Fast C-shape grasping for unknown objects

Lei, Qujiang; Meijer, Jonathan; Wisse, Martijn

DOI
10.1109/AIM.2017.8014068

Publication date
2017

Document Version
Accepted author manuscript

Published in
Proceedings 2017 IEEE International Conference on Advanced Intelligent Mechatronics

Fast C-shape grasping for unknown objects

Qujiang Lei, Jonathan Meijer, Martijn Wisse

***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 this paper, we propose an original fast grasping algorithm for unknown objects. The geometry of the under-actuated gripper is approximated as a C-shape, which is used to fit the point cloud of the target object to find a suitable grasp. In order to make the robot arm quickly execute the grasp found by the grasping algorithm, we made a comparison of the popular online motion planners. The motion planner with the highest solved runs, lowest computing time and the shortest path length is chosen to execute the grasp action. Simulations and experiments on a UR5 robot arm and an under-actuated gripper are used to examine the performance of the grasping algorithm, and successful results are obtained.**

I. Introduction

A. What is the goal of this paper?

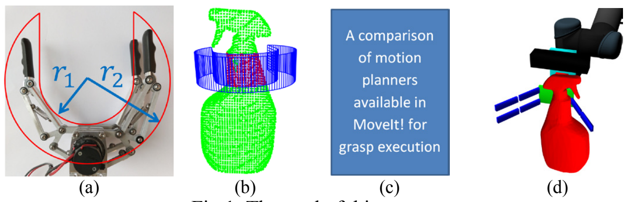

r_1
r_2
A comparison of motion planners available in MoveIt! for grasp execution
(a)
(b)
(c)
(d)

Fig.1. The goal of this paper.

The goal of this paper is to design a fast and general grasping algorithm for unknown objects. The outline of this paper is shown as Fig.1. Specifically, this grasping algorithm is specially designed for under-actuated grippