based on [158], grows two trees from the start and goal state respectively. The algorithm tries to connect the trees at every iteration such that a path between start and goal state can be established. Projection EST (ProjEST), based on [158], detects the less explored area of the configuration space by using a grid, this grid is served as a projection of the state space. Single-query Bi-directional probabilistic roadmap planner with Lazy collision checking, also called SBL that grows two trees. The trees expand in the same manner as the EST planner. Due to its lazy collision checking it will determine if a path is valid after the two trees are connected, like LazyPRM.

KPIECE (Kinodynamic motion Planning by Interior-Exterior Cell Exploration) [159] is a tree-based planner that uses layers of discretization to help estimate the coverage of the state space. The OMPL implementation uses only one layer. OMPL incorporates a bi-directional variant called BKPIECE and a variant which builds on the latter by incorporating lazy collision checking, this variant is called the LBKPIECE.

Fast Marching Tree (FMT) [167] is an asymptotically optimal planner which marches a tree forward in the cost-to-come space on a specified amount of samples. The BFMT [168] planner is a bi-directional variant of this planner.

PDST (Path-Directed Subdivision Tree) [163] represents samples as parts of a path instead of configuration states and uses non-uniform subdivisions to explore the unexplored state space.

STRIDE (Search Tree with Resolution Independent Density Estimation) [164] uses a Geometric Near-neighbor Access Tree (GNAT) to sample the density of the configuration space. This information helps to guide the planner into the less explored area.

6.3.3 Methodologies of comparing motion planners in MoveIt!

In order to compare the performance of the 23 motion planners available in MoveIt!, we created two benchmarks shown in Figure 6.13. The first benchmark resembles a grasp among dense obstacles. The second benchmark resembles a long motion grasp.

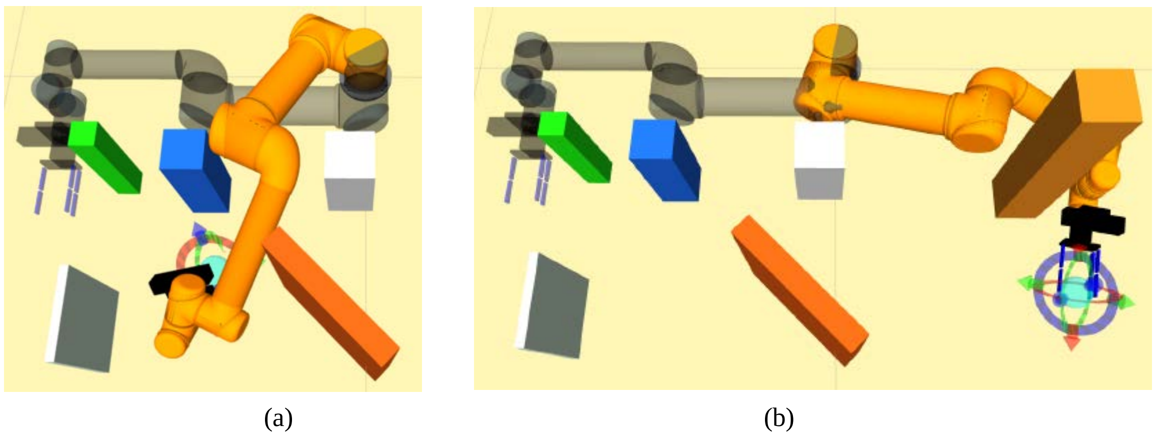


Figure 6.13: Simulation setting for comparison of different motion planners in MoveIt!. (a): Benchmark 1: Grasp among dense obstacles. (b): Benchmark 2: The robot arm needs long motion path for grasping.

The planners are analyzed on the three respects of the solved runs, computing time and path length. Solved runs, computing time and path length are used as metric in our experiments. We analyze the measures individually to provide the best performing planners in each one of the measures. Solved runs is analyzed by terms of percentage of total runs of the planner resulting in feasible paths, higher performance is considered for higher solved runs. Total computing time is measured for the time it takes for planners to produce feasible or optimal paths with path simplification, a shorter time is considered as higher performance. Moreover, planners with a small standard deviation from the average computing time and small interquartile range are considered as better performance. Path length is measured by the length of the sum of motions for a produced path. Shorter lengths are considered as higher performance. Again, planners with a small standard deviation from the average path length and small interquartile range are considered as better performance.

The benchmarking experiments are performed using one thread on a system with an Intel i5 2.70 GHz processor and 8GB of memory. In order to obtain reliable data on the solved runs, computing time and path length, each algorithm was run 30 times for the given motion planning problem. The algorithms were given a maximum computing time of 3s and 10s to show the effect of time on different motion planners. The times are kept low since most robotics applications need to operate quickly.

6.3.4 Parameter selection

Parameters can be set to improve the performance of the motion planners. In this subsection, the parameter selection is presented as shown in Table 6.2. While conducting parameter selections for LBTRRT, we found that this planner is behaving unreliable in our setup. We tested all parameter combinations for this planner when conducting motion planning, however, all parameter combinations resulted in crashes. So we are unable to provide benchmark data for this particular planner.

Table 6.2: Specified planner parameters

SBL	EST	BiEST	ProjEST	RRT	RRTConnect	PRM	LazyPRM	RRTstar	
range: .3125	range: .625	range: 0	range: .625	range: 0	range: .3125	max n.n.: 10	range: .3125	range: 0	
	goal bias: .05		goal bias: .05	goal bias: .05				goal bias: .05	
								delay c.c.: 0	
KPIECE	BKPIECE	LBKPIECE	STRIDE	FMT	BFMT	TRRT	BiTRRT	SPARS	SPARStwo
range: .625	range: .3125	range: .3125	range: .625	samples: 1000	samples: 1000	range: 1.25	range: 1.25	str. factor: 2.6	str. factor: 3
goal bias: .05	border frac.: .9	border frac.: .9	goal bias: .05	rad. mult.: 1.05	rad. mult.: 1.05	goal bias: .05	temp c. fact.: .2	sp. d. frac.: .25	sp. d. frac.: .25
border frac.: .9	failed e.s.f.: .5	min.v.p.frac.: .5	use proj.dist.: 0	nearest k: 1	nearest k: 1	max s. f.: 10	init temp.: 50	d. d. frac.: .001	d. d. frac.: .001
failed e.s.f.: .5	min.v.p.frac.: .5		degree: 8	cache cc: 1	balanced: 1	temp c. fact.: 2	f. threshold: 0	max fails: 1000	max fails: 5000
min.v.p.frac.: .5			max degree: 12	heuristics: 1	optimality: 0	m.temp.: 1e-10	f. n. ratio: .1		
			min degree: 6	extended fmt: 1	cache cc: 1	i.temp.: 1e-6	cost.thres.: 5e4		
			max p.p. leaf: 3		heuristics: 1	f. threshold: 0			
			est. dim.: 0		extended fmt: 1	f.NodeRatio: .1			
			min.v.p.frac.: .1			k constant: 0			

Global planner parameter: There is one parameter that affects all planners. That is the distance parameter (longest_valid_segment_fraction). This parameter is called when the planner checks for collisions between two nodes. Collision detection is not checked for the motion if the distance between the nodes is within the parameter value. In narrow passages and corners, this parameter can be critical. The parameter is set in meters and by default has a value of 0.005m. After conducting experiments with lower values, we found that reducing this parameter did not have an immediate effect on the solved runs for both of these two benchmark problems in Figure 6.13.

Specific parameters for planners: The majority of planners (20 of 23) have their own parameters. For the two benchmarks, each parameter was set to values that benefit one or more performance measures, these values are listed in Table 6.2.

Robot: The UR5 robot that will be used has two joint limit settings for each joint, π and 2π. Validating by means of simple motion planning experiments, we found that setting the joint limits to π resulted in favorable performance for all the performance measures.

6.3.5 Comparison results

Results of benchmark 1 are shown in Figure 6.14 and Table 6.3. The motion problem affects planners EST, RRT, RRTstar, TRRT and SPARStwo since they were not able to solve all the runs with a percentage higher than 50% with a maximum computing time of 3s. For 10s of computing time, more solved runs were retrieved. SBL, BiEST, KPIECE, BKPIECE and LBKPIECE compute valid paths in a computing time shorter than 1 s. RRTConnect is the fastest planner and BiTRRT is the fastest asymptotically optimal planner. RRTConnect paths have the lowest median. However, the average is higher due to significant outliers. SBL has the lowest average path length with a small standard deviation. Planners SBL, KPIECE, LBKPIECE, FMT, and TRRT are able to plan paths of similar median and average lengths. For asymptotically optimal planners, BiTRRT has the lowest median path length. TRRT has the lowest average path length and standard deviation. Selecting a higher limit of computing time did not result in significant changes.

Results of benchmark 2 are shown in Figure 6.15 and Table 6.4. RRTstar, TRRT and SPARStwo have a lower solved runs compared to the other planner algorithms. SBL, BiEST, BKPIECE, LBKPIECE, RRTConnect and BiTRRT compute paths in under 0.1s, SBL is the fastest planner. Planners that have a time invariant stopping goal, except for FMT and TRRT, are producing valid paths within 1s. BiTRRT is the fastest asymptotically optimal planner. Bi-directional planner variants compute valid paths faster. SBL and BiTRRT have the shortest paths. The planners that keep sampling the configuration space or optimizing the path until the maximum computing time is reached see improved performance with respect to path length.

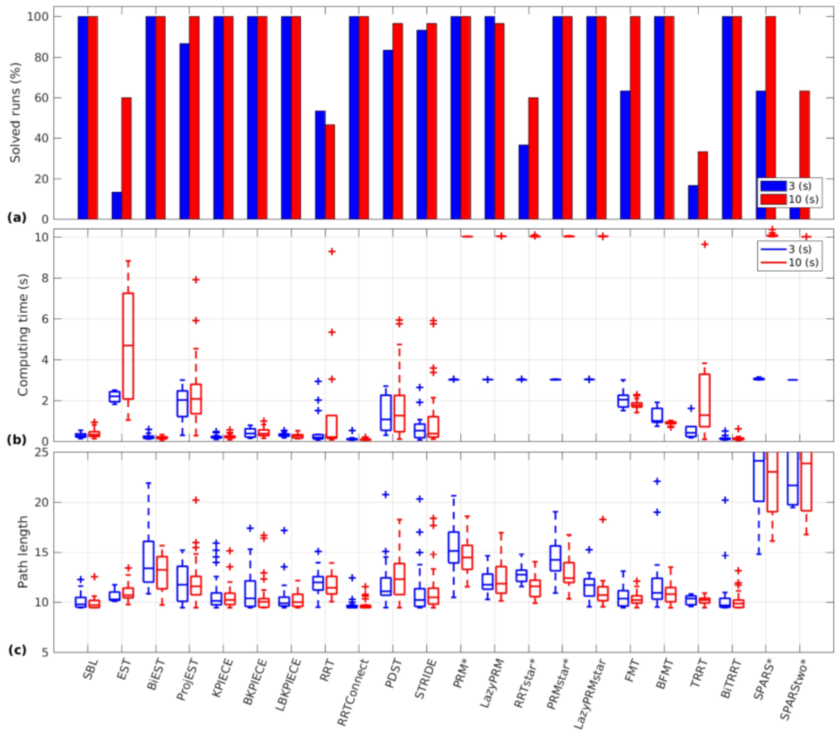


Figure 6.14: Comparison results of 23 motion planners in MoveIt! for benchmark 1.

To summarize, we can find that SBL, BKPIECE, LBKPIECE, RRTConnect and BiTRRT achieve better performance than the other planners from the comparison results of benchmark 1 and benchmark 2. These five planners can work better in the both circumstances of grasp in dense obstacles and grasp in a long motion. However, among these five planners, only BiTRRT is asymptotically optimal. Asymptotically optimal planners are able to exclude potential high-cost paths and rough motions, which can help to achieve a smoother path of the manipulator. Taking all factors into consideration, we will adopt BiTRRT to work as the motion planner for the UR5 manipulator to execute the final grasp in this chapter.

Table 6.3: Average values for benchmark 1

Planner name	Max. 3s computing time		Max. 10s computing time	
	Time (s)	Path length	Time (s)	Path length
SBL	0.29 (0.11)	10.07 (0.74)	0.37 (0.18)	9.90 (0.63)
EST	2.18 (0.31)	10.58 (0.77)	4.65 (2.55)	11.03 (1.01)
BiEST	0.21 (0.10)	14.81 (5.19)	0.18 (0.07)	13.32 (3.14)
ProjEST	1.83 (0.86)	11.82 (1.81)	2.37 (1.57)	12.13 (2.19)
KPIECE	0.20 (0.09)	10.89 (1.80)	0.22 (0.10)	10.55 (1.28)
BKPIECE	0.42 (0.21)	10.94 (1.86)	0.42 (0.21)	10.56 (1.82)
LBKPIECE	0.30 (0.08)	10.41 (1.53)	0.26 (0.11)	12.30 (7.16)
RRT	0.54 (0.84)	11.93 (1.41)	1.48 (2.71)	11.62 (1.14)
RRTConnect	0.11 (0.08)	12.52 (15.68)	0.09 (0.03)	11.89 (9.10)
PDST	1.37 (0.87)	11.96 (2.35)	1.68 (1.61)	12.37 (2.15)
STRIDE	0.59 (0.57)	11.97 (5.35)	1.12 (1.58)	11.20 (2.24)
PRM*	3.01 (0.01)	15.60 (2.26)	10.01 (0.01)	14.49 (1.65)
LazyPRM	3.02 (0.00)	12.13 (1.17)	10.02 (0.01)	12.48 (1.96)
RRTstar*	3.01 (0.01)	12.76 (0.93)	10.02 (0.02)	11.47 (1.03)
PRMstar*	3.02 (0.01)	14.43 (1.90)	10.02 (0.01)	12.99 (1.67)
LazyPRMstar	3.02 (0.00)	11.53 (1.23)	10.03 (0.01)	10.95 (1.59)
FMT	2.07 (0.45)	10.49 (0.99)	1.78 (0.21)	10.33 (0.64)
BFMT	1.17 (0.36)	11.74 (2.65)	0.89 (0.09)	10.88 (1.06)
TRRT	0.57 (0.58)	10.21 (0.52)	2.41 (2.82)	10.12 (0.45)
BiTRRT	0.13 (0.08)	15.56 (16.75)	0.13 (0.10)	11.03 (5.22)
SPARS*	3.04 (0.04)	23.63 (4.74)	10.07 (0.07)	23.87 (5.57)
SPARStwo*	3.00 (0.00)	22.98 (3.97)	10.00 (0.00)	26.34 (10.01)

Table 6.4: Average values for benchmark 2

Planner name	Max. 3s computing time		Max. 10s computing time	
	Time (s)	Path length	Time (s)	Path length
SBL	0.05 (0.01)	9.48 (7.86)	0.04 (0.01)	7.76 (1.76)
EST	0.20 (0.13)	9.53 (2.58)	0.16 (0.11)	9.38 (4.00)
BiEST	0.09 (0.04)	11.36 (1.96)	0.08 (0.03)	12.71 (3.57)
ProjEST	0.18 (0.11)	9.86 (2.51)	0.15 (0.09)	8.99 (1.32)
KPIECE	0.18 (0.09)	9.10 (1.50)	0.14 (0.08)	9.49 (1.68)
BKPIECE	0.11 (0.11)	9.71 (5.83)	0.13 (0.16)	8.17 (2.79)
LBKPIECE	0.09 (0.06)	9.33 (5.53)	0.08 (0.03)	9.23 (3.80)
RRT	0.53 (0.62)	12.03 (7.08)	0.44 (1.10)	10.15 (1.99)
RRTConnect	0.09 (0.04)	13.90 (10.39)	0.06 (0.02)	9.65 (4.05)
PDST	0.24 (0.16)	11.71 (3.56)	0.24 (0.16)	12.27 (4.00)
STRIDE	0.19 (0.21)	9.42 (3.26)	0.14 (0.09)	8.98 (1.17)
PRM*	3.02 (0.01)	12.91 (3.41)	10.01 (0.00)	11.56 (1.56)
LazyPRM	3.02 (0.00)	9.80 (1.88)	10.02 (0.00)	9.58 (1.27)
RRTstar*	3.01 (0.02)	8.78 (0.00)	10.01 (0.01)	8.21 (0.94)
PRMstar*	3.03 (0.01)	12.30 (1.81)	10.02 (0.01)	11.11 (1.65)
LazyPRMstar	3.02 (0.00)	8.62 (0.98)	10.02 (0.01)	7.88 (0.72)
FMT	1.23 (0.16)	9.64 (6.44)	1.10 (0.15)	7.92 (1.15)
BFMT	0.79 (0.06)	8.24 (0.69)	0.73 (0.06)	8.35 (1.64)
TRRT	0.78 (1.00)	7.82 (0.91)	2.05 (2.43)	8.55 (2.17)
BiTRRT	0.08 (0.02)	8.39 (3.00)	0.07 (0.02)	7.75 (1.11)
SPARS*	3.05 (0.04)	19.38 (8.05)	10.07 (0.06)	16.22 (5.38)
SPARStwo*	3.00 (0.01)	14.98 (5.37)	10.00 (0.00)	20.10 (9.43)

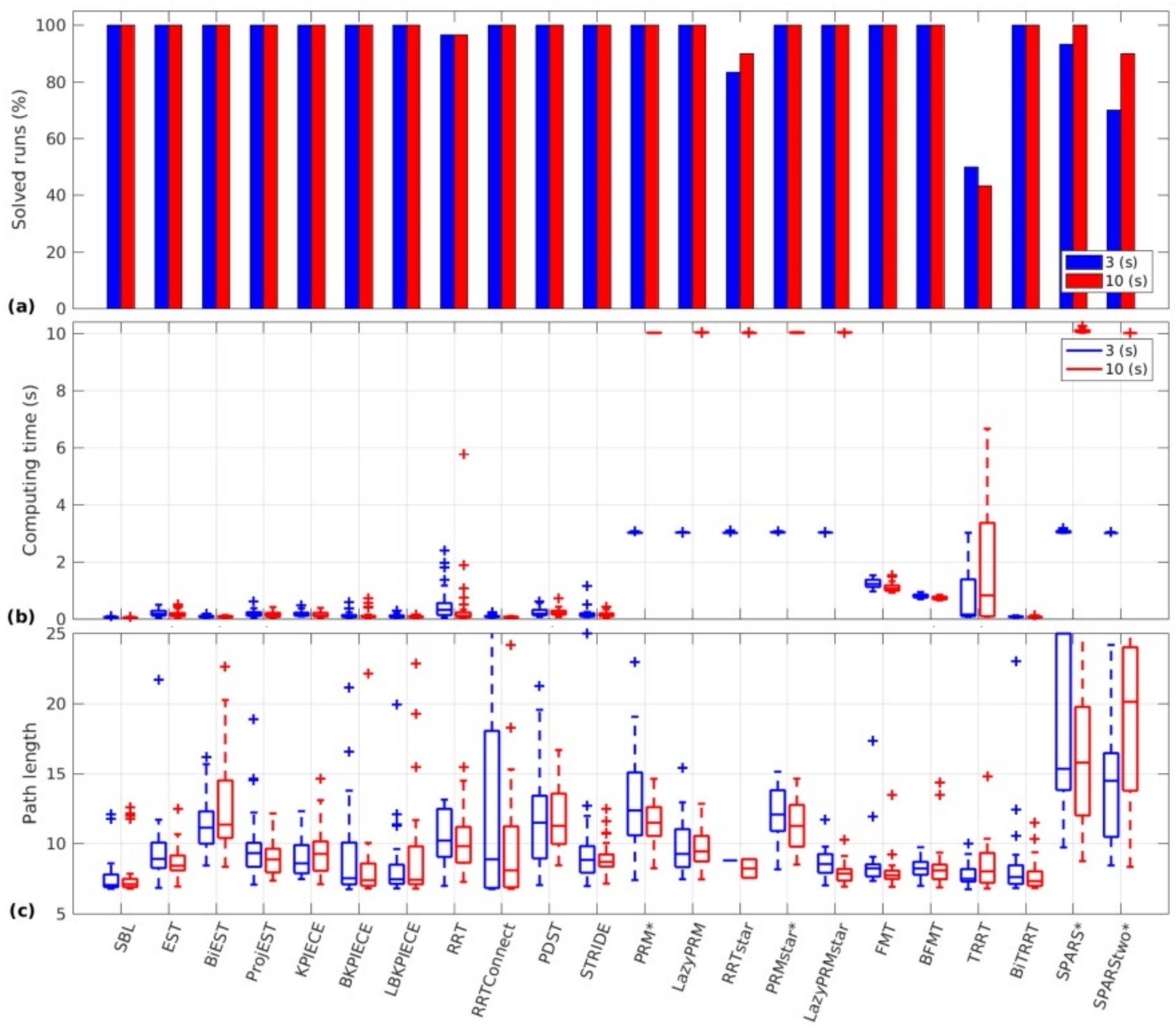
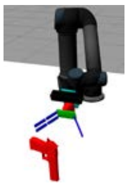
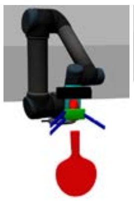
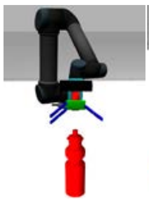
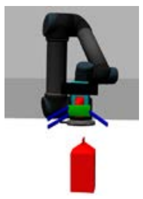
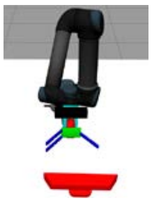
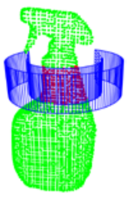
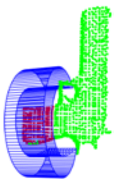
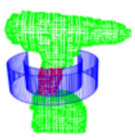
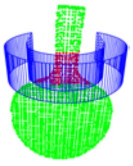
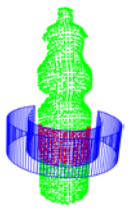
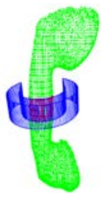
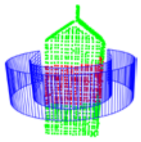
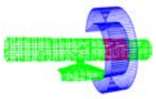
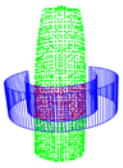
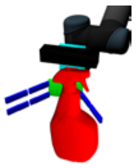
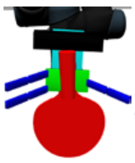
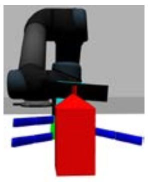
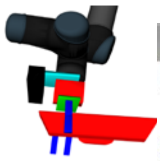
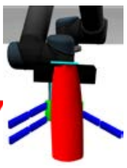


Figure 6.15: Comparison results of 23 motion planners in MoveIt! for benchmark 2.

6.4 Simulation

In order to verify our grasping algorithm, simulations are performed using a personal computer (2 cores, 2.9 GHz). Several objects with different geometry shapes are used in the simulation. All the tested objects can be seen in the second row of Table 6.5. The third row shows an example grasp found by the grasping algorithm. The fourth row shows the robot arm arrived at the grasp point by using BiTRRT as motion planner. The fifth row shows the number of points of the input partial point cloud. The last row shows the average computing time (10 trials for each object). From the simulation, we can find that the algorithm can quickly work out a suitable grasp within 2 seconds for each object.

Table 6.5: Simulation results

Object name	Cleaner spray bottle	Pistol	Electric drill	Table tennis racket	Water bottle	Telephone horn	Milk carton	Kinect	Shampoo bottle
Initial setup									
Example grasp found									
Grasp execution									
Points	8154	4394	7678	6384	7270	12458	4710	4965	5274
Time (s)	1.95	0.89	1.83	1.31	0.87	1.86	0.73	0.92	0.58

6.5 Experiments

The experiments are conducted using a robot arm UR5 and an under-actuated Lacquey Fetch gripper. An Xtion pro live sensor is used to acquire the partial point cloud of the target object. The whole experiment setup and the objects chosen to do experiments are visualized as Figure 6.16. The results of experiments are shown as Table 6.6. The second row shows the experiment setup for every object. The third row shows the example grasp found by the grasping algorithm. The fourth row shows the robot arm arrives ate the grasp position by using BiTRRT as motion planner. The fifth row shows the grasp being executed. The sixth row shows the number of points of the input partial point cloud. The last row shows the computing time (10 trials for each object). The experiments proved the validation of our grasping algorithm. The main difference between the simulations and the experiments is that the point cloud in experiments may lose some points. For example, the coffee jar in the sixth column of Table 6.6 lost some points because the Xtion pro live sensor cannot detect transparent part. The neck of the coffee jar is transparent, so we cannot find the points for neck of the coffee jar. That is why we paint the wineglass in ninth column into white color. From Table 6.6, we can see that even though the partial point cloud of the object has large number of points, our algorithm can quickly work out a suitable grasp within 2 seconds. Comparing with the five fast grasping algorithms [84, 94, 110, 113, 114], our algorithm shows much improvement at the speed of grasp searching.

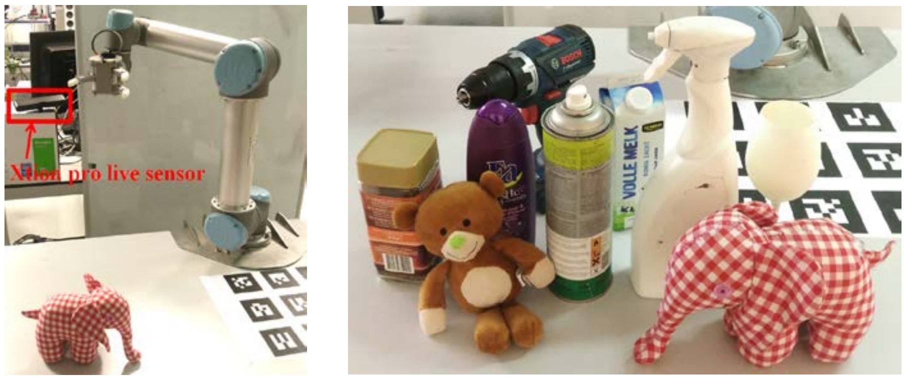
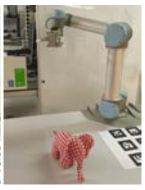
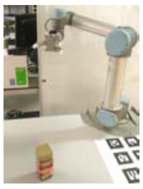
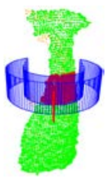
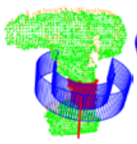
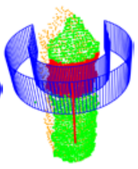
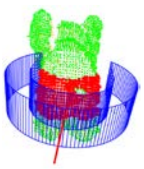
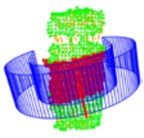
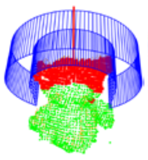
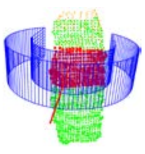
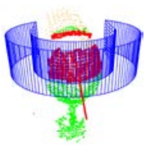
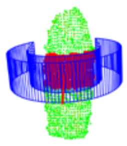
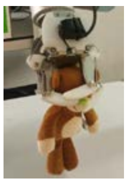


Figure 6.16: Experiments setup and objects used for experiments.

Table 6.6: Experiment results

Object name	Cleaner spray bottle	Electric drill	Spray can	Elephant	Coffee jar	Teddy bear	Milk carton	Wineglass	Shampoo bottle
Initial setup									
Example grasp found									
Grasp execution									
Objects grasped									
Points	10596	9929	7127	8044	4345	4857	5589	3503	5267
Time (s)	1.74	1.56	0.91	1.96	0.68	1.82	0.64	0.53	0.67

6.6 Discussion

In this section, we will discuss the characteristics of our grasping algorithm compared with the existing fast grasping algorithms [84, 94, 110, 113, 114] for unknown objects.

Grasp adaptiveness: Our grasping algorithm is specially designed for under-actuated grippers. Under-actuated grippers add compliance and dexterity without the need of adding additional actuators and sensors. Through the careful design of the end effector's mechanical makeup, under-actuated grippers have great advantages over parallel grippers. Therefore, our grasping algorithm is more adaptive than [84, 110, 113, 114]. Meanwhile, the price of the under-actuated gripper is much cheaper than [94] which uses a barrett hand.

Object complexity: The presented grasping approach is able to find grasp for complex objects like, teddy bear, elephant, electric drill and the cleaner spray bottle. This makes it better than [84, 110, 114], which only considers simple objects. [94] transforms the objects into simple shapes (cylinder, disk, sphere and box), which may result in loss of details of objects.

Computing time: Our algorithm finds a suitable grasp for complex object within 2 seconds. This is similar to [84, 94, 113, 114]. [110] is able to find a grasp faster since it only uses an RGB image at the cost of losing depth information of the object.

Grasping direction: [84, 110, 114] only consider grasping from top, which can result in unreliable grasp, for example, picking up the wineglass. And in some cases, it is not allowed to grasp the target object from top, for example, objects in fridges or shelves. Our grasping algorithm considers the local geometry property of the object. We use the normal of the object to work as the approaching direction, which resembles a human-like grasp.

Grasp execution: From the five fast grasp algorithms, only [113] considers grasp execution. However, no information was given about motion planning. We showed by performing a comparison that using BiTRRT for grasp execution would result in high solved runs, low computing time and short path length.

6.7 Conclusion

In this chapter, a novel algorithm of unknown object grasping is presented for under-actuated grippers. For the grasping algorithm, the gripper is simplified as a C-shape. In order to find suitable grasp, C-shape searching is performed on the partial point cloud of the target object. To accelerate the computing speed, this algorithm only uses a single view partial point cloud as input. Grasp candidates can be greatly reduced by using the normal line of the target object to guide the configuration of the C-shape. Moreover, we propose an original method to deal with the unseen part of the target object to enhance the grasp security. In order to make the robot arm quickly execute the grasp found by the grasping algorithm, a suitable motion planner has to be selected. We compared all the motion planners available in MoveIt!. The planner, BiTRRT, is chosen for motion planning for the UR5 robot arm due to its high solved runs, low computing time and short path length. Furthermore, the effectiveness of our grasping algorithm is verified using available objects by simulations and experiments. Our grasping algorithm can quickly work out a suitable grasp within two seconds for a test unknown object.

Comparing with the five dominant fast grasping algorithms [84, 94, 110, 113, 114], our algorithm shows much improvement at the speed of grasp searching.

Acknowledgement

The work leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 609206.

7 Conclusions, discussions and future directions

7.1 Conclusions

Grasping of unknown objects with neither appearance data nor object models given in advance is very important for service robots that work in unfamiliar environments. To enable service robots as agile as possible to execute various service tasks, the topic of this thesis is designing fast grasping algorithms of unknown objects for service robots. To achieve this target, this thesis sets out to reach three subgoals and one general goal:

Subgoal 1: Improve the time efficiency for unknown object grasping

The simplest and most straightforward way to accelerate grasp searching process is to reduce computational load of grasping algorithms. Computational load can be decreased by using less data for computation and configuring less grasp candidates. Partial point cloud comparing with full 3D model is naturally better at cutting down data used for grasp computation. Configuration of grasp candidates in 3D space means many possibilities. Features (principal axis, boundary and concavity, normals) of unknown objects are excellent clues to decrease useless grasp candidates. Using less data for grasp computing and configuring less grasp candidates, searching process of unknown object grasping can definitely be accelerated, which is validated by fast computing (Chapter 3 to Chapter 6) of suitable grasps for the tested household objects. Using principal axis and boundary to separately guide grasp configurations, the proposed grasping algorithms in Chapter 3 and Chapter 4 can work out suitable grasp on a single-view partial point cloud within 1s. Comparing with dominant fast grasping algorithms [84, 113, 114], the grasping algorithms in Chapter 3 and Chapter 4 are 153% faster than [84], 205% faster than [113] and 275% faster than [114]. In Chapter 5, grasp computation on the concavity of a tested target object can be completed within 2s to output suitable grasps; and the proposed grasping approach in Chapter 6 using C-shape configuration on a single-view partial point cloud can also work out a suitable grasp within 2s for a test unknown object. The computing speed of grasping algorithms in Chapter 5 and Chapter 6 are 20% faster than [84], 42% faster than [113] and 95% faster than [114].

Subgoal 2: Enhance the grasping security of using partial point cloud

Using partial point cloud comparing with full 3D model means less data, which can undoubtedly speed grasp searching. However, utilizing partial point cloud inevitably introduces occlusions resulted in by the unseen parts of target unknown objects. Unseen parts of target unknown objects inevitably lead to fake collision-free configuration of grasp candidates, for example, grasp configurations from side direction and back direction, which may lead to grasp failure. In this thesis, we proposed two methods to deal with unseen parts when using partial point cloud. The first approach is to constrain grasp configurations on the seen part that is actually a “big” partial point cloud constructed by using two 3D cameras, which is validated in Chapter 3. The second way is to add manmade unseen parts for target unknown objects and effectiveness of this method is proven in Chapter 6. Using above two approaches, the proposed grasping methods in Chapter 3 and Chapter 6 can reach 100% percent of success to avoid unexpected contact with the target object to ensure the security of using partial point cloud.

Subgoal 3: Ensure the grasp stability when friction coefficient is unknown

To obtain stable grasps for unknown objects, force closure is a widely used criterion. However, for unknown objects, it is hard to know friction coefficient that is necessary to compute force closure grasp. Existing literature employ approximate methods to achieve rough force closure grasps when friction coefficient is unknwon. For example, two parallel straight lines [110] or two parallel planes [95] are utilized to work as an approximate force closure grasp. Inspired by the above two approximate methods, we propose to fit the two grasp sides into two straight lines. The angle between the two fit lines is used to evaluate the force closure quality of a grasp. Larger angle infers higher possibility that the target object will be squeezed out when the robot tries to perform the grasping action. Therefore, the grasp candidate with minimal angle is chosen as final grasp to ensure the grasp stability during grasp execution. Using such a way, the optimized grasp with the best approximate force closure is selected to work as final grasp, which is validated by Chapter 3 to Chapter 6. The final grasps output from Chapter 3 to Chapter 6 are with locally best force balance. In Chapter 3, force balance angle of final grasp for the tested household objects are within 0.326 radian, i.e. 18.6°, which can ensure that the two grasp sides are almost two parallel lines to thus asure the stability of final grasp. Chapter 4 to Chapter 6 obtained the similar force stability results of the final grasps of the tested objects as Chapter 3.

General goal: Create a general fast grasping algorithm

The general goal of this thesis is to create a fast and general grasping algorithm that can utilize incomplete data to achieve 100% success-rate grasp for any object shape within 2 seconds.

As stated in Chapter 6, [84, 113, 114] are the dominant fastest grasping algorithms that using incomplete data to achieve fast grasps for unknown objects. Among these three fast grasping algorithms, [84] is the fastest one using a partial point cloud as input, the average time of grasp searching for a target object is 2.352s on a computer with an Intel Xeon E3 CPU 3.30 GHz. [113] is the second fastest algorithm using a incomplete point cloud as input, the average computing time for grasping a target object is 2.7s on an Intel i7 3.5 GHz system (four CPU cores) with 16 GB of system memory. [114] can also obtain fast grasp on a partial point cloud of the target object, the average computing time is 4.22s on computer (Intel ® Core ™ i7-4710MQ CPU 2.50 GHz, 16.0GB RAM). Except for the computing speed, another important factor that the grasping community concerns a lot is generality of the designed grasping algorithm. The above fast grasping algorithms are specially designed for parallel grippers because parallel grippers are easy to control. Overall, our goal in this thesis is to design a faster and more general grasping approach for unknown objects than [84, 113, 114].

To design a fast and general grasping algorithm, differing from Chapter 3, 4 and 5 that utilize object features of principal axis, boundary and concavity to achieve fast grasping of unknown objects, a grasping algorithm independent on object features is designed in Chapter 6, in which the geometric shapes of grippers are simplified into C-shapes. The C-shape is then used to match the singe-view partial point cloud of the target object along its normals. After that, collision check, local geometry analysis and force balance computation are used to evaluate the grasp candidate to select out the final grasp. Experiments of nine common household objects on a personal computer (2 cores, Intel i5 2.9 GHz, 8GB RAM) are used to evaluate the designed grasping algorithm. Even though the single-view partial point cloud of the object has large number of points, our algorithm can quickly work out a suitable grasp within 2 seconds. Comparing with [84, 113, 114], our grasping algorithm shows much improvement on time efficacy. Considering the computing time and computing ability of CPU, the proposed grasping algorithm in Chapter 6 is 20% faster than [84], 42% faster than [113] and 95% faster than [114]. As to the success rate of grasping, the proposed grasping algorithm in Chapter 6 showed 100% success rate for the tested objects on the desk, all the tested objects can be successfully grasped without any unexpected falling out of hands. However, after the tested objects are successfully grasped, we did not test the grasping performance when the tested objects are lifted up and stay in the air. Meanwhile, the proposed C-shape grasping approach is not only applicable for under-actuated grippers used in experiments, but also applicable for parallel grippers and dexterous hands, both of which are easily formed into C-shapes. In such a way, our designed fast grasping algorithm can be widely used by different types of robot hands.

Overall, this thesis solved the problem of unknown object grasping step by step. Chapter 3, Chapter 4 and Chapter 5 proposed fast grasping algorithms that rely on object features of principal axis, boundary and concavity. Using object features can definitely accelerate grasp searching, but at the cost of generality of the designed algorithms. To achieve a fast and general grasping algorithm, Chapter 6 started to analyze from the geometric feature of grippers. A C-shape fast grasping algorithm is designed to achieve a fast, secure, stable and general grasping algorithm for unknown objects.

7.2 Discussions

A. Discussions about machine learning based grasping approaches

One of the most frequent questions is how our work compares with machine learning based grasping approaches. Proposed grasping approaches for unknown objects in this thesis are analytical approaches. In addition to analytical grasping approaches, there is another kind of grasping approaches for unknown objects, i.e. machine learning based grasping approaches.

Machine learning approaches show effectiveness for a wide range of perception problems by enabling the perception system to learn a mapping from the feature set to various visual properties. [174] is the first literature to show the potential of using learning approaches to solve the problem of grasping from vision, in which a learning component is introduced to evaluate grasp quality. Richer features and learning methods in recent literature allow robots to grasp known objects that are partially occluded [175] or objects in an unknown pose [176] as well as totally unknown objects [177] that the robot has not seen before.

Recent highly citied literature that using machine learning to achieve grasps for unknown objects are of high scientific value. The work of [178] focuses on detecting a single grasping point from 2D partial-view data, and then heuristic methods are utilized to determine a griper pose based on the grasping point. [179] computes a number of features of grasp quality using the features of both 2D images and 3D point clouds. Using such features, a supervised learning algorithm is applied to estimate which configuration of fingers reflect good grasp. [180] and [181] utilize the shape based approximations as bases for learning algorithms to directly compute an approaching vector for grasping. [182] simplifies grasp detection as a ranking problem of sets of contact points in image space. Supervised learning of Support Vector Machines is used to learn grasping concepts that suit well for novel objects. [183] represents a grasp as a 2D oriented rectangle in image space. SVM is utilized to detect more exactly the gripper pose which should be used for grasping. Due to the availability of inexpensive depth sensors, RGB-D data has been a significant research focus in recent years for various applications. [184] uses an RGB-D view of a scene containing objects to detect robotic grasps. To make detection fast and robust, two deep learning networks are used. The first network has fewer features, which can quickly and efficiently prune out unlikely grasp candidates. The second network with more features is then used to select suitable grasp for final execution. Most recent learning based grasping approach [113] detects grasp poses on a single-view partial point cloud of novel objects presented in a cluttered scenario. The geometric shape of a parallel gripper is used to generate labeled datasets to train the machine learning algorithm. This approach can quickly work out suitable grasp on a single-view partial point cloud.

Overall, grasping approaches based on machine learning shows great potential to solve the problem of unknown object grasping. According to the types of inputs, grasping algorithms based on machine learning can be divided into three types, i.e., 2D image based, RGB-D data and point cloud. Grasping approaches use machine learning on 2D image is not very reliable, because it is easily affected by the brightness of light. Due to the availability of inexpensive depth sensors, RGB-D data [184] and partial point cloud [113] have been significant research trend in recent years for machine learning based grasping approaches of unknown objects. However, [184] did not mention the hardware they use and the time efficiency of the grasp algorithm. [113] is a benchmark of fast grasping using machine learning. In which the average time spent for grasp detection only takes about 2.7 seconds on an Intel i7 3.5 GHz system (four cores) with 16GB of system memory.

Advantages of using machine learning to solve the problem of unknown object grasping are:

(a) Adaptive: comparing with analytical methods directly applied on 2D images or 3D point clouds, machine learning based grasping approaches are more adaptive, because machine learning is actually a classification process using the similarity between the trained data and query data. Comparing with analytical approaches proposed in this thesis, such similarity leaves the grasping methods based machine learning more adaption. For example, Chapter 3 utilizes an analytical way to grasp the target unknown object along the principal axis. If the principal axis cannot be computed correctly because of occasions resulted in by using a single-view partial point cloud, the grasping algorithm may lead to grasp failure.

(b) Self-learning. Robots can acquire new skills when using by combine different learning methods to work together. AlphaGo is the first computer program that uses machine learning to defeat a professional human Go player, which shed light on using machine learning to solve the problem of unknown object grasping. In the future, robots may perform better grasps for unknown objects than humans. Using analytical grasping approaches is impossible to achieve such a goal because no new skills can be acquired by robots.

Disadvantages of using machine learning on unknown object grasping are:

(a) Hard to design good features. The aforementioned approaches based on machine learning require a significant degree of hand-engineering in the form of designing good input features, which is very trivial. For example, 4032 hand-crafted features are used for the learning network in [184]. More importantly, designed features valid for one kind of objects or one type of robot hand may be invalid for other objects or other robot hands. Using the proposed analytic approaches in thesis is much easier than deigning trivial features.

(b) Difficult to be general at present. For the time being, it is hard for a specific machine learning to generalize its skill to different situations to determine suitable grasps for various unknown objects. For example, the learned skills from one grasping algorithm work for bottles, but not for cups. The robot needs to relearn by trial-and-error to autonomously acquire new skills, which can adapt the learning algorithms to suit various grasp occasions. To enable robots acquire versatile grasping skills, a learning architecture that can combine different learning approaches is necessary. Such architecture can enable different learning methods to work together. However, there is not yet such a learning architecture, and it is very difficult to build such architecture. The analytical general grasping approaches in Chapter 6 is useful before the future versatile grasping approaches based on self-learning appears.

To sum up, according to current literature, it is hard to judge which is better between machine learning based grasping approaches and analytical grasping methods. But in the future, unknown object grasping based on machine learning is the scientific direction of the community of grasping field. Future grasping approaches based on machine learning may have better grasping ability than humans if the aforementioned architecture that combines different learning approaches becomes reality.

B. Discussions about different approaches to achieve force closure grasps

How to choose suitable approach to achieve force closure grasp is one of the frequent questions for users of grasping algorithms. According to existing literature, there are three types of methods that can achieve force closure grasp, i.e. analytical simulation approaches, tactile sensor based approaches and approximate force closure approaches. The proposed approaches of achieving force closure can be included in approximate force closure approaches.

(a) Analytical simulation approaches: When the friction coefficient of the target object is known and the full 3D model of the target object is available, GraspIt! and Opengrasp are excellent analytical tools to simulate suitable grasps for unknown objects. Through analytically simulating contact force and torque that the robot hand imposes on the target object, the robot can obtain final grasp with superb force closure. However, the time efficiency of using such simulation tools is not apparent. Time spent for grasp simulation varies from several minutes to several hours depending on the complexity of target objects and the complexity of robot hands.

(b) Tactile sensor based approaches: When tactile sensors are available, continuous force and torque feedback from tactile sensors can help to real-timely adjust the action of fingers and the force imposed on the target object. In such a way, favrable force closure grasps can be obtained. However, such continuous adjustment under the help of unceasing tactile feedback is significantly time-consuming. Meanwhile, financial cost for tactile sensors are pretty higher than using analytical simulation tools.

(c) Approximate force closure approaches: when the above two methods are not applicable, for example, using partial point cloud and a parallel gripper without tactile sensors to achieve force closure grasps, approximate force closure approaches are the best tool in such a situation. Approximate force closure approaches including using parallel lines, employing parallel planes and untilizing the angle based approximate force closure (proposed in this thesis) are valuable to try. The advantage of using such kind of approximate force closure approaches is that they are usually with high time efficiency. However, using approximate force closure approaches may introcude grasp uncertainty to thus lead to grasp failure.

In summary, analytical simulation approaches, tactile sensor based approaches and approximate force closure approaches have their own merits and drawbacks. Users need to make choice according to their grasp occasions. If the friction coefficient and the full 3D model are available, analytical simulation approaches are better, however, at the cost of time efficiency. If the tactile sensors are availbe, tactile sensor based approaches are better, however, also at the cost of time efficiency. If the above two approaches are not applicable, approximate force closure approaches can be utilized, however, at the risk of grasp uncertainty.

C. Discussions about different approaches to deal with occlusions.

When using partial point cloud to accelerate searching process of unknown object grasping, there are two commonly used methods to deal with occlusions, i.e. tactile sensor based exploration and active vision based exploration. These two methods are absolutely safe ways to detect the unseen parts of the target objects, but at the cost of time efficiency. In this thesis, we propose two approaches to solve the problem of occlusions, i.e. virtual exploration in Chapter 3 and manmade unseen part in Chapter 6. Here, we will discuss the merits and drawbacks of different approaches to deal with occlusions.

(a) Tactile sensor based exploration. [185] utilizes feedbacks from tactile sensor arrays to explore the surface of unknown objects. A tactile sensor pad with 16×16 tactels is mounted at the end of a 7-dof Kuka lightweight robot arm. A control framework for tactile servoing is designed to drive the robot to explore the surface of unknown objects. This kind of exploration is with high scientific value, especially for occasions that vision based exploration are not applicable, for example, an object in the fridge. However, the financial cost of tactile sensors is high and the time efficiency for tactile exploration is low.

(b) Vision based exploration. In [93], a camera is installed at the end of the robot arm that will move around the target unknown object. Continuous inputs from the camera are used to explore the unseen part of the target object. Such kind of vision based explorations makes full use of the arm’s ability to move around the target object. Actually, vision based exploration confirms well with human’s action. If humans cannot see the backside of the object, humans usually move around it to explore it. However, this kind of exploration is similar as the construction of full 3D model of the object, which is time-consuming.

(c) Virtual exploration and manmade unseen part. Virtual exploration in this thesis is actually to constrain the grasp configurations on the seen part of the target unknown object. In such a way, the security of the final executable grasp is ensured. Comparing with genuine vision exploration, virtual exploration in this thesis is with less flexibility, but high time efficiency. Manmade unseen part means filling the unseen parts with artificial obstacles. In such a way, the robot cannot grasp the target object from the side direction or back direction. It is a quite safe way to deal with occlusions, but at the cost of ignoring many potential grasps.

To sum up, the above three approaches have their own advantages and drawbacks. Users need to make choice depdent on their own grasping conditions.

D. Success rate of the proposed grasping approaches

Unfortunately, we could not test the success rate of the proposed grasping approaches because of malfunction of our gripper. According to current literature, the most acceptable way to evaluate whether a grasp is successful or not is as following steps:

(a) Successfully reach the target object. The robot arm must move to the desired grasping location without hitting the target object and obstacles. The work of arm movement can be done using motion planners to do path planning for robots. In order to enable users quickly select suitable motion planners for their grasping purpose, a profound comparison of all available motion planners in MoveIt! is conducted using two benchmarks in Chapter 6.

(b) Successfully grasp the target object. After the robot hand arrives at the grasping point, robot hand will close to grasp the target object, which is the most important part to see the grasp is successful or not. For example, during the execution of the grasp in Figure 3.7, forces that fingers impose on the target object will lead to movement of the object. If such movement is magnitude, the target object may fall out of the robot hand, which will lead to grasp failure.

(c) Successfully pick the target object up and hold it for a period. For example, a grasp is judged to be successful in [183] if the robot can lift a tested object up and hold it for 30 seconds.

Our proposed grasping approaches of unknown objects present excellent performance for reaching and grasping target object. Unfortunately, we did not successfully lift the tested objects and hold them for a period due to the problem of our gripper. The maximum grasping force of our gripper is too weak to lift the tested objects. That is why we did not include the success rate of our grasping algorithms. We strongly suggest future researchers to strictly follow the aforementioned three steps to test the success rate of their grasping algorithms. The above three steps are of significant importance to test the success rate of a grasping algorithm.

7.3 Future directions

All proposed grasping approaches in this thesis are analytical grasping approaches. As discussed in the section of 7.2, for the time being, it is hard to judge which is better between analytical grasping methods and machine learning based grasping approaches. However, in the future, we firmly believe that unknown object grasping based on machine learning is the scientific direction of the community of grasping field.

A. Short-term future direction: Integration of multiple geometric features

Existing machine learning based grasping approaches utilizing simple geometric features suffer from a limited generality. It is hard for a specific machine learning based grasping approach that utilizes a simple geometric feature to generalize its skill to different situations to determine a suitable grasp for various unknown objects. For example, machine learning based grasping approach using boundary features may not suit unknown objects with apparent concavity feature. Therefore, multiple cues of geometric features are predicted to be integrated to achieve more robust grasping approaches for unknown objects in the future.

B. Long-term future direction: Architecture for self-learning

In the future, the robot need learn by trial-and-error to autonomously acquire new skills to grasp unknown objects, which is the long-term scientific direction of machine learning based grasping. For example, the learned skill from one grasping algorithm works for cups, but not for bottles, then the robot needs to relearn to obtain new skills. Autonomous acquiring of new skills can adapt the learning algorithms to suit various grasp occasions. To enable robots acquire versatile grasping skills, a learning architecture that can combine different learning approaches is in crucial demand. Such architecture can enable different learning methods to work together to provide better grasping solutions for unknown objects.

References

[1] Population Division, Department of Economic and Social Affairs, United Nations. World Population Ageing 2015. Report, 2015.

[2] Population Division, Department of Economic and Social Affairs, United Nations. World population prospects: the 2012 revision. DVD edition, 2013.

[3] Frank Chung. The ‘crossing’ will see old people to outnumber children for the first time before 2020. Australian news, 2016.

[4] Rob Janssen, Erik van Meijl, Daniel Di Marco, René van de Molengraft and Maarten Steinbuch. Integrating planning and execution for ROS enabled service robots using hierarchical action representations. In 16th International Conference on Advanced Robotics (ICAR), Pages: 1-7, 2013.

[5] Nikolaus Vahrenkamp, Tamim Asfour and Rudiger Dillmann. Simultaneous Grasp and Motion Planning: Humanoid Robot ARMAR-III. IEEE Robotics and Automation Magazine, Volume: 19, Issue: 2, Pages: 43-57, 2012.

[6] Matthias Nieuwenhuisen, David Droeschel, Dirk Holz, Jörg Stückler, Alexander Berner, Jun Li, Reinhard Klein and Sven Behnke. Mobile bin picking with an anthropomorphic service robot. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 2327-2334, 2013.

[7] V. Ng-Thow-Hing, P. Luo, and S. Okita. Synchronized gesture and speech production for humanoid robots. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Pages: 4617-4624, 2010.

[8] M. Webster, C. Dixon, M. Fisher, M. Salem, J. Saunders, K. L. Koay, K. Dautenhahn, and J. Saez-Pons. Toward reliable autonomous robotic assistants through formal verification: A case study. IEEE Trans. Human-Machine Systems, Volume: 46, Issue: 2, Pages: 186-196, 2016.

[9] Arkadiusz Gardecki and Michal Podpora. Experience from the operation of the Pepper humanoid robots. In 2017 Progress in Applied Electrical Engineering (PAEE), Pages: 1-6, 2017.

[10] https://www.eu-robotics.net/cms/upload/topic_groups/SRA2020_SPARC.pdf

[11] Mehmet Dogar, Ross A. Knepper, Andrew Spielberg, Changhyun Choi, Henrik I. Christensen, Daniela Rus. Towards coordinated precision assembly with robot teams. Experimental Robotics, volume 109, Pages: 655-669, 2015

[12] S. Jorg, J. Langwald, J. Stelter, G. Hirzinger, C. Natale. Flexible robot-assembly

using a multi-sensory approach. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 3687-3694, 2000.

[13] Luka Peternel, Tadej Petrič, Jan Babič. Human-in-the-loop approach for teaching robot assembly tasks using impedance control interface. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 1497-1502, 2015.

[14] J. K. Antonio. Optimal trajectory planning for spray coating. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 2570-2577, 1994.

[15] S. Duncan, P. Jones, P. Wellstead. Robot path planning for spray coating: a frequency domain approach. In Proceedings of the 2004 American Control Conference, Pages: 4643-4648, 2004.

[16] Z. M. Bi, Sherman Y. T. Lang. A Framework for CAD- and Sensor-Based Robotic Coating Automation. In IEEE Transactions on Industrial Informatics, Volume: 3, Issue: 1, Pages: 84-91, 2007.

[17] Terrence Fong, Charles Thorpe, Charles Baur. Collaboration, Dialogue, Human-Robot Interaction. Robotics Research, Springer Tracts in Advanced Robotics, vol 6, Pages: 255-266, 2003.

[18] Michael A. Goodrich, Alan C. Schultz. Human-robot interaction: a survey. Foundations and Trends in Human-Computer Interaction archive, Volume: 1, Issue: 3, Pages: 203-275, 2007.

[19] Oya Celiktutan, Efstratios Skordos, Hatice Gunes. Multimodal Human-Human-Robot Interactions (MHHRI) Dataset for Studying Personality and Engagement. IEEE Transactions on Affective Computing, Volume: PP, Issue: 99, Pages: 1-14, 2017.

[20] Benjamin Kuipers, Yung-Tai Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. In Robotics and Autonomous Systems, Volume: 8, Issues: 1–2, Pages: 47-63, 1991.

[21] R. Zlot, A. Stentz, M. B. Dias, S. Thayer. Multi-robot exploration controlled by a market economy. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 3016-3023, 2002.

[22] Stefan Oßwald, Maren Bennewitz, Wolfram Burgard, Cyrill Stachniss. Speeding-Up Robot Exploration by Exploiting Background Information. IEEE Robotics and Automation Letters, Volume: 1, Issue: 2, Pages: 716-723, 2016.

[23] Yasuhisa Hirata, Zhidong Wang, Kenta Fukaya, Kazuhiro Kosuge. Transporting an Object by a Passive Mobile Robot with Servo Brakes in Cooperation with a Human. Advanced Robotics, Volume: 23, Issue: 4, Pages: 387-404, 2009.

[24] Jianing Chen, Melvin Gauci, Wei Li, Andreas Kolling, Roderich Groß.

Occlusion-Based Cooperative Transport with a Swarm of Miniature Mobile Robots. IEEE Transactions on Robotics. Volume: 31, Issue: 2, Pages: 307-321, 2015.

[25] Clemens Mühlbacher, Stephan Gspandl, Michael Reip, Gerald Steinbauer. Improving dependability of industrial transport robots using model-based techniques. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 3133-3140, 2016.

[26] Jaka Katrasnik, Franjo Pernus, Bostjan Likar. A Survey of Mobile Robots for Distribution Power Line Inspection. IEEE Transactions on Power Delivery, Volume: 25, Issue: 1, Pages: 485-493, 2010.

[27] Seung-Nam Yu, Jae-Ho Jang, Chang-Soo Han. Auto inspection system using a mobile robot for detecting concrete cracks in a tunnel. Automation in Construction, Volume: 16, Issue: 3, Pages: 255-261, 2007.

[28] Tatsuya Kishi, Megumi Ikeuchi, Taro Nakamura. Development of a peristaltic crawling inspection robot for 1-inch gas pipes with continuous elbows. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Pages: 3297-3302, 2013.

[29] K.B. Shimoga. Robot Grasp Synthesis Algorithms: A Survey. International Journal of Robotics Research, Volume: 15, Issue: 3, pages: 230-266, 1996.

[30] Qujiang Lei, Jonathan Meijer, Martijn Wisse. A survey of unknown object grasping and our fast grasping algorithm-C shape grasping. In 3rd International Conference on Control, Automation and Robotics (ICCAR), Pages: 150-157, 2017.

[31] Dan Song, Carl Henrik Ek, Kai Huebner, Danica Kragic. Task-Based Robot Grasp Planning Using Probabilistic Inference. IEEE Transactions on Robotics, Volume: 31, Issue: 3, Pages: 546-561, 2015.

[32] Christian Smith, Yiannis Karayiannidis, Lazaros Nalpantidis, Xavi Gratal, Peng Qi, Dimos V. Dimarogonas, Danica Kragic. Dual arm manipulation—A survey. In Robotics and Autonomous Systems, Volume: 60, Issue: 10, Pages: 1340-1353, 2012.

[33] Mustafa Ersen, Erhan Oztop, Sanem Sariel. Cognition-Enabled Robot Manipulation in Human Environments: Requirements, Recent Work, and Open Problems. IEEE Robotics & Automation Magazine, Volume: 24, Issue: 3, Pages: 108-122, 2017.

[34] Cristina Garcia Cifuentes, Jan Issac, Manuel Wüthrich, Stefan Schaal, Jeannette Bohg. Probabilistic Articulated Real-Time Tracking for Robot Manipulation. IEEE Robotics and Automation Letters, Volume: 2, Issue: 2, Pages: 577-584, 2017.

[35] Jeannette Bohg, Antonio Morales, Tamim Asfour, and Danica Kragic. Data-Driven Grasp Synthesis: A Survey. IEEE Transactions on Robotics, Volume: 30, Issue: 2,

Pages: 289-309, 2014.

[36] C. Borst, M. Fischer, and G. Hirzinger. Grasping the dice by dicing the grasp. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Pages: 3692-3697, 2003.

[37] Antonio Morales, Tamim Asfour, Pedram Azad, Steffen Knoop and Rudiger Dillmann. Integrated grasp planning and visual object localization for a humanoid robot with five-fingered hands. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Pages: 5663-5668, 2006.

[38] S. Ekvall and D. Kragic. Learning and evaluation of the approach vector for automatic grasp generation and planning. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 4715-4720, 2007.

[39] J. Tegin, S. Ekvall, D. Kragic, B. Iliev, and J.Wikander. Demonstration based learning and control for automatic grasping. Intelligent Service Robotics, Volume: 2, Issue: 1, Pages: 23-30, 2008.

[40] M. Ciocarlie and P. Allen. Hand posture subspaces for dexterous robotic grasping. International Journal of Robotics Research, Volume: 28, Issue: 7, Pages: 851-867, 2009.

[41] O.B. Kroemer, R. Detry, J. Piater, J. Peters. Combining active learning and reactive control for robot grasping. Robotics and Autonomous Systems, Volume 58, Issue 9, Pages 1105-1116, 2010.

[42] R. Diankov. Automated construction of robotic manipulation programs. Ph.D. dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 2010.

[43] P. Brook, M. Ciocarlie, and K. Hsiao. Collaborative grasp planning with multiple object representations. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 2851-2858, 2011.

[44] M. A. Roa, M. J. Argus, D. Leidner, C. Borst, and G. Hirzinger. Power grasp planning for anthropomorphic robot hands. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 563-569, 2012.

[45] R. Balasubramanian, L. Xu, P. D. Brook, J. R. Smith, and Y. Matsuoka. Physical human interactive guidance: Identifying grasping principles from human-planned grasps. IEEE Transactions on Robotics, Volume: 28, Issue: 4, Pages: 899-910, 2012.

[46] C. Papazov, S. Haddadin, S. Parusel, K. Krieger, and D. Burschka. Rigid 3-D geometrymatching for grasping of known objects in cluttered scenes. International Journal of Robotics Research, Volume: 31, Issue: 4, Pages: 538-553, 2012.

[47] Stefan Escaida Navarro, David Weiss, Denis Stogl, Dimitar Milev and Bjoern Hein.

Tracking and Grasping of Known and Unknown Objects from a Conveyor Belt. In 41st International Symposium on Robotics (ISR), Pages: 1-8, 2014.

[48] Antonio Morales, E. Chinellato, A. Fagg, and A. del Pobil. Using experience for assessing grasp reliability. International Journal of Humanoid Robotics, Volume: 1, Issue: 4, Pages: 671-691, 2004.

[49] Y. Li and N. Pollard. A Shape Matching Algorithm for synthesizing humanlike enveloping grasps. In IEEE-RAS International Conference on Humanoid Robots, Pages: 442-449, 2005.

[50] N. Curtis and J. Xiao. Efficient and effective grasping of novel objects through learning and adapting a knowledge base. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Pages: 2252-2257, 2008.

[51] N. Bergstrom, J. Bohg and D. Kragic. Integration of visual cues for robotic grasping. In International Conference on Computer Vision Systems (ICVS), Pages: 245-254, 2009.

[52] J. Bohg and D. Kragic. Learning grasping points with shape context. Robotics and Autonomous Systems, Volume: 58, Issue: 4, Pages: 362-377, 2010.

[53] C. Goldfeder and P. Allen. Data-driven grasping. Autonomous Robots, Volume: 31, Issue: 1, Pages: 1-20, 2011.

[54] R. Detry, C. H. Ek, M. Madry, J. Piater, and D. Kragic. Generalizing grasps across partly similar objects. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 3791-3797, 2012.

[55] H. Dang and P. K. Allen. Semantic grasping: Planning robotic grasps functionally suitable for an object manipulation task. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Pages: 1311-1317, 2012.

[56] Alexander Herzog, P. Pastor, M. Kalakrishnan, L. Righetti, T. Asfour, and S. Schaal. Template-based learning of grasp selection. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 2379-2384, 2012.

[57] R. Detry, C. H. Ek, M. Madry, J. Piater, and D. Kragic. Learning a dictionary of prototypical grasp-predicting parts from grasping experience. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 601-608, 2013.

[58] Chunfang Liu, Wenliang Li, Fuchun Sun and Jianwei Zhang. Grasp planning by human experience on a variety of objects with complex geometry. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Pages: 511-517, 2015.

[59] Nima Shafii, S. Hamidreza Kasaei and Luís Seabra Lopes. Learning to grasp familiar objects using object view recognition and template matching. In IEEE/RSJ

International Conference on Intelligent Robots and Systems (IROS), Pages: 2895-2900, 2016.

[60] E. Rounis, Z. Zhang, G. Pizzamiglio, M. Duta, G. Humphreys. Factors influencing planning of a familiar grasp to an object: what it is to pick a cup. Experimental Brain Research, Volume: 235, Issue: 4, Pages: 1281-1296, 2017.

[61] Antonio Morales, P. J. Sanz, A. P. del Pobil, and A. H. Fagg. Vision-based three-finger grasp synthesis constrained by hand geometry. Robotics and Autonomous Systems, Volume: 54, Issue: 6, Pages: 496-512, 2006.

[62] G. M. Bone, A. Lambert, and M. Edwards. Automated modelling and robotic grasping of unknown three-dimensional objects. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 292-298, 2008.

[63] D. Kraft, N. Pugeault, E. Baseski, M. Popovic, D. Kragic, S. Kalkan, F. Wörgötter, and N. Krüger. Birth of the object: Detection of objectness and extraction of object shape through object action complexes. International Journal of Humanoid Robotics, Volume: 5, Issue: 2, Pages: 247-265, 2009.

[64] Alexis Maldonado, U. Klank and M. Beetz. Robotic grasping of unmodeled objects using time-of-flight range data and finger torque information. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Pages: 2586-2591, 2010.

[65] Mila Popović, G. Kootstra, J. A. Jørgensen, D. Kragic, and N. Krüger. Grasping unknown objects using an early cognitive vision system for general scene understanding. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Pages: 987-994, 2011.

[66] E. Klingbeil, D. Rao, B. Carpenter, V. Ganapathi, A. Y. Ng and O. Khatib. Grasping with application to an autonomous checkout robot. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 2837-2844, 2011.

[67] B. Kehoe, D. Berenson, and K. Goldberg. Toward cloud-based grasping with uncertainty in shape: Estimating lower bounds on achieving force closure with zero-slip push grasps. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 576-583, 2012.

[68] V. Lippiello, F. Ruggiero, B. Siciliano and L. Villani. Visual grasp planning for unknown objects using a multifingered robotic hand. IEEE/ASME Transactions on Mechatronics, Volume: 18, Issue: 3, Pages: 1050-1059, 2013.

[69] Jean-Philippe Saut, Serena Ivaldi, Anis Sahbani and Philippe Bidaud. Grasping objects localized from uncertain point cloud data. Robotics and Autonomous Systems, Volume: 62, Issue: 12, Pages: 1742-1754, 2014.

[70] Hiroyuki Masuta, Hun-Ok Lim, Tatsuo Motoyoshi, Ken'ichi Koyanagi and Toru Oshima. Invariant Perception for Grasping an Unknown Object Using 3D Depth Sensor. In IEEE Symposium Series on Computational Intelligence, Pages: 122-129, 2015.

[71] Hiroyuki Masuta, Soichiro Nariki, Tatsuo Motoyoshi, Ken'ichi Koyanagi, Toru Oshima and Hun-ok Lim. Estimation of easily visible position for unknown object grasping. In IEEE Congress on Evolutionary Computation (CEC), Pages: 4866-4872, 2016.

[72] Aghil Jafari and Jee-Hwan Ryu. Independent force and position control for cooperating manipulators handling an unknown object and interacting with an unknown environment. Journal of the Franklin Institute, Volume: 353, Issue: 4, Pages: 857-875, 2016.

[73] Nikolaus Correll, Kostas E. Bekris, Dmitry Berenson, Oliver Brock, Albert Causo, Kris Hauser, Kei Okada, Alberto Rodriguez, Joseph M. Romano, Peter R. Wurman. Analysis and Observations from the First Amazon Picking Challenge. IEEE Transactions on Automation Science and Engineering, Volume: PP, Issue: 99, Pages: 1-17, 2017.

[74] Kai Huebner, Danica Kragic. Selection of robot pre-grasps using box-based shape approximation. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Pages: 1765-1770, 2008.

[75] Bone GM, Lambert A and Edwards M. Automated modeling and robotic grasping of unknown three dimensional objects. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 292-298, 2008.

[76] E. Lopez-Damian, D. Sidobre and R. Alami. A grasp planner based on inertial properties. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 754-759, 2005.

[77] Hyun-Ki Lee, Myun-Hee Kim, Sang-Ryong Lee. 3D optimal determination of grasping points with whole geometrical modeling for unknown objects. Sensors and Actuators A: Physical, Volume: 107, Issue: 2, Pages: 146-151, 2003.

[78] K. Yamazaki, M. Tomono, and T. Tsubouchi. Picking up an Unknown Object through Autonomous Modeling and Grasp Planning by a Mobile Manipulator. Field and Service Robotics, Volume: 42, Pages: 563-571, 2008.

[79] Andrew T. Miller, S. Knoop, H. I. Christensen, and P. K. Allen. Automatic grasp planning using shape primitives. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 1824-1829, 2003.

[80] Goldfeder C, Ciocarlie M, Peretzman J, Dang H, Allen PK. Data-driven grasping with partial sensor data. In IEEE/RSJ International Conference on Intelligent

Robots and Systems (IROS), Pages: 1278–1283, 2009.

[81] Goldfeder C, Ciocarlie M, Dang H, Allen PK. The Columbia grasp database. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 1710-1716, 2009.

[82] Collet A, Berenson D, Srinivasa SS, Ferguson D. Object recognition and full pose registration from a single image for robotic manipulation. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 48-55, 2009.

[83] Ana Huamán Quispe, Benoît Milville, Marco A. Gutiérrez, Can Erdogan, Mike Stilman, Henrik Christensen and Heni Ben Amor. Exploiting symmetries and extrusions for grasping household objects. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 3702-3708, 2015.

[84] Yu-chi Lin, Shao-ting Wei, and Li-chen Fu. Grasping unknown objects using depth gradient feature with eye-in-hand RGB-D sensor. In IEEE International Conference on Automation Science and Engineering (CASE), Pages: 1258-1263, 2014.

[85] Rajesh Kanna Ala, Dong Hwan Kim, Sung Yul Shin, ChangHwan Kim, and Sung-Kee Park. A 3D grasp synthesis algorithm to grasp unknown objects based on graspable boundary and convex segments. Information Sciences, Volume: 295, Pages: 91-106, 2015.

[86] Berk Calli, Martijn Wisse and Pieter Jonker. Grasping of unknown objects via curvature maximization using active vision. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Pages: 995-1001, 2011.

[87] Jiatong Bao, Yunyi Jia, Yu Cheng, and Ning Xi. Saliency-Guided Detection of Unknown Objects in RGBD Indoor Scenes. Sensors, Volume: 15, Issue: 9, Pages: 21054-21074, 2015.

[88] R. Platt Jr., A. H. Fagg and R. A. Grupen. Nullspace composition of control laws for grasping. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Pages: 1717-1732, 2002.

[89] J. Felip and A. Morales. Robust sensor-based grasp primitive for a three-finger robot hand. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Pages: 1811-1816, 2009.

[90] K. Hsiao, P. Nangeroni, M. Huber, A. Saxena and A.Y. Ng. Reactive grasping using optical proximity sensors. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 2098-2105, 2009.

[91] M. Prats, P. Martinet, S. Lee and P.J. Sanz. Compliant physical interaction based on external vision-force control and tactile-force combination. In IEEE

International Conference on Multisensor Fusion and Integration for Intelligent Systems, Pages: 405-410, 2008.

[92] K. Hsiao, S. Chitta, M. Ciocarlie, and E. G. Jones. Contact-reactive grasping of objects with partial shape information. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Pages: 1228-1235, 2010.

[93] Berk Calli. Active Grasp Synthesis for Grasping Unknown Objects. PhD thesis, 2015.

[94] Clemens Eppner and Oliver Brock. Grasping unknown objects by exploiting shape adaptability and environmental constraints. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Pages: 4000-4006, 2013.

[95] Zhaojia Liu, Hiromasa Kamogawa and Jun Ota. Fast grasping of unknown objects through automatic determination of the required number of mobile robots. Advanced Robotics, Volume: 27, Issue: 6, Pages: 445-458, 2013.

[96] B. Bounab, D. Sidobre and A. Zaatri. Central axis approach for computing n-finger force closure grasps. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 1169–1174, 2008.

[97] Jae-Sook Cheong, Heinrich Kruger, A. Frank van der Stappen. Output-Sensitive Computation of Force-Closure Grasps of a Semi-Algebraic Object. IEEE Transactions on Automation Science and Engineering, Volume: 8, Issue: 3, Pages: 495-505, 2011.

[98] Fidel Gilart, Raúl Suárez. Determining Force-Closure Grasps Reachable by a Given Hand. IFAC Proceedings Volumes, Volume: 45, Issue: 22, Pages: 235-240, 2012.

[99] H. Kruger, E. Rimon, and A. F. van der Stappen. Local force closure. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 4176-4182, 2012.

[100] Heinrich Kruger, A. Frank van der Stappen. Independent contact regions for local force closure grasps. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 1588-1594, 2013.

[101] Noé Alvarado Tovar, Raúl Suárez. Searching force-closure optimal grasps of articulated 2D objects with links. IFAC Proceedings Volumes, Volume: 47, Issue: 3, Pages 9334-9340, 2014.

[102] Yun-Hui Liu; Miu-Ling Lam; D. Ding. A complete and efficient algorithm for searching 3-D form-closure grasps in the discrete domain. IEEE Transactions on Robotics, Volume: 20, Issue: 5, Pages: 805-816, 2004.

[103] Jae-Sook Cheong, Herman J. Haverkort and A. Frank van der Stappen. On

Computing All Immobilizing Grasps of a Simple Polygon with Few Contacts. Algorithmica, Volume: 44, Issue: 2, Pages: 117-136, 2006.

[104] Jae-Sook Cheong and A. Frank van der Stappen. Computing all form-closure grasps of a rectilinear polyhedron with seven frictionless point fingers. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Pages: 3276-3282, 2007.

[105] R. Krug, D. Dimitrov, K. Charusta and B. Iliev. Prioritized independent contact regions for form closure grasps. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Pages: 1797-1803, 2011.

[106] Elon Rimon, A. Stappen. Immobilizing 2-D Serial Chains in Form-Closure Grasps. IEEE Transactions on Robotics, Volume: 28, Issue: 1, Pages: 32-43, 2012.

[107] K. Ramakrishna, Dibakar Sen. Curvature based mobility analysis and form closure of smooth planar curves with multiple contacts. Mechanism and Machine Theory, Volume: 75, Pages: 131-149, 2014.

[108] Andrew T. Miller, Peter K. Allen. Graspit! A versatile simulator for robotic grasping. IEEE Robotics & Automation Magazine, Volume: 11, Issue: 4, Pages: 110-122, 2004.

[109] Rosen Diankov. Automated Construction of Robotic Manipulation Programs. PhD thesis, 2010.

[110] Johannes Baumgartl and Dominik Henrich. Fast Vision-based Grasp and Delivery Planning for unknown Objects. In the 7th German Conference on Robotics (ROBOTIK), pages 1-5, 2012.

[111] Augusto Gómez Eguíluz, I. Rañó, S. A. Coleman, T. M. McGinnity. Reliable object handover through tactile force sensing and effort control in the Shadow Robot hand. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 372-377, 2017.

[112] Lorenzo Jamone, Alexandre Bernardino, José Santos-Victor. Benchmarking the Grasping Capabilities of the iCub Hand With the YCB Object and Model Set. IEEE Robotics and Automation Letters, Volume: 1, Issue: 1, Pages: 288-294, 2016.

[113] Andreas ten Pas and Robert Platt. Using Geometry to Detect Grasps in 3D Point Clouds. In IEEE International Syposium on Robotics Research (ISRR), Pages: 1-16, 2015.

[114] Toshitaka Suzuki, Tetsushi Oka. Grasping of unknown objects on a planar surface using a single depth image. In IEEE International Conference on Advanced Intelligent Mechatronics (AIM), Pages: 572-577, 2016.

[115] http://www.ifr.org/service-robots/statistics/, accessed on 12th February 2017.

[116] Jeannette Bohg, Matthew Johnson-Roberson, Beatriz León, Javier Felip, Xavi Gratal, Niklas Bergstrom, Danica Kragic, and Antonio Morales. Mind the gap-robotic grasping under incomplete observation. In IEEE International

Conference on Robotics and Automation (ICRA), Pages: 686-693, 2011.

[117] C. Dune, E. Marchand, C. Collowet and C. Leroux. Active rough shape estimation of unknown objects. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Pages: 3622-3627, 2008.

[118] Corey Goldfeder, Peter K. Allen, Claire Lackner and Raphael Pelossof. Grasp planning via decomposition trees. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 4679-4684, 2007.

[119] Yun Jiang, Stephen Moseson, and Ashutosh Saxena. Efficient grasping from RGBD images: Learning using a new rectangle representation. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 3304-3311, 2011.

[120] Mario Richtsfeld and Markus Vincze. Grasping of Unknown Objects from a Table Top. Workshop on Vision in Action: Efficient strategies for cognitive agents in complex environments, 2008.

[121] Leon Bodenhagen, Dirk Kraft, Mila Popovic, Emre Ba¸seski, Peter Eggenberger Hotz, and Norbert Krüger. Learning to grasp unknown objects based on 3D edge information. In IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA), Pages: 421-428, 2009.

[122] Alexis Maldonado, Ulrich Klank, and Michael Beetz. Robotic grasping of unmodeled objects using time of flight range data and finger torque information. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Pages: 2586-2591, 2010.

[123] Qujiang Lei and Martijn Wisse. Fast grasping of unknown objects using force balance optimization. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Pages: 2454–2460, 2014.

[124] Qujiang Lei and Martijn Wisse. Unknown object grasping using force balance exploration on a partial point cloud. In IEEE International Conference on Advanced Intelligent Mechatronics (AIM), Pages: 7-14, 2015.

[125] Robert Haschke. Motion and Operation Planning of Robotic Systems, volume 29 of Mechanisms and Machine Science. Springer International Publishing, Cham, Zwitserland, 2015.

[126] Jonathan Meijer, Qujiang Lei, Martijn Wisse. An Empirical Study of Single-Query Motion Planning for Grasp Execution. In IEEE International Conference on Advanced Intelligent Mechatronic (AIM 2017), Pages: 59-516, 2017.

[127] S. Dragiev, M. Toussaint and M. Gienger. Uncertainty aware grasping and tactile exploration. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 113-119, 2013.

[128] Claudio Zito, Marek S Kopicki, Rustam Stolkin, Christopher Borst, Florian Schmidt, Maximo Roa, and Jeremy L Wyatt. Sequential trajectory re-planning with

tactile information gain for dexterous grasping under object-pose uncertainty. In IEEE/RSJ international conference on intelligent robots and systems (IROS), Pages: 4013-4020. 2013.

[129] N. Sommer, M. Li and A. Billard. Bimanual compliant tactile exploration for grasping unknown objects. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 6400-6407, 2014.

[130] Z. Yi, R. Calandra, F. Vegia, H. van Hoof, T. Hermans, Y. Zhang, and J. Peters. Active Tactile Object Exploration with Gaussian Processes. In IEEE/RSJ international conference on intelligent robots and systems (IROS), Pages: 4925-4930, 2016.

[131] Artem Molchanov, Oliver Kroemer, Zhe Su and Gaurav S. Sukhatme. Contact localization on grasped objects using tactile sensing. In IEEE/RSJ international conference on intelligent robots and systems (IROS), Pages: 216–222, 2016.

[132] Alvaro Collet, Siddhartha S. Srinivasa. Efficient multi-view object recognition and full pose estimation. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 2050-2055, 2010.

[133] Gregory Kahn, Peter Sujan, Sachin Patil, Shaunak Bopardikar, Julian Ryde, Ken Goldberg, Pieter Abbeel. Active exploration using trajectory optimization for robotic grasping in the presence of occlusions. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 4783-4790, 2015.

[134] Yingbai Hu, Guodong Lin, Chenguang Yang, Zhijun Li, Chun-Yi Su. Manipulation and grasping control for a hand-eye robot system using sensory-motor fusion. In IEEE International Conference on Robotics and Biomimetics (ROBIO), Pages: 351-356, 2015.

[135] T. Allen, J. Burdick and E. Rimon. Two-fingered caging of polygons via contact-space graph search. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 4183-4189, 2012.

[136] Jianhua Su, Hong Qiao, Zhicai Ou, Zhiyong Liu. Vision-Based Caging Grasps of Polyhedron-Like Workpieces With a Binary Industrial Gripper. IEEE Transactions on Automation Science and Engineering, Volume: 12, Issue: 3, Pages: 1033-1046, 2015.

[137] E. Rimon and A. Blake. Caging 2D bodies by 1-parameter two-fingered gripping systems. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 1458-1464, 1996.

[138] Elon Rimon, Andrew Blake. Caging Planar Bodies by One-Parameter by Two-Fingered Gripping Systems. International Journal of Robotics Research, Volume: 18, Issue: 3, Pages: 299-318, 1999.

[139] Attawith Sudsang and J. Ponce. On grasping and manipulating polygonal objects with disc-shaped robots in the plane. In IEEE International Conference on

Robotics and Automation (ICRA), Pages: 2740-2746, 1998.

[140] P. Pipattanasomporn and A. Sudsang. Two-finger caging of nonconvex polytopes. IEEE Transactions on Robotics, Volume: 27, Issue: 2, Pages: 324-333, 2011.

[141] M. Vahedi and A. van der Stappen. Caging polygons with two and three fingers. International Journal of Robotics Research, Volume: 27, Issue: 11-12, Pages: 1308-1324, 2008.

[142] G. Pereira, M. Campos and V. Kumar. Decentralized algorithms for multi-robot manipulation via caging. International Journal of Robotics Research, Volume: 23, Issue: 7-8, Pages: 783-795, 2004.

[143] J. Fink, M. A. Hsieh and V. Kumar. Multi-robot manipulation via caging in environments with obstacles. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 1471-1476, 2008.

[144] Yanyan Dai, Yoon-Gu Kim, Dong-Ha Lee and SukGyu Lee. Symmetric caging formation for convex polygon object transportation by multiple mobile robots. In IEEE International Conference on Advanced Intelligent Mechatronics (AIM), Pages: 595-600, 2015.

[145] J. Li and J. Xiao. Progressive generation of force-closure grasps for an n-section continuum manipulator. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 4016-4022, 2013.

[146] B. Le´on, S. Ulbrich, R. Diankov, G. Puche, M. Przybylski, A. Morales,T. Asfour, S. Moisio, J. Bohg, J. Kuffner and R. Dillmann. OpenGRASP: A toolkit for robot grasping simulation. In International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR), Pages: 109-120, 2010.

[147] Zhaojia Liu, Lounell B. Gueta and Jun Ota. Feature Extraction from Partial Shape Information for Fast Grasping of Unknown Objects. In IEEE International Conference on Robotics and Biomimetics (ROBIO), Pages: 1332-1337, 2011.

[148] Ilaria Gori, Ugo Pattacini, Vadim Tikhanoff and Giorgio Metta. Three-finger precision grasp on incomplete 3D point clouds. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 5366-5373, 2014.

[149] R. Diankov and J. Kuffner. Openrave: A planning architecture for autonomous robotics. Robotics Institute, Carnegie Mellon University, Tech. Rep. CMU-RI-TR-08-34, 2008.

[150] Jyh-Ming Lien, Nancy M. Amato. Approximate convex decomposition of polygons. Computational Geometry, Volume: 35, Issue: 1, Pages: 100-123, 2006.

[151] H. Zimmer, M. Campen, L. Kobbelt. Efficient computation of shortest path-concavity for 3D meshes. In International Conference on Computer Vision and Pattern Recognition (CVPR), Pages: 2155-2162, 2013.

[152] Ioan A. Sucan and S. Chitta. Moveit!. http://moveit.ros.org, 2013.

[153] Ioan A. Sucan, Mark Moll, Lydia E. Kavraki. The Open Motion Planning Library.

IEEE Robotics & Automation Magazine, Volume: 19, Issue: 4, Pages: 72-82, 2012.

[154] M. Kalakrishnan, S. Chitta, E. Theodorou, Peter Pastor, and Stefan Schaal. STOMP: Stochastic trajectory optimization for motion planning. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 4569-4574, 2011.

[155] M Zucker, N Ratliff, A. Dragan, M. Pivtoraiko, M. Klingensmith, C. Dellin, J. A. Bagnell, and S. Srinivasa. CHOMP: Covariant Hamiltonian Optimization for Motion Planning. International Journal of Robotics Research, Volume: 32, Issue: 9-10, Pages: 1164-1193, 2013.

[156] M. Likhachev. http://www.ros.org/wiki/sbpl, 2010.

[157] Gildardo Sánchez, Jean-Claude Latombe. A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking. In Robotics Research, Part of the Springer Tracts in Advanced Robotics book series (STAR, volume 6), Pages: 403-417, 2003.

[158] D. Hsu, J. C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 719-2726, 1997.

[159] Ioan A. Sucan and L. E. Kavraki. Kinodynamic motion planning by interior-exterior cell exploration. Algorithmic Foundation of Robotics VIII, Part of the Springer Tracts in Advanced Robotics book series (STAR, volume 57), Pages: 449-464, 2008.

[160] R. Bohlin and L. E. Kavraki. Path planning using lazy PRM. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 521-528, 2000.

[161] S. M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical Report, 1998.

[162] J. J. K. Jr. and S. M. Lavalle. RRT-connect: An efficient approach to single-query path planning. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 995-1001, 2000.

[163] Andrew M. Ladd, R. Unversity, L. E. Kavraki and R. Unversity. Motion planning in the presence of drift, under-actuation and discrete system changes. In Robotics: Science and Systems (RSS), Pages: 233-241, 2005.

[164] B. Gipson, M. Moll, and L. E. Kavraki. Resolution Independent Density Estimation for motion planning in high dimensional spaces. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 2437-2443, 2013.

[165] L. Kavraki, P. Svestka, J. claude Latombe and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 566-580, 1996.

[166] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion

planning. International Journal of Robotics Research, Volume: 30, Issue: 7, Pages 846-894, 2011.

[167] L. Janson and M. Pavone. Fast marching trees: a fast marching sampling-based method for optimal motion planning in many dimensions. Robotics Research, Part of the Springer Tracts in Advanced Robotics book series (STAR, volume 114), Pages: 667-684, 2016.

[168] Joseph A. Starek, Javier V. Gomez, Edward Schmerling, Lucas Janson, Luis Moreno, Marco Pavone. An asymptotically-optimal sampling-based algorithm for Bi-directional motion planning. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Pages: 2072-2078, 2015

[169] Oren Salzman, Dan Halperin. Asymptotically Near-Optimal RRT for Fast, High-Quality Motion Planning. IEEE Transactions on Robotics, Volume: 32, Issue: 3, Pages: 473-483, 2016

[170] L. Jaillet, J. Corts and T. Simon. Sampling-based path planning on configuration-space costmaps. IEEE Transactions on Robotics, Volume: 26, Issue: 4, Pages: 635-646, 2010.

[171] D. Devaurs, T. Sim´eon and J. Cort´es. Enhancing the transition-based RRT to deal with complex cost spaces. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 4105-4110, 2013.

[172] Andrew Dobson, A. Krontiris and K. E. Bekris. Sparse Roadmap Spanners. Algorithmic Foundations of Robotics X, Pages: 279-296, 2013.

[173] Andrew Dobson and K. E. Bekris. Improving sparse roadmap spanners. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 4106-4111, 2013.

[174] Kamon I, Flash T and Edelman S. Learning to grasp using visual information. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 2470-2476, 1996.

[175] Jared Glover, Daniela Rus, Nicholas Roy. Probabilistic Models of Object Geometry with Application to Grasping. International Journal of Robotics Research, Volume: 28, Issue: 8, Pages: 999-1019, 2009.

[176] R. Detry, E. Baseski, M. Popovic, Y. Touati, N. Kruger, O. Kroemer, J. Peters, J. Piater. Learning object-specific grasp affordance densities. In IEEE 8th International Conference on Development and Learning (ICDL), Pages: 1-7, 2009.

[177] Ashutosh Saxena, Justin Driemeyer, Justin Kearns, Andrew Y. Ng. Robotic grasping of novel objects. In proceeding of Neural Information Processing Systems (NIPS), Pages: 1-8, 2006.

[178] Ashutosh Saxena, Justin Driemeyer, Andrew Y. Ng. Robotic grasping of novel objects using vision. International Journal of Robotics Research, Volume: 27, Issue: 2, Pages: 157-173, 2008.

[179] Saxena A, Wong LLS and Ng AY, Learning grasp strategies with partial shape information. In AAAI Conference on Artificial Intelligence, Pages: 1491–1494, 2008.

[180] Staffan Ekvall, Danica Kragic. Learning and Evaluation of the Approach Vector for Automatic Grasp Generation and Planning. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 4715-4720, 2007.

[181] Kai Huebner, Danica Kragic. Selection of robot pre-grasps using box-based shape approximation. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Pages: 1765-1770, 2008.

[182] Quoc V. Le, David Kamm, Arda F. Kara, Andrew Y. Ng. Learning to grasp objects with multiple contact points. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 5062-5069, 2010.

[183] Yun Jiang, Stephen Moseson, Ashutosh Saxena. Efficient Grasping from RGBD images: Learning using a new Rectangle Representation. In IEEE International Conference on Robotics and Automation (ICRA), Pages: 3304-3311, 2011.

[184] Ian Lenz, Honglak Lee, Ashutosh Saxena. Deep learning for detecting robotic grasps. International Journal of Robotics Research, Volume: 34, Issue: 4-5, Pages: 705-724, 2015.

[185] Qiang Li, Carsten Schürmann, Robert Haschke, Helge Ritter.A control framework for tactile servoing. In Robotics: Science and Systems (RSS), 2013.

[186] http://www.ros.org/

Appendix

Comparison of motion planners in Chapter 6 is conducted by Jonathan Meijer under my supervision. In order to make it easier for readers to quickly choose suitable motion planner(s), two conference papers are included in the appendix, in which 23 motion planners available in MoveIt! are compared for the purpose of grasp execution.

[1] Jonathan Meijer, Qujiang Lei, Martijn Wisse, “An Empirical Study of Single-Query Motion Planning for Grasp Execution”, 2017 IEEE International Conference on Advanced Intelligent Mechatronic (AIM 2017), Pages: 1234-1241, Munich, Germany.

[2] Jonathan Meijer, Qujiang Lei, Martijn Wisse, “Performance Study of Single-Query Motion Planning for Grasp Execution Using Various Manipulators”, 2017 18th International Conference on Advanced Robotics (ICAR 2017), Pages: 450-457, Hong Kong.

An Empirical Study of Single-Query Motion Planning for Grasp Execution

Jonathan Meijer, Qujiang Lei, Martijn Wisse

Abstract—**This paper identifies high-performing OMPL planners, available in MoveIt!, when carrying out several grasp executions with a UR5 manipulator. Simultaneously, this paper presents useful benchmark data. The single-query performance of the planners was measured by means of solved runs, computing time and path length. Based on the results, recommendations are made for planner choice that shows high performance.**

I. Introduction

MoveIt! [1] is widely used for robot manipulation within ROS (Robot Operating System). MoveIt! comes with the Open Motion Planning Library (OMPL) [2] plugin. This lets the user easily choose state-of-the-art sampling-based motion planners from the OMPL library. Currently, 23 motion planning algorithms of OMPL are configured to use in MoveIt!. Recommendations for picking a motion planning algorithm is not given. The Planner Arena [3] is created to help users determine which planner suits a given motion planning problem. However, none of the problems resemble the use of a manipulator performing a grasp execution.

This paper aims to provide insight in choosing the right planner(s) for performing grasp executions by conducting an empirical study on the available motion planners of OMPL in MoveIt!. We choose to only investigate the single-query performance of the available sampling-based motion planners, included multi-query planners are being used as single-query planners. The performance of such planners depends on the configuration space of the robot, in particular, motion planning through narrow passages can cause issues [4]. For this paper, four grasp execution motions are defined in the MoveIt! Benchmark environment to discover the behavior of the planners. The manipulator that will be used in the benchmarking is the UR5 robot due to its universal use. To resemble the real-world grasping problem, we fitted the manipulator with a virtual camera and gripper.

The performance of the planners is measured in terms of solved runs, computing time and path length. Solved runs can be noted as a percentage of the total motion planning runs that finish correctly, barplots are used to visualize the difference. For every run, the total computing time and path length can change due to the randomization in sampling-based motion planners. Boxplots and tables are used to analyze the planners with respect to computing time and path length. Performance depends on the users need, right planners for one performance measure can be wrong planners for a different performance measure. By looking at each measure, we can discuss on the preferred planner choice.

TABLE I: Summary of available planners of OMPL in MoveIt!

Planner name	Optimizing planners	Time-invariant goal
SBL [5]		✓
EST [6]		✓
BiEST [6]		✓
ProjEST [6]		✓
KPIECE [7]		✓
BKPIECE [7]		✓
LBKPIECE [7][8]		✓
RRT [9]		✓
RRTConnect [10]		✓
PDST [11]		✓
STRIDE [12]		✓
PRM [13]		
LazyPRM [8]		
RRTstar [14]	✓	
PRMstar [13][14]	✓	
LazyPRMstar [8][14]	✓	
FMT [15]	✓	✓
BFMT [16]	✓	✓
LBTRRT [17]	✓	✓
TRRT [18]	✓	✓
BiTRRT [19]	✓	✓
SPARS [20]	✓	
SPARStwo [21]	✓	

Through a quick survey of the available planners of OMPL in MoveIt!, several observations can be made on how they handle motion problems. The comparison (Tab. I) shows that the promise of using FMT, BFMT, LBTRRT, TRRT and BiTRRT. These planners have an optimizing step and stop once an optimized path is found.

II. Background

A. Software

The open-source Robot Operation System is a suite of software libraries that help create robot applications. ROS was created to encourage collaborative robotics software development. MoveIt! serves as a framework in ROS to help with the manipulation of robotic hardware. Within MoveIt! several motion planner libraries can be added to perform motion planning for the specified robot. OMPL is a well-integrated motion planner library and is the default library for MoveIt!. It houses state-of-the-art sampling-based motion planners. OMPL itself only implements the basic primitives of sampling-based motion planning. MoveIt! configures OMPL

*The work leading to these results has received funding from the European Communitys Seventh Framework Programme (FP7/2007-2013) under grant agreement n 609206.

All authors are with the TU Delft Robotics Institute, Delft University of Technology, 2628 CD, Delft, The Netherlands. Email address of Jonathan Meijer is J.G.J.Meijer@student.tudelft.nl. Email addresses of Qujiang Lei and Martijn Wisse are q.lei, m.wisse@tudelft.nl.

and provides the back-end for OMPL to work with motion planning problems.

When executed through MoveIt!, OMPL creates a path to solve the motion planning problem. By default, OMPL tries to perform path simplification. These are routines that shorten the path. The smoothness of the path may not be affected by this simplification.

B. Overview planning algorithms

Sampling-based motion planners are proven to be probabilistically complete [13], which implicates that the probability of not finding a feasible path in an unbounded setting approaches zero. For this reason, sampling-based motion planners are widely used to find feasible paths in high-dimensional and geometrically constraint environments. Optimizing planners can refrain from potential high-cost paths and rough motions [14]. However, computational effort for finding an optimized path is increased.

Among the 23 motion planners in Tab. I, six can be considered as multi-query planning methods. A common multi-query planning method is the Probabilistic RoadMap (PRM) [13]. The planner attempts to find a path in a constructed roadmap. The construction of the roadmap is executed by sampling valid nodes (configuration states) in the configuration space. These nodes are connected to other nearby nodes by edges (path segments). In OMPL, the roadmap construction is finished when a certain time limit is reached. Afterward, a simple graph search (query) can be performed on the roadmap to find a path between the start and goal node. Because the algorithm covers the total configuration space with a roadmap, *multiple queries* can be started to find a path with different a start and goal node.

In MoveIt!, three variants of the PRM planner are available for use. The LazyPRM [8] planner initially does not check for valid states when sampling nodes for roadmap construction. Once a path has been found from between start and goal node, collision checking is performed along the nodes and edges of the roadmap. Invalid nodes and edges are removed and a new graph search is attempted. This process is repeated until a feasible path is found. PRMstar [14] is the asymptotically optimal variant of the PRM planner. It rewires nodes to other near nodes if this is beneficial to the cost towards the node. An asymptotically optimal path is found if there is a great number of nodes. LazyPRMstar [14] combines the LazyPRM and PRMstar.

In addition to the PRM planner and its variants, OMPL has two more multi-query planners, SPARS and SPARStwo. They are similar to PRMstar but adds another sparse subgraph. This subgraph is an asymptotically optimal roadmap that houses nodes which resemble multiple nodes in a dense graph. Therefore less computing memory is needed to store the asymptotically optimal roadmap. SPARStwo is different since it has an infinite iteration loop.

The remaining 17 planners in Tab. I are considered as single-query planning methods. These create a roadmap every time a new planning query has to be determined. A common single-query planner is the Rapidly Exploring Random Tree (RRT) method [9]. It grows one tree (mono-directional) from the initial configuration state in the direction of the unexplored areas of the bounded free space. This is realized by randomly sampling nodes in the free space, sampled nodes that can be are within a certain distance of tree nodes are added to the tree by edges. The process of adding nodes and edges is repeated until the tree reaches the goal node. The *goal bias* parameter in this planner specifies the probability of choosing the goal configuration as a sample rather than a random sample.

The RRTConnect method [10] is a bi-directional version of the RRT method, meaning that two trees are grown. Two processes of RRT are started, one in the start node and one in the goal node. At every iteration or edge addition, it is checked whether the trees can be connected to each other. A path that solves the motion planning problem, is found if these trees can be connected. The near-optimal variant of RRT, RRTstar [14], checks whether the new sampled node can be connected to other near nodes so that the state space is more locally refined. The RRTstar removes the connections of the new sample that are not beneficial towards the cost of the path, like PRMstar. When the number of nodes is big enough, it can result in an asymptotically optimal path from the start-to-goal node. As shown in Tab. I, the RRTstar goal is time-invariant. It keeps trying to optimize the trees by adding new nodes until specified time limit is met.

Lower Bound Tree-RRT (LBT-RRT) [17] is an asymptotically optimal planner and uses a so-called lower bound graph which is an auxiliary graph. To maintain the tree, a similar method as RRTstar is used. Transition-based RRT or TRRT [18] is a combination of the RRT method and a stochastic optimization method for global minima. It performs transition tests to accept new states to the tree. The algorithm computes an optimized path that is not tied to a time limit, unlike RRTstar. The Bi-TRRT [19] is a bi-directional version of this planner.

The EST method [6] stands for Expansive Space Trees. Other than RRT, EST tries to determine the direction of the tree by looking at neighboring nodes. The tree will grow in the direction of the less explored space. Bi-directional EST (BiEST), based on [6], grows two trees like RRTConnect. Projection EST (ProjEST), also based on [6], detects the less explored area of the configuration space by using a grid. This grid serves as a projection of the state space. Single-query Bi-directional probabilistic roadmap planner with Lazy collision checking, also called SBL, grows two trees. The trees expand in the same manner as EST. Due to its lazy collision checking it will determine if a path is valid after the two trees are connected. It deletes nodes and edges of the path that are not valid, similar to LazyPRM.

KPIECE (Kinodynamic motion Planning by Interior-Exterior Cell Exploration) [7] is a tree-based planner that uses layers of discretization to help estimate the coverage of the state space. The OMPL implementation only uses one layer. OMPL incorporates a bi-directional variant called BKPIECE and a variant which incorporates lazy collision checking, this is the LBKPIECE.

Fast Marching Tree (FMT) [15] is an asymptotically optimal planner which marches a tree forward in the cost-to-come space on a specified amount of samples. The BFMT [16] planner is a bi-directional variant of this planner.

PDST (Path-Directed Subdivision Tree) [11] represents samples as path segments instead of configuration states. It uses non-uniform subdivisions to explore the state space.

STRIDE (Search Tree with Resolution Independent Density Estimation) [12] uses a Geometric Nearneighbor Access Tree (GNAT) to sample the density of the configuration space. This information helps to guide the planner into the less explored area.

III. Problem formulation

The available planners consist of non-optimizing and optimizing planners. The problem formulation follows the work of Karaman and Frazzoli [14]. Non-optimizing planners attempt to find a feasible path in the bounded d-dimensional configuration space $C = [0,1]^d$. The free configuration space is defined by $C_{\text{free}} = cl(C \setminus C_{\text{obs}})$, in which $cl(\cdot)$ denotes the closure of a set and in which C_{obs} denotes the obstacle space. A path p is called feasible when:

$$p(0) = x_{\text{init}},\ p(1) = x_{\text{goal}} \qquad (1)$$
$$p(x) \in C_{\text{free}} \text{ for all } x \in [0,1]$$

Optimizing planners that are given a motion planning problem $(C_{\text{free}}, x_{\text{init}}, x_{\text{goal}})$ and a cost function c, find a optimized path p^* such that:

$$c(p^*) = \min\{c(p) : p \textit{ is feasible } \} \qquad (2)$$

Problem implementation. Non-optimizing planners are asked to produce feasible paths with a maximum computing time of 3s and 10s. Optimizing planners are asked to produce an optimized path within a maximum computing time of 3s and 10s. Path simplification by OMPL is turned on.

Performance metric. Solved runs, computing time and path length are used as metric in our experiments. We analyze the measures individually to provide the best performing planners in each one of the measures. Solved runs is analyzed in terms of percentage of total runs of the planner resulting in feasible paths, higher performance is considered for higher solved runs. Total computing time is measured for the time it takes for planners to produce feasible or optimized paths with path simplification, a shorter time is considered as higher performance. Moreover, planners with a small standard deviation from the average computing time and small interquartile range are considered as better performance. Path length is measured by the length of the sum of motions for a produced path. Shorter lengths are considered as higher performance. Again, planners with a small standard deviation from the average path length and small interquartile range are considered as better performance.

Parameters. In MoveIt! and OMPL parameters can be set to increase the performance of the planners. To choose them, we conducted an iterative process in which we investigated the different parameter settings for each planner by increasing and decreasing the default settings by a factor of two. Parameter tuning was conducted again in the direction performance increase was noticed until no better performance was achieved. These new parameter values were then set before conducting the benchmarks.

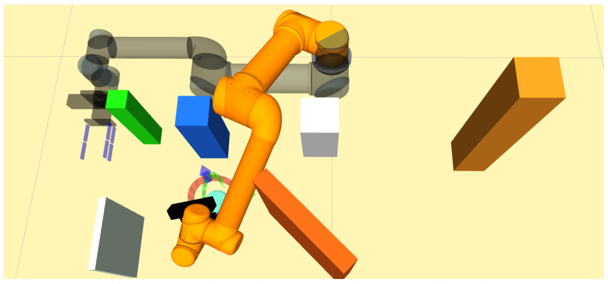


Fig. 1: Benchmark 1: Grasp between obstacles

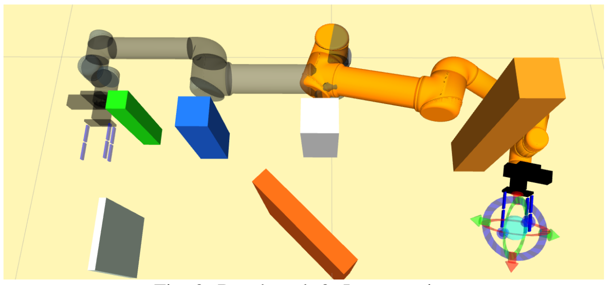


Fig. 2: Benchmark 2: Long motion

IV. Defined motion planning problems

Four grasp execution motions have been defined to measure the performance of the planners.

A. *Grasp between obstacles*

Benchmark 1 has its end-effector goal placed in a such a way that the manipulator has to move through a narrow passage, shown in Fig. 1. This benchmark uses an environment that resembles a table with obstacles. To be able to move through the narrow passage the UR5 robot needs to be in a specific configuration due to its geometry, this decreases the free configuration space near the goal configuration.

B. *Simple motion*

Benchmark 2 has its end-effector goal placed at the other side of the scene, shown in Fig. 2. It operates in the same environment as benchmark 1. The end-effector has to be displaced 1.5m in order to reach the goal. This motion planning problem can be solved by mainly actuating the shoulder lift joint. To actuate less joints, using optimizing planners could be beneficial.

C. *Place motion*

In benchmark 3, initial end-effector position is located at the end of a shelf box, shown in Fig. 3a. This benchmark uses an environment that houses a simplified shelf box and obstacles placed on a flat surface. The goal position is the

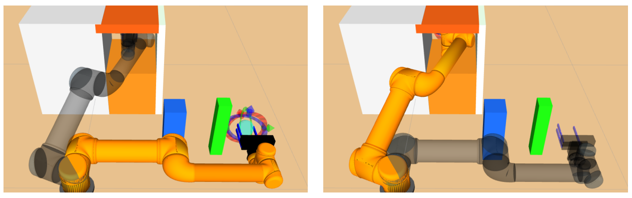

(a) Place motion (b) Pick motion

Fig. 3: Benchmark 3 and 4: a, b respectively

initial configuration used in benchmark 1 and 2 (orange state in figure). The motion problem starts in a narrow passage (transparent state in figure), the goal is situated in a less constrained space.

D. Pick motion

Benchmark 4 is attempting to resemble a picking motion from a narrow shelf box, shown in Fig. 3b. The environment is identical to benchmark 3. The goal of the problem is to be in a specific gripper orientation at the end of this shelf. The planner will have to produce a motion plan with high accuracy to reach the end of the shelf. The motion problem starts in a less constrained space (transparent state in figure), the goal is situated in a narrow passage (orange state in figure).

V. Parameter selection

Parameters can be set to improve the performance of the planners. In this section, the parameter selection is presented.

While conducting parameter selections for LBTRRT it was found that this planner is behaving unreliable in our setup. We tested all parameter combinations for this planner when conducting various motion planning which resulted in crashes. So we are unable to provide benchmark data for this particular planner.

A. Global planner parameter

There is one parameter that affects all planners. This is the distance parameter *longest_valid_segment_fraction*. The parameter is called when the planner checks for collisions between two nodes. Collision detection is not checked for the motion if the distance between the nodes is within the parameter value. In narrow passages and corners, this parameter can be critical. The parameter is set in meters and by default has a value of 0.005m. After conducting experiments with lower values, it was found that reducing this parameter did not have an immediate effect on the solved runs for the various benchmark problems.

B. Planner specific parameters

The majority of planners (20 of 23) have their own parameters. For the benchmarks, each parameter was set to values that benefit one or more performance measures, these values are noted in Tab. II.

C. Robot

The UR5 robot that will be used has two joint limit settings for each joint, π and 2π. Validating by means of simple motion planning experiments it was found that setting the joint limits to π resulted in favorable performance for all the performance measures.

VI. Results

A. Methodology

The benchmarking experiments are performed using one thread on a system with an Intel i5 2.70GHz processor and 8Gb of memory. To give reliable data on the solved runs, computing time and path length, each algorithm was run 30 times for the given motion planning problem. The algorithms were given a maximum computing time of 3s and 10s to show the effect of time on the algorithms for which the goal is not time-invariant (shown in Tab. I). The times are kept low since for most robotics applications results are required quickly. Specifically for grasping approaches that try to find grasps in a short amount of time. Planners, where path simplification was not performed by OMPL, have been marked with a * behind the planner name.

B. Plots and Tables

Results of benchmark 1 are shown in Fig. 4 and Tab. III. The motion planning problem affects planners EST, RRT, RRTstar, TRRT and SPARStwo since they were not able to solve all the runs with a percentage higher than 80% with a maximum computing time of 3s and 10s. With exception of SPARStwo, these are mono-directional planners. SBL, BiEST, KPIECE, BKPIECE and LBKPIECE compute valid paths in a computing time shorter than 0.5s. RRTConnect is the fastest planner and BiTRRT is the fastest optimizing planner. SBL has the lowest average path length with a small standard deviation. For solved runs higher than 80%, planners SBL, KPIECE, and LBKPIECE are able to plan paths of similar lengths. For optimizing planners, BiTRRT has the lowest median path length. TRRT has the lowest average path length and standard deviation. Selecting a higher limit for computing time showed shorter paths for planners RRTstar, PRMstar and LazyPRMstar, due to the optimization step. Path simplification contributes to shorter paths.

Results of benchmark 2 are shown in Fig. 5 and Tab. IV. RRTstar, TRRT and SPARStwo have lower solved runs compared to the other planner algorithms. SBL, BiEST, BKPIECE, LBKPIECE, RRTConnect and BiTRRT compute paths in under 0.1s, all being bi-directional planners. BiTRRT is the fastest optimizing planner. SBL and BiTRRT have the shortest paths. The planners that keep sampling the configuration space or optimizing the path until the maximum computing time is reached see improved performance with respect to path length. However, compared to non-optimizing planners

Results of benchmark 3 are shown in Fig. 6 and Tab. V. None of the multi-query planners are able to find feasible paths. Increased computing effort is needed to cover the total

TABLE II: Specified planner parameters

SBL	EST	BiEST	ProjEST	RRT	RRTConnect	PRM	LazyPRM	RRTstar
range: .3125	range: .625	range: 0	range: .625	range: 0	range: .3125	max n.n.: 10	range: .3125	range: 0
	goal bias: .05		goal bias: .05	goal bias: .05				goal bias: .05
								delay c.c.: 0

KPIECE	BKPIECE	LBKPIECE	STRIDE	FMT	BFMT	TRRT	BiTRRT	SPARS	SPARStwo
range: .625	range: .3125	range: .3125	range: .625	samples: 1000	samples: 1000	range: 1.25	range: 1.25	str. factor: 2.6	str. factor: 3
goal bias: .05	border frac.: .9	border frac.: .9	goal bias: .05	rad. mult.: 1.05	rad. mult.: 1.05	goal bias: .05	temp c. fact.: .2	sp. d. frac.: .25	sp. d. frac.: .25
border frac.: .9	failed e.s.f.: .5	min.v.p.frac.: .5	use proj.dist.: 0	nearest k: 1	nearest k: 1	max s. f.: 10	init temp.: 50	d. d. frac.: .001	d. d. frac.: .001
failed e.s.f.: .5	min.v.p.frac.: .5		degree: 8	cache cc: 1	balanced: 1	temp c. fact.: 2	f. threshold: 0	max fails: 1000	max fails: 5000
min.v.p.frac.: .5			max degree: 12	heuristics: 1	optimality: 0	m.temp.: 1e-10	f. n. ratio: .1		
			min degree: 6	extended fmt: 1	cache cc: 1	i.temp.: 1e-6	cost.thres.: 5e4		
			max p.p. leaf: 3		heuristics: 1	f. threshold: 0			
			est. dim.: 0		extended fmt: 1	f.NodeRatio: .1			
			min.v.p.frac.: .1			k constant: 0			

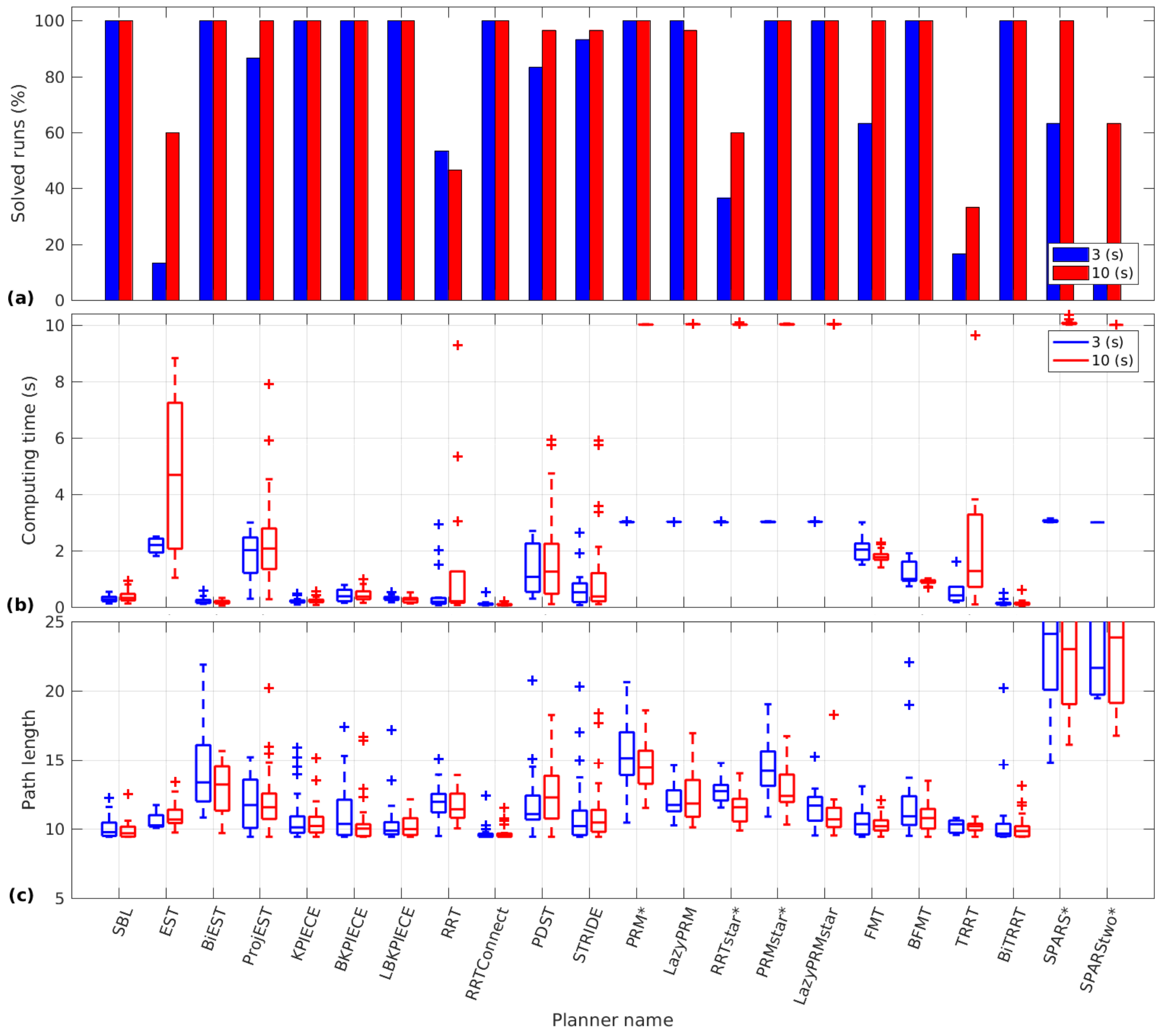


Fig. 4: Results for benchmark 1 for 3s and 10s maximum computing time. a. Solved runs (higher is better), b. Computing time (lower is better, small interquartile range is better), c. Path length (lower is better, small interquartile range is better).

free configuration space with these planners. Of the single-query planners BiEST, RRT, RRTstar and FMT are not able to reach a high level of solved runs. Indicating that these planners are not fast enough to have a proper coverage of the free configuration space. BKPIECE and RRTConnect reach 100% solved runs for 3s maximum computing time. SBL, EST, ProjEST, KPIECE, LBKPIECE, PDST, STRIDE, TRRT and BiTRRT perform better with respect to solved runs when the maximum computing time is set to 10s. RRTConnect is the fastest planner, TRRT is the fastest optimizing planner with solved runs higher than 50%. With exception of RRTConnect, the single-directional planner variants are able to plan a shorter path length compared to the bi-directional planner variant. These planners propagate a path out of a narrow passage and with the use of goal bias shorter paths can be obtained.

Results of benchmark 4 are shown in Fig. 7 and Tab. VI.

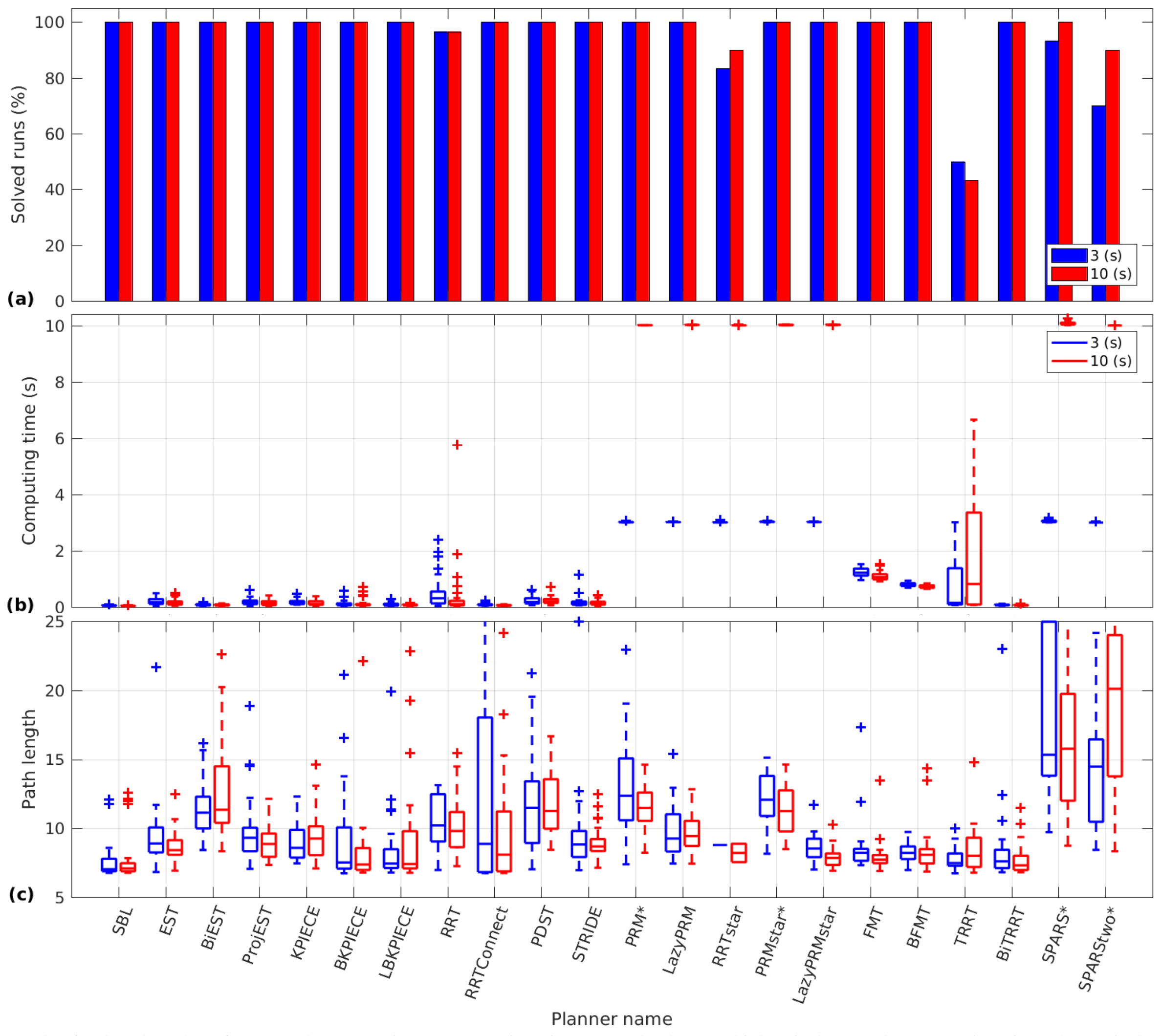


Fig. 5: Results for benchmark 2 for 3s and 10s maximum computing time. a. Solved runs (higher is better), b. Computing time (lower is better, small interquartile range is better), c. Path length (lower is better, small interquartile range is better).

5 of the 22 planners were able to compute paths with solved runs higher than 50%. These are all bi-directional planners, motion planning with these planners are also started in the goal configuration. These planners provide high solved runs for max 10s computing time. KPIECE and RRTConnect are the fastest performing planners. BiTRRT has the shortest path planning length. RRTConnect shows significant performance increase for path length with a higher maximum computing time.

C. Discussion

From the results, observations are made and discussed.

Solved runs. The planners RRTstar, TRRT and SPARStwo show consistent lower solved runs for all the benchmarks, making them less desirable to use for the grasp executions we presented. SBL, LBKPIECE and BiTRRT have high solved runs for a maximum computing time of 10s in all benchmarks. When high solved runs has to be achieved in a shorter time, BKPIECE and RRTConnect are the best choices when performing varied grasp executions.

TABLE III: Average values for benchmark 1

Planner name	Max. 3s computing time		Max. 10s computing time	
	Time (s)	Path length	Time (s)	Path length
SBL	0.29 (0.11)	10.07 (0.74)	0.37 (0.18)	9.90 (0.63)
EST	2.18 (0.31)	10.58 (0.77)	4.65 (2.55)	11.03 (1.01)
BiEST	0.21 (0.10)	14.81 (5.19)	0.18 (0.07)	13.32 (3.14)
ProjEST	1.83 (0.86)	11.82 (1.81)	2.37 (1.57)	12.13 (2.19)
KPIECE	0.20 (0.09)	10.89 (1.80)	0.22 (0.10)	10.55 (1.28)
BKPIECE	0.42 (0.21)	10.94 (1.86)	0.42 (0.21)	10.56 (1.82)
LBKPIECE	0.30 (0.08)	10.41 (1.53)	0.26 (0.11)	12.30 (7.16)
RRT	0.54 (0.84)	11.93 (1.41)	1.48 (2.71)	11.62 (1.14)
RRTConnect	0.11 (0.08)	12.52 (15.68)	0.09 (0.03)	11.89 (9.10)
PDST	1.37 (0.87)	11.96 (2.35)	1.68 (1.61)	12.37 (2.15)
STRIDE	0.59 (0.57)	11.97 (5.35)	1.12 (1.58)	11.20 (2.24)
PRM*	3.01 (0.01)	15.60 (2.26)	10.01 (0.01)	14.49 (1.65)
LazyPRM	3.02 (0.00)	12.13 (1.17)	10.02 (0.01)	12.48 (1.96)
RRTstar*	3.01 (0.01)	12.76 (0.93)	10.02 (0.02)	11.47 (1.03)
PRMstar*	3.02 (0.01)	14.43 (1.90)	10.02 (0.01)	12.99 (1.67)
LazyPRMstar	3.02 (0.00)	11.53 (1.23)	10.03 (0.01)	10.95 (1.59)
FMT	2.07 (0.45)	10.49 (0.99)	1.78 (0.21)	10.33 (0.64)
BFMT	1.17 (0.36)	11.74 (2.65)	0.89 (0.09)	10.88 (1.06)
TRRT	0.57 (0.58)	10.21 (0.52)	2.41 (2.82)	10.12 (0.45)
BiTRRT	0.13 (0.08)	15.56 (16.75)	0.13 (0.10)	11.03 (5.22)
SPARS*	3.04 (0.04)	23.63 (4.74)	10.07 (0.07)	23.87 (5.57)
SPARStwo*	3.00 (0.00)	22.98 (3.97)	10.00 (0.00)	26.34 (10.01)

Standard deviation in parentheses

TABLE IV: Average values for benchmark 2

Planner name	Max. 3s computing time		Max. 10s computing time	
	Time (s)	Path length	Time (s)	Path length
SBL	0.05 (0.01)	9.48 (7.86)	0.04 (0.01)	7.76 (1.76)
EST	0.20 (0.13)	9.53 (2.58)	0.16 (0.11)	9.38 (4.00)
BiEST	0.09 (0.04)	11.36 (1.96)	0.08 (0.03)	12.71 (3.57)
ProjEST	0.18 (0.11)	9.86 (2.51)	0.15 (0.09)	8.99 (1.32)
KPIECE	0.18 (0.09)	9.10 (1.50)	0.14 (0.08)	9.49 (1.68)
BKPIECE	0.11 (0.11)	9.71 (5.83)	0.13 (0.16)	8.17 (2.79)
LBKPIECE	0.09 (0.06)	9.33 (5.53)	0.08 (0.03)	9.23 (3.80)
RRT	0.53 (0.62)	12.03 (7.08)	0.44 (1.10)	10.15 (1.99)
RRTConnect	0.09 (0.04)	13.90 (10.39)	0.06 (0.02)	9.65 (4.05)
PDST	0.24 (0.16)	11.71 (3.56)	0.24 (0.16)	12.27 (4.00)
STRIDE	0.19 (0.21)	9.42 (3.26)	0.14 (0.09)	8.98 (1.17)
PRM*	3.02 (0.01)	12.91 (3.41)	10.01 (0.00)	11.56 (1.56)
LazyPRM	3.02 (0.00)	9.80 (1.88)	10.02 (0.00)	9.58 (1.27)
RRTstar	3.01 (0.02)	8.78 (0.00)	10.01 (0.01)	8.21 (0.94)
PRMstar*	3.03 (0.01)	12.30 (1.81)	10.02 (0.01)	11.11 (1.65)
LazyPRMstar	3.02 (0.00)	8.62 (0.98)	10.02 (0.01)	7.88 (0.72)
FMT	1.23 (0.16)	9.64 (6.44)	1.10 (0.15)	7.92 (1.15)
BFMT	0.79 (0.06)	8.24 (0.69)	0.73 (0.06)	8.35 (1.64)
TRRT	0.78 (1.00)	7.82 (0.91)	2.05 (2.43)	8.55 (2.17)
BiTRRT	0.08 (0.02)	8.39 (3.00)	0.07 (0.02)	7.75 (1.11)
SPARS*	3.05 (0.04)	19.38 (8.05)	10.07 (0.06)	16.22 (5.38)
SPARStwo*	3.00 (0.01)	14.98 (5.37)	10.00 (0.00)	20.10 (9.43)

Standard deviation in parentheses

TABLE V: Average values for benchmark 3

Planner name	Max. 3s computing time		Max. 10s computing time	
	Time (s)	Path length	Time (s)	Path length
SBL	1.54 (0.69)	13.72 (5.89)	2.06 (1.56)	15.02 (6.16)
EST	1.06 (0.64)	12.91 (2.14)	1.03 (0.79)	12.84 (1.89)
BiEST	0.11 (0.00)	16.57 (0.00)	3.51 (3.03)	16.99 (4.03)
ProjEST	0.98 (0.74)	13.26 (2.44)	1.21 (1.03)	13.02 (1.87)
KPIECE	1.15 (0.86)	13.31 (3.07)	1.00 (0.90)	13.47 (6.05)
BKPIECE	1.32 (0.78)	21.42 (31.12)	1.19 (0.87)	17.10 (7.24)
LBKPIECE	1.50 (0.88)	14.37 (3.13)	1.34 (1.28)	14.82 (4.58)
RRT*	- (-)	- (-)	3.82 (2.87)	17.20 (2.46)
RRTConnect	0.62 (0.23)	16.13 (13.07)	0.68 (0.28)	14.17 (3.69)
PDST	1.27 (0.71)	13.21 (2.75)	2.07 (1.77)	14.96 (5.60)
STRIDE	0.89 (0.59)	14.76 (4.44)	1.08 (0.82)	13.00 (1.87)
RRTstar*	3.00 (0.00)	18.06 (0.00)	10.01 (0.00)	17.64 (0.00)
FMT	2.91 (0.15)	14.99 (3.77)	6.57 (1.23)	13.47 (0.03)
TRRT	0.90 (0.66)	12.47 (3.10)	1.74 (1.92)	13.07 (4.57)
BiTRRT	1.74 (0.62)	12.55 (2.59)	2.65 (1.63)	16.12 (10.42)

Standard deviation in parentheses

TABLE VI: Average values for benchmark 4

Planner name	Max. 3s computing time		Max. 10s computing time	
	Time (s)	Path length	Time (s)	Path length
SBL	1.71 (0.73)	24.37 (51.46)	2.40 (1.67)	32.01 (83.89)
BKPIECE	1.00 (0.63)	20.15 (30.20)	1.23 (0.80)	20.14 (30.63)
LBKPIECE	1.21 (0.60)	22.68 (35.48)	1.49 (1.10)	21.29 (31.37)
RRTConnect	0.75 (0.32)	21.15 (30.21)	0.91 (0.54)	15.66 (14.27)
PRM*	- (-)	- (-)	10.01 (0.00)	18.22 (0.00)
BiTRRT	1.61 (0.65)	28.14 (35.93)	3.05 (2.02)	28.88 (62.82)

Standard deviation in parentheses

Computing time. When feasible paths need to be found quickly, a bi-directional planner is recommended similar motion planning problems to benchmark 1, 2 and 4. When a path creation starts in a narrow passage as in benchmark 3, a single-directional planner can give improved computing times, with exception of RRTConnect.

Path length. Picking an optimizing planner to find shorter path lengths can not be justified from the benchmark data. Only in benchmark 2, the TRRT and BiTRRT planners for 3s and 10s maximum computing time respectively compute shorter paths compared to non-optimizing planners. Having more computing time for time-variant goals decreases path lengths, however, not drastically. The RRTConnect shows low path length means in all benchmark problems.

Multi-query. For this paper, we only looked at single-query motion planning problems, multi-query planners can also be used as single-query planners. Though this paper fails to show the potential benefit of using the same roadmap multiple times and how this can affect computing time. Though we do notice that RRTConnect and BiTRRT are able to give valid paths in very short amounts of time that we argue the need for multi-query planners for online grasp executions.

Parameter selection. Since parameter values have to be set for a planner to operate, the aim was to achieve maximum performance of the planners. By manually conducting the iterative process explained before, a guarantee of maximum performance can not be given. We executed the iterative process to the best of our ability to achieve maximum performance.

Combined metrics. When looking at all the benchmarks, SBL, BKPIECE, LBKPIECE, RRTConnect and BiTRRT show high performance in the metrics solved runs, computing time and path length.

Optimization with a time-invariant goal. Planners FMT, BFMT, LBTRRT, TRRT and BiTRRT stop once an optimized path is found. Of these, BiTRRT is the fastest performing planner. However, compared to not non-optimizing planners the path length is not consistently shorter. More research has to be done to see whether other metrics will make the use of this planner more desirable.

Robot. The UR5 is used to compute the motion plans though we have not investigated the change in the performance measures for different types of manipulators. It needs to be determined if the results hold for other types of manipulators. For future work, multiple manipulators can be used to perform the same benchmark to find a better answer on the consistency of the planner's performance.

Path constraints. The results presented do not give any estimation on how the planners perform when implementing hard path constraints.

VII. Conclusion

This paper presented benchmark data of available OMPL planners in MoveIt! for geometrically constrained grasp executions using a 6-DOF manipulator. Planner performance was studied by means of solved runs, computing time and path length for two maximum computing time settings. From the performance analysis remarks and recommendations were made depending on the performance measure. For the defined benchmarks, the bi-directional planners SBL, LBKPIECE, RRTConnect and BiTRRT are high performing planners in terms of all the studied metrics. For future work, we would like to investigate the use of different manipulators to find consistency in planner performance when performing grasp executions.

References

[1] I. A. Sucan and S. Chitta, "MoveIt!" [Online]. Available: http://moveit.ros.org

[2] I. Sucan, M. Moll, and L. Kavraki, "Open Motion Planning Library: A Primer," 2014.

[3] M. Moll and A. S. Ioan, "Benchmarking Motion Planning Algorithms," *Ieee Robotics & Automation Magazine*, no. August, 2015.

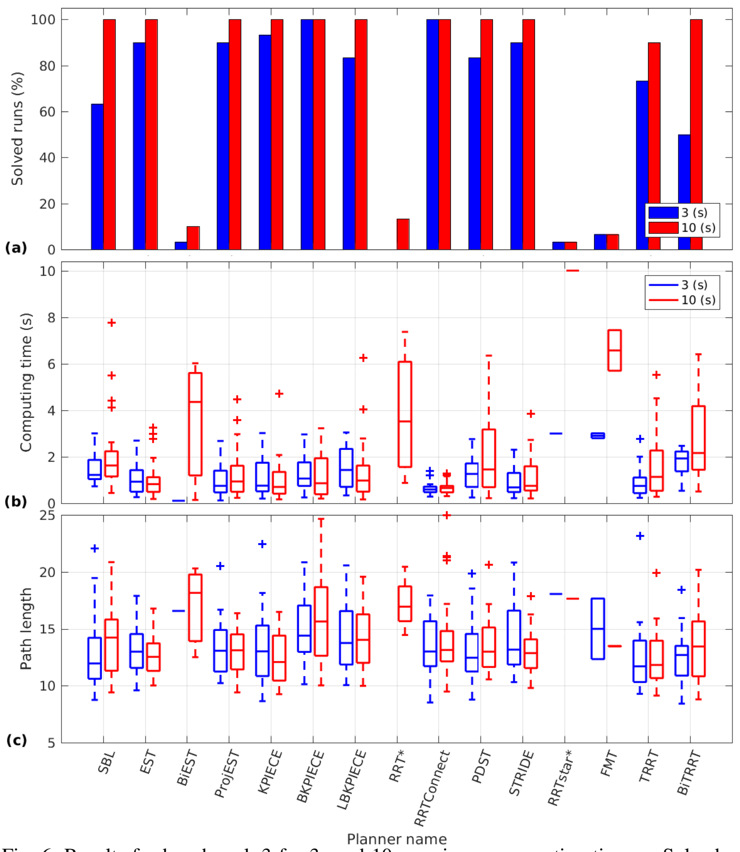


Fig. 6: Results for benchmark 3 for 3s and 10s maximum computing time. a. Solved runs (higher is better), b. Computing time (lower is better, small interquartile range is better), c. Path length (lower is better, small interquartile range is better)

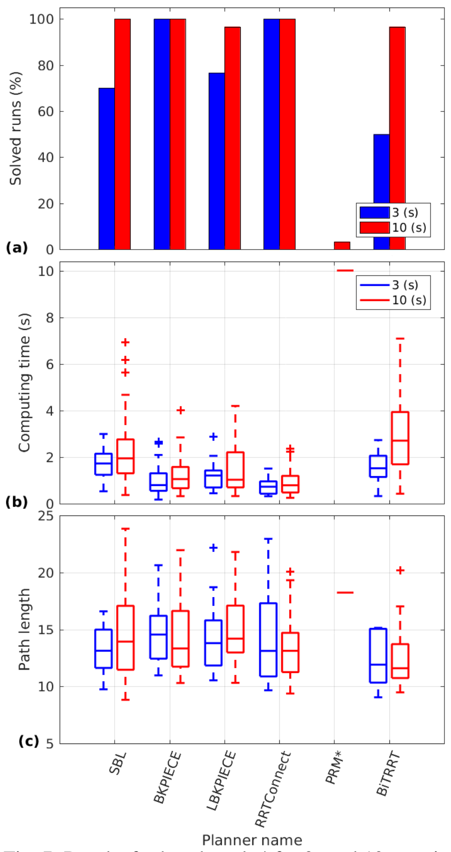


Fig. 7: Results for benchmark 4 for 3s and 10s maximum computing time. a. Solved runs (higher is better), b. Computing time (lower is better, small interquartile range is better), c. Path length (lower is better, small interquartile range is better)

[4] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin, “On finding narrow passages with probabilistic roadmap planners,” 1998.

[5] G. Sánchez and J.-C. Latombe, “A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking,” in *Robotics Research*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 403–417.

[6] D. Hsu, J. C. Latombe, and R. Motwani, “Path planning in expansive configuration spaces,” in *Proceedings of International Conference on Robotics and Automation*, vol. 3, 1997, pp. 2719–2726 vol.3.

[7] I. A. Sucan and L. E. Kavraki, “Kinodynamic motion planning by interior-exterior cell exploration,” in *IN WORKSHOP ON THE ALGORITHMIC FOUNDATIONS OF ROBOTICS*, 2008.

[8] R. Bohlin and L. E. Kavraki, “Path planning using lazy prm,” in *In IEEE Int. Conf. Robot. Autom*, 2000, pp. 521–528.

[9] S. M. Lavalle, “Rapidly-exploring random trees: A new tool for path planning,” Tech. Rep., 1998.

[10] J. J. K. Jr. and S. M. Lavalle, “Rrt-connect: An efficient approach to single-query path planning,” in *Proc. IEEE Intl Conf. on Robotics and Automation*, 2000, pp. 995–1001.

[11] A. M. Ladd, R. Unversity, L. E. Kavraki, and R. Unversity, “Motion planning in the presence of drift, underactuation and discrete system changes,” in *In Robotics: Science and Systems I*. MIT Press, 2005, pp. 233–241.

[12] B. Gipson, M. Moll, and L. E. Kavraki, “Resolution Independent Density Estimation for motion planning in high-dimensional spaces,” in *2013 IEEE International Conference on Robotics and Automation*. IEEE, may 2013, pp. 2437–2443.

[13] L. Kavraki, P. Svestka, J. claude Latombe, and M. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” in *IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION*, 1996, pp. 566–580.

[14] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *CoRR*, vol. abs/1105.1186, 2011.

[15] L. Janson and M. Pavone, “Fast marching trees: a fast marching sampling-based method for optimal motion planning in many dimensions - extended version,” *CoRR*, vol. abs/1306.3532, 2013.

[16] J. A. Starek, J. V. Gómez, E. Schmerling, L. Janson, L. Moreno, and M. Pavone, “An asymptotically-optimal sampling-based algorithm for bi-directional motion planning,” *CoRR*, vol. abs/1507.07602, 2015.

[17] O. Salzman and D. Halperin, “Asymptotically near-optimal RRT for fast, high-quality, motion planning,” *CoRR*, vol. abs/1308.0189, 2013. [Online]. Available: http://arxiv.org/abs/1308.0189

[18] L. Jaillet, J. Corts, and T. Simon, “Sampling-based path planning on configuration-space costmaps,” *IEEE Transactions on Robotics*, pp. 635–646, 2010.

[19] D. Devaurs, T. Siméon, and J. Cortés, “Enhancing the transition-based rrt to deal with complex cost spaces,” in *IEEE International Conference on Robotics and Automation, ICRA ’13*, Karlsruhe, Germany, 2013, pp. 4105–4110.

[20] A. Dobson, A. Krontiris, and K. E. Bekris, *Sparse Roadmap Spanners*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 279–296.

[21] A. Dobson and K. E. Bekris, “Improving sparse roadmap spanners,” in *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, 2013.

Performance Study of Single-Query Motion Planning for Grasp Execution Using Various Manipulators

Jonathan Meijer
TU Delft Robotics Institute
Delft University of Technology
Delft, The Netherlands
j.g.j.meijer@student.tudelft.nl

Qujiang Lei
TU Delft Robotics Institute
Delft University of Technology
Delft, The Netherlands
q.lei@tudelft.nl

Martijn Wisse
TU Delft Robotics Institute
Delft University of Technology
Delft, The Netherlands
m.wisse@tudelft.nl

Abstract— **This paper identifies high performing motion planners among three manipulators when carrying out grasp executions. Simultaneously, this paper presents useful benchmarking data. Sampling-based motion planners of OMPL available for use in MoveIt! are compared by performing several grasping-related motion planning problems. The performance of the planners is measured by means of solved runs, computing time and path length. Based on the results, recommendations are made for planner choice that shows high performance for the used manipulators.**

I. Introduction

Currently, 23 sampling-based motion planners of OMPL (Open Motion Planning Library) [1] are available for use in MoveIt! [2], a robot manipulation framework for ROS. OMPL is the default and most supported motion planning library in MoveIt!. Support for picking a planner is not provided. In the literature, no research about planner performance can be found when performing grasp executions. Moreover, performance information for 12 planners is scarce, since they have just been released (Decemeber 2016). This leaves the user to perform time-consuming benchmarks in order to find the right planner.

This paper aims to identify high performing OMPL motion planners available in MoveIt! when carrying out grasp executions. Simultaneously, it aims to present useful benchmarking data for all the 23 planners. By conducting several grasp executions for three different manipulators, we hope to find planners that consistently show high performance. We will use three manipulators that have different geometry but have similar specifications. These are Universal Robots UR5, KUKA LWR 4+ and Kinova JACO. To resemble a real-world grasping problem, the manipulators are fitted with a gripper. In this study, we only consider sampling-based motion planners of OMPL available in MoveIt!, listed in Tab. I. This includes so-called multi-query planning methods. However, only single-query performance is measured. We have designed a virtual environment that resembles a shelf in which objects can be picked or placed. Planner choice is investigated by considering geometry constraints for two motion planning problems. For the third problem we add a path constraint.

Due to randomization of sampling-based motion planners, motion planning problems have to be run multiple times to provide saturated results on the performance. The performance of the planners is measured in terms of solved runs, computing time and path length. The metric solved runs can be expressed as a percentage of the total motion planning runs that finish correctly. Barplots are used to visualize the difference. For every run, computing time and path length can change due to the randomization in sampling-based motion planners. Boxplots and tables are used to analyze the planners with respect to computing time and path length. Performance depends on the need of the user, high performance for one metric can result in low performance for an other metric. By analyzing each metric separately, we can elaborate on right planners for each metric.

Tab. I shows there are optimizing OMPL planners available in MoveIt! that have a time-invariant goal. This means they stop computing as soon as a path is found that is considered to be more optimal. However, compared to non-optimizing planners, computing effort is increased which can result in an increase in computing time. For optimizing planners is expected that they will produce shorter path lengths. This study can clarify if extra computing time of optimizing planners will considerably decrease the path length compared to non-optimizing planners.

II. Background

A. Manipulators

Universal Robots UR5 [3]: Universal Robots aims to provide easily programmable, safe and flexible industrial robots. The UR5 manipulator is designed to be lightweight, flexible and collaborative. The manipulator has 6 degrees of freedom (DOF). Joint limits are max 2π in both directions. The manipulator has a maximum payload of 5kg and a span of 850mm. A model of the UR5 fitted with a gripper is shown in Fig. 1.

KUKA LWR 4+ [4]: KUKA supplies intelligent automation solutions and is currently one of the top brands in this field. The 7-DOF LWR 4+ is a lightweight collaborative manipulator. Its furthest reach is 1178.5mm and it can lift up to 7kg. The manipulator is made for universal purpose, meaning it can be used for many applications. The upper and lower limits are ±170 degrees for four joints and ±120 degrees for the remaining joints. A model of the LWR 4+ is shown in Fig. 2.

*The work leading to these results has received funding from the European Communitys Seventh Framework Programme (FP7/2007-2013) under grant agreement n 609206.

TABLE I: SUMMARY OF AVAILABLE PLANNERS OF OMPL IN MOVEIT!

Planner name & Reference		Optimizing planners	Multi-query	Time-invariant goal
SBL	[6]			✓
EST	Based on [7]			✓
BiEST	[7]			✓
ProjEST	Based on [7]			✓
KPIECE	[8]			✓
BKPIECE	Based on [8]			✓
LBKPIECE	Based on [8][9]			✓
RRT	[10]			✓
RRTConnect	[11]			✓
PDST	[12]			✓
STRIDE	[13]			✓
PRM	[14]		✓	
LazyPRM	[9]		✓	
RRTstar	[15]	✓		
PRMstar	Based on [14][15]	✓	✓	
LazyPRMstar	Based on [9][15]	✓	✓	
FMT	[16]	✓		✓
BFMT	[17]	✓		✓
LBTRRT	[18]	✓		✓
TRRT	[19]	✓		✓
BiTRRT	[20]	✓		✓
SPARS	[21]	✓	✓	
SPARStwo	[22]	✓	✓	

Kinova JACO^1 and JACO^2 [5]: Kinova has developed two JACO versions, the JACO^1 and the JACO^2. They are lightweight and geometrically identical. The manipulators have a maximum reach of 900mm. Other than the UR5 and LWR 4+, which have straight links, this manipulator has two links which have a curve. Both versions have three joints that can rotate continuously, remaining joints have limits due to the geometry. The maximum payload is 1.5kg for the JACO^1 and 2.6kg for the JACO^2. The manipulator has 6 degrees of freedom. A model is shown in Fig. 3.

B. Software

The open-source Robot Operating System (ROS) is a suite of software libraries and was created to encourage collaborative robotics software development. Inside ROS, the MoveIt! framework deals with the manipulation of robotic hardware. Several motion planner libraries can be configured with MoveIt! to help solve a motion planning problem. OMPL (Open Motion Planning Library) is the most supported motion planner library and is the default library for MoveIt!. The library consists of state of the art sampling-based motion planners.

Using MoveIt!, OMPL creates a path to solve the motion planning problem. By default, OMPL tries to perform path simplification. These are routines that shorten the path. The smoothness of the path may not be affected by this simplification.

C. Overview of planners

Sampling-based motion planners are proven to be probabilistic complete [14], which implicates that the probability of not finding a feasible path in an unbounded setting approaches zero. Therefore, these planners are widely used to find feasible paths in high-dimensional and geometrically constraint environments. Optimizing sampling-based motion planners can refrain from potential high-cost paths and rough motions [15]. However, computing effort for finding an optimized path is increased.

One of the most common is sampling-based motion planner is the Probabilistic RoadMap (PRM) [14]. The planner makes a roadmap by sampling random states in the configuration space and mark them as nodes (vertices). Nodes are connected to other nearby nodes if this path segment (edge) is collision-free. Once the construction of the roadmap is finished a graph search is performed in order to find a connection from the initial state towards the goal state. The roadmap of the PRM planner attempts to cover the total free configuration space, making it suitable to reuse the roadmap for a different motion planning problem in the same configuration space. This is referred to as a multi-query planning method. Among the 23 motion planners in Tab. I, six can be considered as multi-query planning methods.

The LazyPRM [9] planner is different from the PRM method, since it initially accepts invalid configuration states to construct a roadmap. After the graph search is performed, the invalid parts of the candidate solution are altered to make the path collision-free. PRMstar [15] is the asymptotically optimal variant of the PRM planner. It rewires nodes to other near nodes if this minimizes cost towards the node. LazyPRMstar [15] combines the LazyPRM and PRMstar.

The remaining planners in Tab. I are refered to as single-query planning methods. These create a roadmap every time a new planning query has to be solved. A common single-query planner is the Rapidly-exploring Random Tree (RRT) method [10]. It grows a tree structure from the initial configuration state in the direction of the unexplored areas of the bounded free space. This is realized by randomly sampling nodes in the free configuration space, sampled nodes that can be are within a certain distance of tree nodes are added to the tree by edges. The process of adding nodes and edges is repeated until the tree reaches the goal node. The RRTConnect method [11] is a bi-directional version of the RRT method, meaning that two trees are grown. Two processes of RRT are started, one in the start node and one in the goal node. At every iteration or edge addition, it is checked whether the trees can be connected to each other, which solves the the motion planning problem.

The RRT with an optimizing step is the RRTstar [15] planner. This variant of RRT checks whether the new sampled node can be connected to other near nodes so that the state space is more locally refined, similar to PRMstar. The Lower Bound Tree-RRT (LBT-RRT) [18] planner is another optimizing planner. It uses a so-called lower bound graph which is an auxiliary graph. To maintain the tree, a similar method as RRTstar is used. Transition-based RRT or TRRT [19] is a combination of the RRT method and a stochastic optimization method for global minima. It performs transition tests to accept new states to the tree. The Bi-TRRT [20] is a bi-directional version of this planner.

The EST method [7] stands for Expansive Space Trees. The planner tries to determine the direction of the tree by checking the density of nodes in the configuration space.

The tree will expand towards the less explored space. Bi-directional EST (BiEST) [7] grows two trees, similar to RRTConnect. Projection EST (ProjEST), based on [7], detects the less explored area of the configuration space by using a grid. This grid serves as a projection of the configuration space. Single-query Bi-directional probabilistic roadmap planner with Lazy collision checking, also called SBL [6], grows two trees, which expand in the same manner as EST. The planner differentiates from EST by the lazy collision-checking.

KPIECE (Kinodynamic motion Planning by Interior-Exterior Cell Exploration) [8] is a tree-based planner that uses layers of discretization to help estimate the coverage of the state space. There also exists a bi-directional variant called BKPIECE and a variant which incorporates lazy collision checking, this is the LBKPIECE.

Fast Marching Tree (FMT) [16] is an asymptotically optimal planner which marches a tree forward in the cost-to-come space on a specified amount of samples. The BFMT [17] planner is a bi-directional variant of this planner.

PDST (Path-Directed Subdivision Tree) [12] represents samples as path segments instead of configuration states. It uses non-uniform subdivisions to explore the state space.

STRIDE (Search Tree with Resolution Independent Density Estimation) [13] uses a Geometric Nearneighbor Access Tree (GNAT) to estimate the density of the configuration space. This helps to guide the tree into the less explored area.

III. Problem formulation

The available planners consist of non-optimizing and optimizing planners. The problem formulation follows the work of Karaman and Frazzoli [15]. Non-optimizing planners attempt to find a feasible path in the bounded d-dimensional configuration space $C = [0,1]^d$. The free configuration space is defined by $C_{\text{free}} = cl(C \setminus C_{\text{obs}})$, in which $cl(\cdot)$ denotes the closure of a set and in which C_{obs} denotes the obstacle space. A path p is called feasible when:

$$\begin{aligned} p(0) = x_{\text{init}},\, p(1) = x_{\text{goal}} \\ p(x) \in C_{\text{free}} \text{ for all } x \in [0,1] \end{aligned} \tag{1}$$

Optimizing planners that are given a motion planning problem $(C_{\text{free}}, x_{\text{init}}, x_{\text{goal}})$ and a cost function c, find a optimized path p^* such that:

$$c(p^*) = \min\{c(p) : p \textit{ is feasible } \} \tag{2}$$

For the implementation of motion constraints, the free configuration space is reduced. Only configurations that satisfy the motion constraint can be valid configurations. This can result increase the amount of narrow passages, which are known to cause issues for sampling-based motion planners [23].

Problem implementation. Grasp executions will be simulated in geometry constrained scenes inside the MoveIt! framework to retrieve data on planner performance. Non-optimizing planners are instructed to produce feasible paths. The optimizing planners are instructed to produce optimized paths. The motion constraint problem is defined to keep the gripper horizontal. Rotation of the gripper in the horizontal plane is allowed (max 2π), other rotations are limited to 0.1rad. Path simplification by OMPL for all the motion planning problems is turned on. Multi-query planners are being used as single-query planners.

Performance metric. Solved runs, computing time and path length are used as metrics in our experiments. We analyze the metrics outcome individually to provide the best performing planners in each of the metrics. Solved runs is expressed in terms of the percentage of total runs resulting in feasible or optimized paths. High solved runs is considered as high performance. Computing time is measured for the time it takes for planners to produce feasible paths or optimized paths. Planners with a low computing time are considered as high performance. Path length is measured by the length of the sum of motions for a produced path. Planners with short path length are considered as high performance. Mean and standard deviation values of computing time and path length can provide extra information on the performance. Low mean and small standard deviations values are considered as high performance.

Parameters. For 20 of the 23 OMPL planners in MoveIt!, parameters have to be set in order for the planner to solve a motion planning problem. Choosing right parameter values can improve the performance of the planner. To aim for maximum performance an extensive parameter selection was conducted, since no automatic optimization process is available to this date.

IV. Defined motion planning problems

Three grasp execution motions have been defined to measure the performance of the planners. They are defined in the same environment that consists of a simplified shelf and obstacles.

A. Benchmark 1: Place grasp

Benchmark 1 initial end-effector position is located at the end of a shelf, shown as the orange colored robot state in Fig. 1,2,3. The goal position is a stretched arm configuration in front of the shelf (gray robot state). The motion planning problem starts a in a narrow passage, the goal is situated in a less constrained space. This would identify which planner is able to produce the best results when moving out of a constrained space.

B. Benchmark 2: Pick grasp

Benchmark 2 is attempting to resemble a picking motion from a narrow shelf, shown as the gray robot state in Fig. 1,2,3. The goal of the problem is to achieve a specific gripper orientation at the end of this shelf (orange robot state). The planner will have to produce a motion plan with high accuracy to reach the end of the shelf. The motion

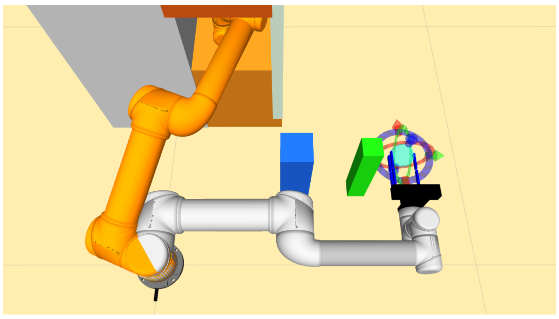


Fig. 1. Pick and place benchmark for UR5.

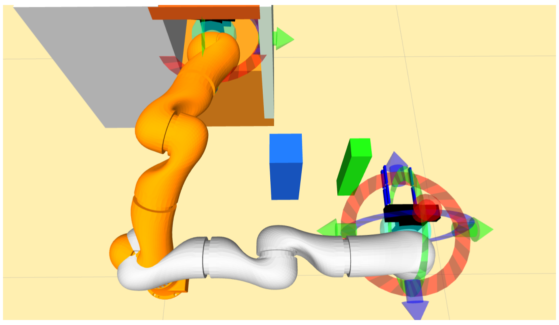


Fig. 2. Pick and place benchmark for KUKA LWR 4+.

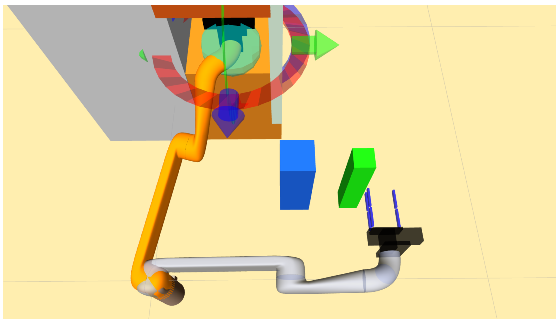


Fig. 3. Pick and place benchmark for Kinova JACO.

planning problem starts in a less constrained space, the goal is situated in a narrow passage. This would identify which planner is able to produce the best results when moving into a constrained space.

C. Benchmark 3: Place grasp with motion constraints

For benchmark 3, the same motion planning problem as benchmark 1 is defined. However, for this problem motion constraints are added to keep the gripper horizontally leveled within a small margin. This resembles placing a glass of water from the shelf on the table, without spilling. This would identify which planner is able to produce the best results when having a constrained free configuration space.

V. Results

A. Methodology

The benchmarking experiments are performed using one thread on a system with an Intel i5 2.70GHz processor and 8Gb of memory. Parameter estimation for the planners is conducted using an extensive iterative process to achieve maximum performance of the planners with respect to the manipulator, these planner specific parameters are presented in Tab. II. The global OMPL parameter $longest\_valid\_segment\_fraction$ is set to 0.005. To give reliable data on the solved runs, computing time and path length, each algorithm was run 50 times for the given motion planning problem. The planners were given a maximum computing time 10s for benchmarks 1 and 2. Due to the increased limitation of the free configuration space of motion constraint planning, benchmark 3 was given 20s maximum computing time. The time is kept low since most robotics applications need to operate quickly.

B. Simulation results

Results of benchmark 1 are shown in Fig. 4 and Tab. III. Considering all manipulators, solved runs of 80% and higher were found for all single-query planners, except for FMT. Since multi-query planners do not focus on one specific motion planning problem, the roadmap construction needs to cover the whole $\mathcal{C}_{\text{free}}$. A demerit of this is the extra needed computing effort. Single-query planners do not need to cover the total free configuration space. Heuristics of these planners help propagating a path outwards of a constrained space. Lowest computing times were retrieved with EST, ProjEST, KPIECE and STRIDE, which are all mono-directional planners with a *goal bias* property. Since the goal configuration is located in a large free space, the probability of finding a solution with the goal configuration as sample increases. The fastest planners also use heuristics to quickly cover the configuration space. EST and STRIDE do this by looking at the density of present samples. KPIECE uses a discretization layer which coarsely covers the configuration space. With goal bias in an open space, short paths can be found with EST, ProjEST and KPIECE.

When considering the results of manipulators in benchmark 1 separately, it can be noted that the JACO manipulator was able to generate higher solved runs for multi-query planners. This manipulator does not incorporate any restrictions on joint limits, which helps to find more connections in the free configuration space between nodes, increasing the solved runs. Moreover, a solution faster is found faster and with a shorter path length. Computing times for the UR5 manipulator were lowest with KPIECE, RRT and RRTConnect. Computing times for the LWR 4+ were lower than the UR5 with SBL, EST, KPIECE, BKPIECE, LBKPIECE and STRIDE. In addition to those planners, RRTConnect also had low computing times for the JACO. For the UR5, short path lengths are found with BiEST, ProjEST, KPIECE, RRTConnect, TRRT and BiTRRT. For the LWR 4+, these are EST, KPIECE, BKPIECE, LBKPIECE and LazyPRMstar. Planners EST, BiEST, ProjEST, KPIECE, LazyPRMstar and BiTRRT found short paths for the JACO manipulator.

Results of benchmark 2 are shown in Fig. 5 and Tab. IV. Considering all manipulators, solved runs of 80% and higher were retrieved with SBL, BKPIECE, LBKPIECE, RRTConnect and BiTRRT. These are all bi-directional tree-based planners. Because of this property path planning is

TABLE II: Planner parameters for UR5 (U), LWR 4+ (L) and JACO (J)

SBL	U	L	J	EST	U	L	J	BiEST	U	L	J	ProjEST	U	L	J	RRT	U	L	J	RRTConnect	U	L	J
range	0.525	0.6	0.6	range	0.6	0.6	0.6	range	0.6	0.6	0.6	range	0.45	0.6	0.6	range	0.75	1.2	1.8	range	0.2	0.6	0.6
				goal bias	0.05	0.05	0.075					goal bias	0.075	0.025	0.075	goal bias	0.075	0.075	0.025				

PRM	U	L	J	LazyPRM	U	L	J	RRTstar	U	L	J	KPIECE	U	L	J	BKPIECE	U	L	J	LBKPIECE	U	L	J
max n.n.	10	10	10	range	0.525	0.6	0.6	range	0.75	1.2	0.6	range	0.525	0.3	0.225	range	0.525	0.3	0.225	range	0.6	0.225	0.3
								goal bias	0.075	0.05	0.05	goal bias	0.075	0.05	0.075	border_f.	0.9	0.9	0.8	border_f.	0.9	0.8	0.8
								delay c.c.	1	1	1	border_f.	0.9	0.9	0.8	exp. s.f.	0.7	0.5	0.5	valid p.f.	0.5	1	0.5
												exp. s.f.	0.7	0.5	0.5	valid p.f.	0.5	0.5	0.5				
												valid p.f.	0.5	0.5	0.5								

STRIDE	U	L	J	FMT	U	L	J	TRRT	U	L	J	BiTRRT	U	L	J	SPARS	U	L	J	SPARStwo	U	L	J
range	0.6	0.45	0.45	num samp.	1000	1000	1000	range	1.35	1.2	1.8	range	0.6	0.9	0.6	stretch f.	3	2.6	2.6	stretch f.	3	3	3
goal bias	0.05	0.05	0.075	rad. Mult.	1.15	1.15	1.15	goal bias	0.075	0.05	0.025	temp c.f.	0.3	0.3	0.3	sparse d.f.	0.25	0.25	0.25	sparse d.f.	0.25	0.25	0.25
use proj d.	0	0	0	Nearest k	1	1	1	max state f.	5	5	5	init temp	75	75	75	dense d.f.	0.001	0.001	0.001	dense d.f.	0.001	0.001	0.001
degree	24	8	8	cache cc	1	1	1	temp c.f.	2.5	2	2	frountier	1	1	1	max fail.	1000	1000	1000	max fail.	5000	5000	5000
max deg.	28	12	12	heuristics	1	1	1	min temp	1e-9	1e-9	1e-9	frountierN.r.	0.1	0.1	0.1								
min deg.	12	6	6	ext. fmt	1	1	1	init temp	125	125	125	cost thres.	1000	1000	1000								
max p.p.l.	6	3	3					frountier	2.25	2.5	2.5												
est. dim.	0	0	0					frountierN.r.	0.1	0.1	0.1												
valid p.f.	0.1	0.1	0.1					k constant	0	0	0												

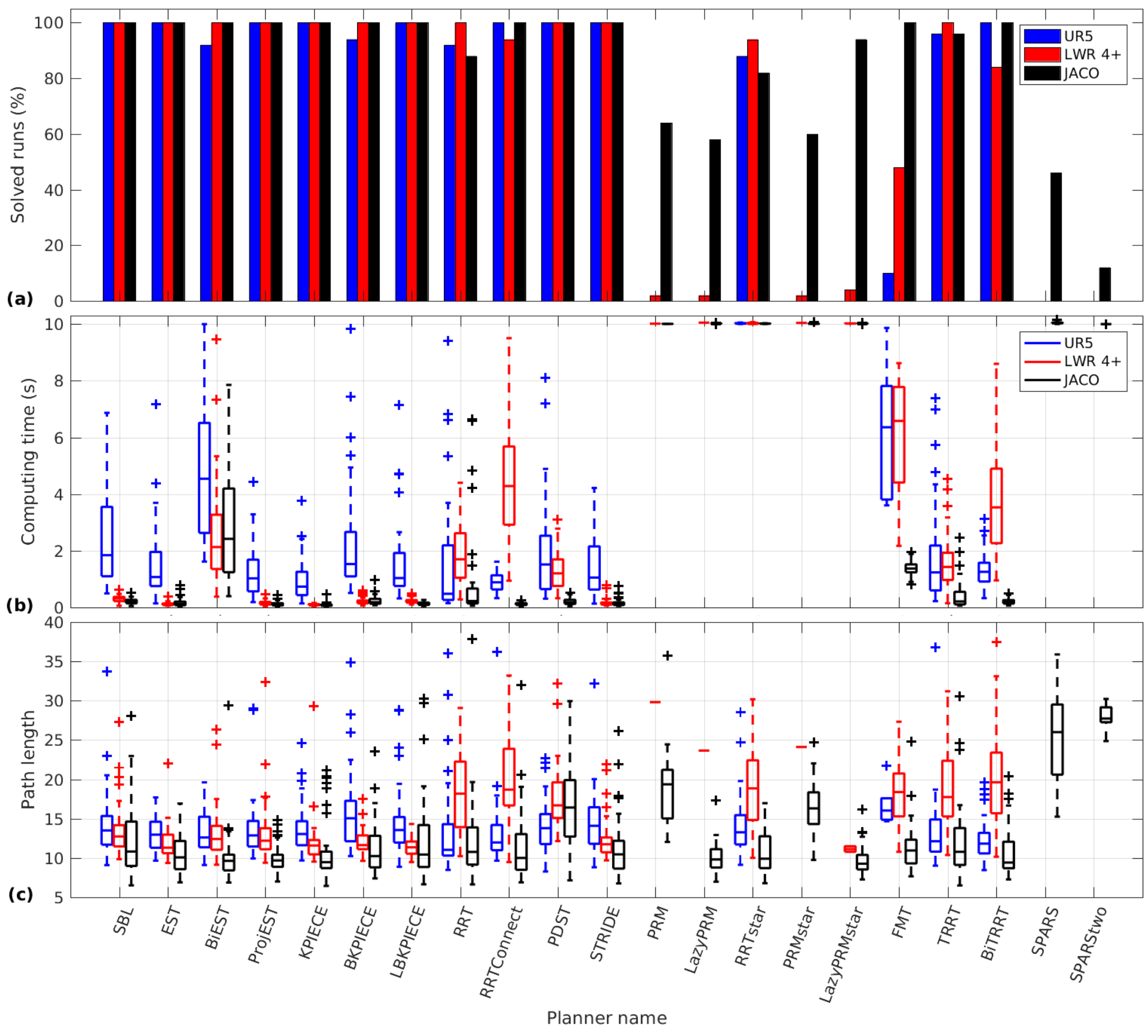


Fig. 4. Results for benchmark 1. (a) Solved runs; higher is better. (b) Computing time; lower is better, small interquartile range is better. (c) Path length; lower is better, small interquartile range is better.

also started in the goal state, which acts similar to the benchmark 1 moving out of a constrained space. SBL and LBKPIECE showed the lowest computing times. These planners use lazy collision-checking. Collision-checking is only performed on a candidate path instead on all vertices, which can lower the computing time. Shortest paths are found with SBL and LBKPIECE, however, standard deviations are higher for the SBL planner. Using a discretization layer to cover the configuration space coarsely helps finding more consistent results (lower standard deviations from the mean).

When considering the results of manipulators in benchmark 2 separately, it can be noted that the UR5 and LWR 4+ also managed to produce solved runs of 80% and

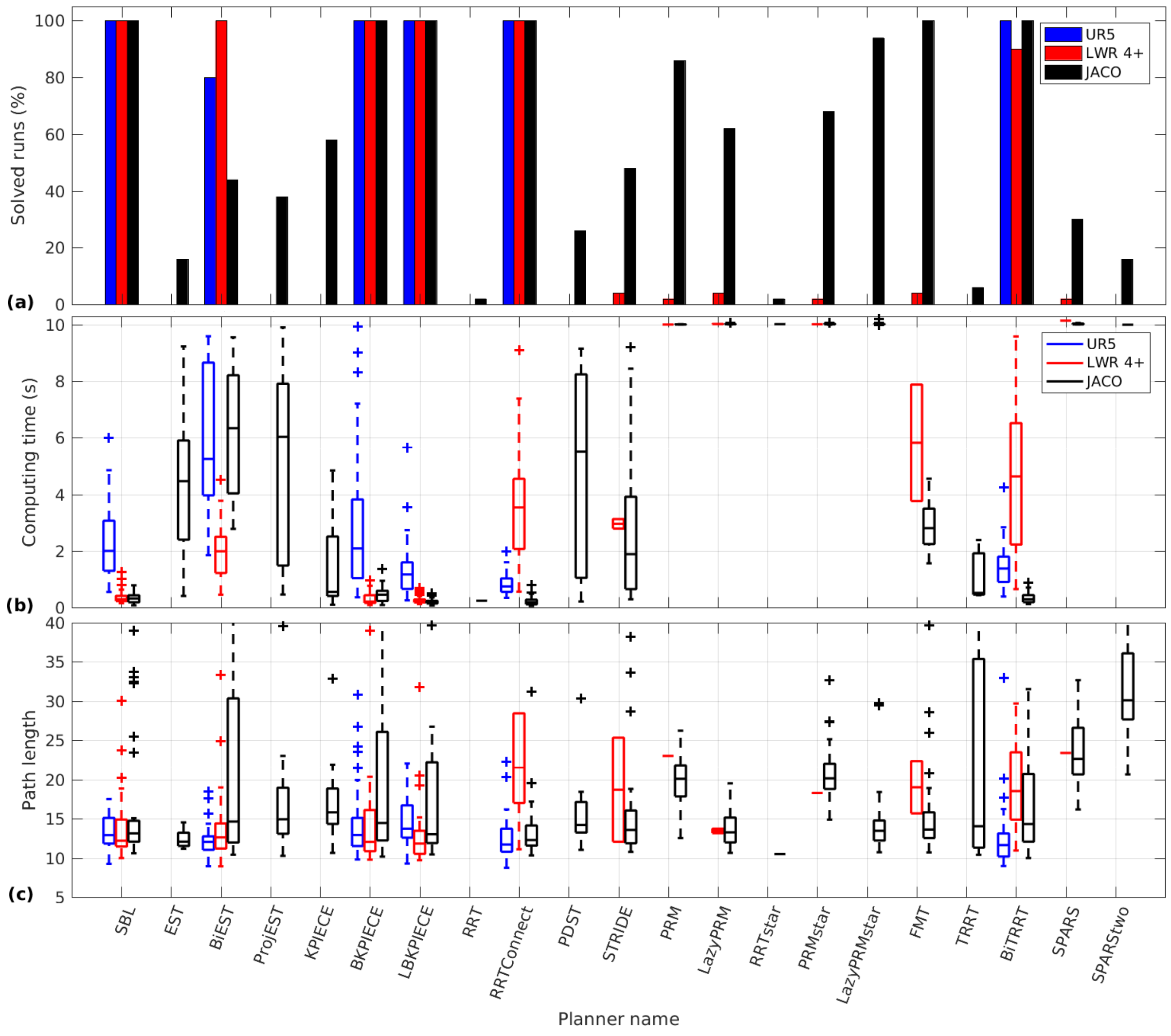


Fig. 5. Results for benchmark 2. (a) Solved runs; higher is better. (b) Computing time; lower is better, small interquartile range is better. (c) Path length; lower is better, small interquartile range is better.

higher for the BiEST planner. This planner is also a bi-directional planner. The expanded configuration space of the JACO manipulator helps to produce solved runs of 80% and higher for planners PRM, PRMstar and LazyPRMstar. However, computing times are considerably higher since these planners keep optimizing the roadmap until the time-limit is reached. RRTConnect is the fastest planner for the UR5 and LBKPIECE is the fastest for LWR 4+ and JACO. For all three manipulators, LBKPIECE is able to compute feasible paths within 1.5s. Considering high solved runs: BiTRRT finds the shortest paths for the UR5, SBL for the LWR 4+ and FMT for the JACO respectively.

Results of benchmark 3, incorporating motion constraints, are shown in Fig. 6 and Tab. V. The extra constraint limits the free configuration space. Considering all manipulators, only the BiEST planner was able to produce solved runs of 80% and higher. The planner looks at the density of samples in its neighbourhood to help its expansion. In the case of planning with motion constraints, this shows to be effective compared to the other 20 planners. The bi-directional property helps finding a solution within the time limit, since the mono-directional EST planner is not able to find a feasible path with a maximum computing time of 20s.

Depending on the manipulator, the highest performing planner differs, which indicates that there is less consistency in planner performance when incorporating extra motion constraints. SBL, BKPIECE and RRTConnect also had solved runs of 80% and higher for the UR5 manipulator. For the JACO manipulator the KPIECE planner manages to get solved runs of 100%.

C. Discussion

The motivation of this work is to help users pick high-performing motion planners for grasp executions. Shortcomings of this work will be discussed.

Computing time. This paper only showed results for motion planning with a time-constraint of 10s or 20s. This was chosen to select high-performing motion planners that find a solution in a timely manner. Selecting a higher time-limit could show increased performance in solved runs and path length. However, this is not covered in this work.

TABLE III: MEAN VALUES FOR BENCHMARK 1

Planner name	UR5		LWR 4+		JACO	
	Time (s)	Path length	Time (s)	Path length	Time (s)	Path length
SBL	2.42 (1.68)	14.24 (4.11)	0.31 (0.11)	13.38 (3.23)	0.21 (0.09)	15.93 (19.85)
EST	1.54 (1.26)	14.57 (7.50)	0.12 (0.06)	11.82 (2.00)	0.16 (0.15)	10.55 (2.41)
BiEST	4.71 (2.25)	13.73 (4.93)	2.55 (1.77)	13.02 (3.21)	2.98 (2.04)	10.63 (5.47)
ProjEST	1.22 (0.89)	13.70 (3.73)	0.15 (0.08)	14.31 (9.77)	0.12 (0.07)	10.71 (5.77)
KPIECE	0.97 (0.73)	13.72 (3.07)	0.10 (0.03)	11.91 (2.86)	0.09 (0.06)	10.73 (3.62)
BKPIECE	2.31 (1.94)	16.55 (7.70)	0.21 (0.11)	11.92 (1.47)	0.23 (0.16)	11.22 (3.39)
LBKPIECE	1.51 (1.30)	14.45 (4.16)	0.23 (0.09)	11.48 (1.08)	0.13 (0.05)	12.17 (5.20)
RRT	1.59 (2.04)	14.39 (9.07)	1.94 (1.12)	18.68 (6.05)	0.83 (1.59)	12.14 (5.10)
RRTConnect	0.89 (0.35)	13.06 (3.98)	4.49 (2.06)	21.34 (10.20)	0.13 (0.06)	13.88 (12.11)
PDST	1.88 (1.62)	16.27 (17.34)	1.25 (0.61)	25.74 (36.23)	0.20 (0.11)	16.18 (5.22)
STRIDE	1.46 (1.13)	16.36 (15.01)	0.16 (0.14)	12.35 (2.51)	0.16 (0.13)	14.66 (20.43)
PRM			10.02 (0.00)	29.82 (0.00)	10.01 (0.00)	18.73 (4.55)
LazyPRM			10.05 (0.00)	23.66 (0.00)	10.02 (0.01)	10.11 (2.17)
RRTstar	10.02 (0.01)	14.32 (4.15)	10.03 (0.02)	19.06 (5.04)	10.02 (0.01)	10.66 (2.73)
PRMstar			10.04 (0.00)	24.11 (0.00)	10.02 (0.01)	16.65 (3.28)
LazyPRMstar			10.02 (0.00)	11.13 (0.50)	10.02 (0.01)	9.79 (1.86)
FMT	6.17 (2.57)	16.71 (2.88)	6.06 (1.92)	18.29 (4.32)	1.39 (0.25)	11.30 (2.98)
TRRT	1.76 (1.68)	13.40 (4.31)	1.63 (0.93)	20.47 (12.40)	0.41 (0.51)	13.48 (8.55)
BiTRRT	1.31 (0.60)	13.79 (11.02)	3.61 (1.84)	21.05 (8.62)	0.20 (0.10)	10.77 (3.19)
SPARS					10.04 (0.03)	25.52 (6.04)
SPARStwo					10.00 (0.00)	27.83 (1.84)

Standard deviation in parentheses
Gray cells $\rightarrow$ *time* within $2 \cdot \text{time}_{\text{min}}$ and *path length* within $1.2 \cdot \text{path\_length}_{\text{min}}$, for solved runs $> 80\%$

TABLE IV: MEAN VALUES FOR BENCHMARK 2

Planner name	UR5		LWR 4+		JACO	
	Time (s)	Path length	Time (s)	Path length	Time (s)	Path length
SBL	2.30 (1.36)	35.21 (99.24)	0.36 (0.22)	20.66 (37.57)	0.33 (0.16)	19.85 (18.23)
EST					4.40 (2.94)	12.41 (1.20)
BiEST	5.82 (2.52)	20.99 (39.61)	1.99 (0.95)	26.90 (35.64)	6.37 (2.26)	24.80 (20.19)
ProjEST					5.13 (3.25)	18.57 (10.76)
KPIECE					1.37 (1.38)	22.36 (20.44)
BKPIECE	2.83 (2.34)	16.55 (17.33)	0.29 (0.21)	37.36 (73.78)	0.45 (0.25)	24.45 (20.39)
LBKPIECE	1.34 (0.95)	19.01 (23.57)	0.28 (0.15)	21.24 (31.19)	0.20 (0.08)	21.46 (18.29)
RRT					0.24 (0.00)	61.77 (0.00)
RRTConnect	0.81 (0.36)	26.84 (63.29)	3.62 (1.89)	72.72 (112.46)	0.21 (0.14)	20.70 (30.91)
PDST					5.02 (3.42)	15.83 (4.94)
STRIDE			2.96 (0.24)	18.69 (9.38)	2.75 (2.54)	17.40 (10.26)
PRM			10.01 (0.00)	23.00 (0.00)	10.01 (0.00)	19.93 (2.99)
LazyPRM			10.03 (0.00)	13.43 (0.40)	10.02 (0.01)	13.97 (2.58)
RRTstar					10.02 (0.00)	10.48 (0.00)
PRMstar			10.02 (0.00)	18.28 (0.00)	10.03 (0.02)	20.68 (3.62)
LazyPRMstar					10.03 (0.05)	14.29 (3.88)
FMT			5.82 (2.92)	19.01 (4.71)	2.87 (0.83)	17.15 (12.78)
TRRT					1.11 (1.10)	22.31 (17.58)
BiTRRT	1.45 (0.70)	14.49 (13.19)	4.47 (2.57)	65.18 (125.80)	0.33 (0.17)	25.30 (23.12)
SPARS			10.15 (0.00)	23.39 (0.00)	10.03 (0.02)	23.24 (4.76)
SPARStwo					10.00 (0.00)	31.47 (7.23)

Standard deviation in parentheses
Gray cells $\rightarrow$ *time* within $2 \cdot \text{time}_{\text{min}}$ and *path length* within $1.2 \cdot \text{path\_length}_{\text{min}}$, for solved runs $> 80\%$

TABLE V: MEAN VALUES FOR BENCHMARK 3

Planner name	UR5		LWR 4+		JACO	
	Time (s)	Path length	Time (s)	Path length	Time (s)	Path length
SBL	8.88 (4.19)	20.66 (8.09)			12.44 (0.00)	13.55 (0.00)
EST	8.99 (7.49)	18.03 (4.03)	9.08 (0.00)	67.08 (0.00)	11.06 (4.46)	12.25 (1.48)
BiEST	2.81 (1.21)	21.18 (10.97)	8.77 (4.36)	35.71 (26.06)	8.72 (6.00)	27.84 (16.28)
ProjEST					10.06 (5.40)	13.41 (1.25)
KPIECE	9.08 (3.45)	26.58 (12.90)	9.50 (6.58)	41.99 (39.29)	3.84 (2.70)	15.10 (4.43)
BKPIECE	5.72 (4.82)	24.92 (13.76)	9.92 (4.51)	40.82 (30.96)	9.78 (7.12)	44.91 (24.50)
LBKPIECE	15.21 (4.39)	25.39 (13.22)			9.88 (0.00)	12.85 (0.00)
RRT	9.06 (5.63)	29.45 (24.11)	12.32 (5.71)	43.74 (42.21)	12.10 (5.65)	12.28 (1.80)
RRTConnect	4.86 (3.63)	27.92 (23.82)	13.03 (4.00)	65.97 (55.08)	10.74 (5.79)	26.76 (16.25)
PDST	9.23 (0.00)	11.77 (0.00)	12.42 (5.30)	41.06 (39.27)	8.86 (6.06)	51.48 (32.37)
STRIDE	9.98 (10.20)	58.23 (60.47)	12.64 (0.00)	13.15 (0.00)	12.45 (7.28)	23.87 (15.41)
PRM	20.12 (0.04)	65.88 (12.16)	20.14 (0.05)	43.60 (7.56)	20.09 (0.05)	55.03 (20.03)
RRTstar	20.07 (0.04)	22.89 (13.78)	20.03 (0.01)	25.18 (11.74)	20.05 (0.03)	14.97 (3.80)
PRMstar	20.15 (0.13)	41.66 (36.26)	20.18 (0.18)	41.76 (14.76)	20.09 (0.05)	48.12 (27.74)
LazyPRMstar	20.04 (0.01)	46.70 (7.36)	20.87 (1.52)	37.99 (12.06)	21.91 (6.20)	30.31 (13.98)
FMT	19.34 (1.27)	18.54 (8.12)			15.21 (2.42)	15.31 (8.70)
TRRT	13.14 (5.03)	33.26 (30.84)	7.21 (5.60)	17.36 (2.33)	8.27 (5.15)	13.00 (2.45)
BiTRRT	14.47 (5.82)	29.01 (18.58)			8.46 (8.52)	11.54 (0.48)

Standard deviation in parentheses
Gray cells $\rightarrow$ *time* within $2 \cdot \text{time}_{\text{min}}$ and *path length* within $1.2 \cdot \text{path\_length}_{\text{min}}$, for solved runs $> 80\%$

Parameter selection. Since parameter values have to be set for a planner to operate, the aim was to achieve maximum performance of the planners. By manually conducting the iterative process explained before, a guarantee of maximum performance cannot be given. We executed the iterative process to the best of our abilities in order to achieve maximum performance. For planners with an exposed *range* parameter, the distance has to be low enough to provide dense coverage of the configuration space.

Multi-query. For this paper, we only considered single-query motion planning performance. This paper fails to show the potential benefit of lower computing times when using the same roadmap multiple times. For benchmark 1 and 2, we do notice that planners single-query planners are able to compute feasible paths in short amount of time. We therefore argue the need for multi-query planners for online grasp executions similar to benchmarks 1 and 2. The use of multi-query planners can be beneficial when motion constraints have to be used all the time. Computing paths for these motion planning problems can consume more time, as shown in the results. Using the same roadmap again will decrease computing time. However, this map needs to be detailed and the environment needs to be static.

Optimization with time-invariant goal. BiTRRT is the fastest optimizing planner. However, compared to non-optimizing planners, the path length is not consistently shorter. More research needs to be conducted to show the real potential of optimizing planners.

Manipulators. The manipulators studied in this paper have similar specifications. The effect of different manipulator on planner performance cannot be verified with the presented benchmark data. We believe similarly shaped manipulators will yield a similar planner choice. Best overall planner performance, with respect to solved runs, can be obtained with a JACO manipulator.

BFMT and LBTRRT. These planners resulted in errors for the defined motion planning problems. We were unable to provide reliable results to present in this paper. More effort is needed to make these planners work more reliable.

VI. CONCLUSION

This paper presented benchmark data for 21 of the current 23 OMPL planners in MoveIt! for three different manipulators. This data can be useful when performing similar grasp executions. Simultaneously, this paper selected high-performing planners for different motion planning problems, resembling a grasp execution. Planner performance was studied by means of solved runs, computing time and path length. The results showed that the mono-directional KPIECE planner was highest performing when initiating motion planning from a constrained configuration towards a less constrained space. Bi-directional planners with lazy collision-checking (SBL and LBKPIECE) showed fastest performance when the goal configuration is located within a constrained space. Shorter paths were found with LBKPIECE. For motion planning problems incorporating a motion constraints, consistent high performance over the three manipulators was retrieved with BiEST. Considering all the grasp executions presented in this work, RRTConnect was the most reliable planner due to high solved runs. For future work, we would like to investigate the option to implement a faster, easier and more robust method to select parameter values for the planners

REFERENCES

[1] I. Sucan, M. Moll, and L. Kavraki, "Open Motion Planning Library: A Primer," 2014.

[2] I. A. Sucan and S. Chitta, "MoveIt!" 2013.

[3] Universal Robots, "Technical Specifications UR5," accessed 2017-03-13. [Online]. Available: https://www.universal-robots.com/

[4] KUKA, "KUKA LWR," accessed 2017-03-13. [Online]. Available: http://www.kukakore.com/

[5] Kinova Robotics, "Technical Specifications," accessed 2017-03-13. [Online]. Available: http://www.kinovarobotics.com/

[6] G. Sánchez and J.-C. Latombe, "A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking," in *Robotics Research*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 403–417.

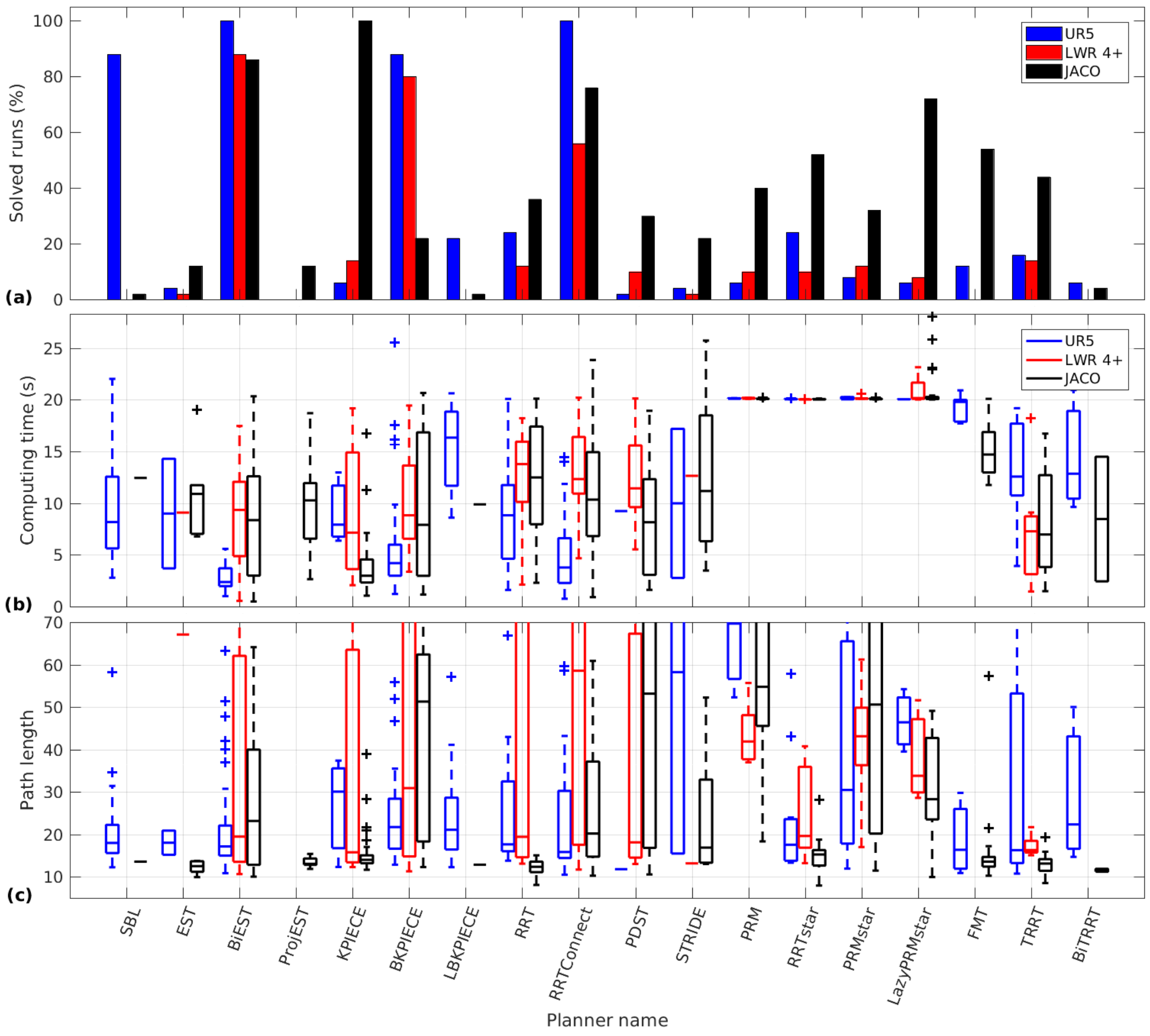


Fig. 6. Results for benchmark 3. (a) Solved runs; higher is better. (b) Computing time; lower is better, small interquartile range is better. (c) Path length; lower is better, small interquartile range is better.

[7] D. Hsu, J. C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," in *Proceedings of Int. Conf. on Robotics and Automation*, vol. 3, 1997, pp. 2719–2726 vol.3.

[8] I. A. Sucan and L. E. Kavraki, "Kinodynamic motion planning by interior-exterior cell exploration," in *In Workshop on the Algorithmic Foundation of Robotics*, 2008.

[9] R. Bohlin and L. E. Kavraki, "Path Planning Using Lazy PRM," in *In IEEE Int. Conf. Robot. & Autom*, 2000, pp. 521–528.

[10] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998.

[11] J. J. Kuffner Jr. and S. M. Lavalle, "RRT-Connect: An efficient approach to single-query path planning," in *Proc. IEEE Intl Conf. on Robotics and Automation*, 2000, pp. 995–1001.

[12] A. M. Ladd, R. Unversity, L. E. Kavraki, and R. Unversity, "Motion planning in the presence of drift, underactuation and discrete system changes," in *In Robotics: Science and Systems I*. MIT Press, 2005, pp. 233–241.

[13] B. Gipson, M. Moll, and L. E. Kavraki, "Resolution Independent Density Estimation for motion planning in high-dimensional spaces," in *2013 IEEE Int. Conf. on Robotics and Automation*. IEEE, may 2013, pp. 2437–2443.

[14] L. Kavraki, P. Svestka, J. claude Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," in *IEEE Int. Conf. on Robotics and Automation*, 1996, pp. 566–580.

[15] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *CoRR*, vol. abs/1105.1186, 2011.

[16] L. Janson and M. Pavone, "Fast marching trees: a fast marching sampling-based method for optimal motion planning in many dimensions - extended version," *CoRR*, vol. abs/1306.3532, 2013.

[17] J. A. Starek, J. V. Gómez, E. Schmerling, L. Janson, L. Moreno, and M. Pavone, "An asymptotically-optimal sampling-based algorithm for bi-directional motion planning," *CoRR*, vol. abs/1507.07602, 2015.

[18] O. Salzman and D. Halperin, "Asymptotically near-optimal RRT for fast, high-quality, motion planning," *CoRR*, vol. abs/1308.0189, 2013. [Online]. Available: http://arxiv.org/abs/1308.0189

[19] L. Jaillet, J. Corts, and T. Simon, "Sampling-based path planning on configuration-space costmaps," *IEEE Transactions on Robotics*, pp. 635–646, 2010.

[20] D. Devaurs, T. Siméon, and J. Cortés, "Enhancing the transition-based rrt to deal with complex cost spaces," in *IEEE Int. Conf. on Robotics and Automation, ICRA '13*, Karlsruhe, Germany, 2013, pp. 4105–4110.

[21] A. Dobson, A. Krontiris, and K. E. Bekris, *Sparse Roadmap Spanners*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 279–296.

[22] A. Dobson and K. E. Bekris, "Improving sparse roadmap spanners," Karlsruhe, Germany, 2013.

[23] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin, "On finding narrow passages with probabilistic roadmap planners," 1998.

Acknowledgements

Helps of others (my supervisors, my friends and my family) made it possible for me to successfully finish the research in this thesis.

I would like to start by thanking Professor Jiaxu Wang, My master's thesis supervisor and China scholarship council (CSC). Doing a PhD research abroad was my biggest dream since when I was a freshman in my bachelor study. This is not because I can work with people from different education background, but also I can broaden my views on different cultures. It is Professor Jiaxu Wang and CSC who helped me to achieve my dream of study overseas. I probably could not have the chance to experience such a surprising and fruitful four-year's research life in TU Delft without the encouragement and support from Professor Jiaxu Wang during the preparation for studying abroad and the financial sponsor from CSC.

Under the encouragement of Professor Jiaxu Wang, I sent an email to Professor Martijn Wisse to show my interest to be a PhD candidate at TU Delft under his supervision. My memory is still fresh when I received the reply email from Martijn on May 12, 2012. I was told in the reply email that he is interested in my CV and he wanted to arrange a Skype meeting with me. Several days later after the Skype meeting, I received an acceptance letter from Martijn. In January of 2013, I started my PhD journey in the group of Delft Biorobotics Lab (DBL). In this journey, I had confusions, impatience and nervousness. However, I received numerous patience, encouragement, guidance and scientific inspiration from Martijn. I learned many useful strategies of solving problems in research and life from Martijn. During the past four years of my PhD research, Martijn spent quite much time on discussing research questions with me. Also similarly much time he used to read my papers. To help me submit a paper in time, he even read and corrected my papers in evenings and weekends. Besides his efforts on helping me make progress on my research, he also put energy to encourage me and increase my work efficiency. Furthermore, he provides me many precious opportunities to attend international conferences to building my confidence in interpreting my research. I am really thankful for Martijn's help and I will remember the valuable research experience under his supervision.

Apart from my supervisors, I am eager to express my thanks to my friends and DBLers. Undoubtedly, my PhD cannot be so impressive without the company of DBLers including Berk Calli, Wouter Caarls, Tim Vercruyssen, Mukunda Bharatheesha, Gijs van der Hoorn, Xin Wang, Shiqian Wang, Jun Zhang, Jeff van Egmond, Michiel Plooij and Wouter Wolfslag. In addition to lovely DBLers, I would thank my friends who helped me on my research: Zichao Pan, Lei Zhou, Tao Lu, Guangming Chen, Peng Qi, Juan Wang, Yiping Zuo and many others.

Finally, I would like to thank my family: my father and mother. I would probably never have considered to doing a PhD without the support from my family. This thesis is dedicated to them as their endless love, support and understanding make so many beautiful things in my life possible.

Qujiang

About the author

Qujiang Lei

June 13, 1987

Born in Sichuan Province, China

2003.09-2006.07

Quxian Sanhui Middle School

Quxian, Sichuan Province, China

2006.09-2010.07

Bachelor of Engineering in the major of Mechanical Design and Automation

Southwest University of Science and Technology, Mianyang, China

2010.09-2012.12

Master of Engineering in the major of Mechanical Design and Theory

Chongqing University, Chongqing, China

2013.01-2017.01

PhD researcher in the Department of BioMechanical Engineering

Delft University of Technology, Delft, The Netherlands

List of publications

Conference papers

[1] Qujiang Lei, Martijn Wisse, “Fast grasping of unknown objects using force balance optimization”, 2014 IEEE International Conference on Intelligent Robots and Systems (IROS 2014), pp. 2454–2460. Chicago, USA.

[2] Qujiang Lei, Martijn Wisse, “Unknown object grasping using force balance exploration on a partial point cloud”, 2015 IEEE International Conference on Advanced Intelligent Mechatronic (AIM 2015), pp. 7–14. Busan, Korea.

[3] Qujiang Lei, Martijn Wisse, “Object grasping by combining caging and force closure”, 2016 IEEE International Conference on Control, Automation, Robotics and Vision (ICARCV 2016), pp. 1–8. Phuket, Thailand.

[4] Qujiang Lei, Martijn Wisse, “Fast grasping of unknown objects using cylinder searching on a single point cloud”, 2016 9th International Conference on Machine Vision (ICMV 2016). Nice, France.

[5] Qujiang Lei, Martijn Wisse, “Unknown object grasping by using concavity”, 2016 IEEE International Conference on Control, Automation, Robotics and Vision (ICARCV 2016), pp. 1–8. Phuket, Thailand.

[6] Qujiang Lei, Jonathan Meijer, Martijn Wisse, “A Survey of Unknown Object Grasping and Our Fast Grasping Algorithm-C Shape Grasping”, 2017 IEEE International Conference on Control, Automation and Robotics (ICCAR 2017), pp. 150–157. Nagoya, Japan.

[7] Qujiang Lei, Martijn Wisse, “Fast C-shape grasping for unknown objects”, 2017 IEEE International Conference on Advanced Intelligent Mechatronic (AIM 2017), pp. 509 – 516, Munich, Germany.

[8] Jonathan Meijer, Qujiang Lei, Martijn Wisse, “An Empirical Study of Single-Query Motion Planning for Grasp Execution”, 2017 IEEE International Conference on Advanced Intelligent Mechatronic (AIM 2017). pp. 1234 – 1241. Munich, Germany.

[9] Jonathan Meijer, Qujiang Lei, Martijn Wisse, “Performance Study of Single-Query Motion Planning for Grasp Execution Using Various Manipulators”, 2017 18th International Conference on Advanced Robotics (ICAR 2017). pp. 450 – 457. Hong Kong.

[10] Berk Calli, Wouter Caarls, Qujiang Lei, Martijn Wisse, Pieter Jonker, “SMAG: Simultaneous Modeling and Grasping”, in RSS workshop, Robotics: Science and Systems (RSS 2013). Berlin, Germany.

Journal papers

[1] Qujiang Lei, Guangming Chen, Martijn Wisse, “Fast grasping of unknown objects using principal component analysis”, in Journal of AIP Advances, Volume: 7, Issue: 9, Pages: 1-21, 2017.

[2] Qujiang Lei, Jonathan Meijer, Martijn Wisse, “Fast grasping of unknown objects using C-shape configuration on a single view partial point cloud”, accepted by Journal of AIP Advances.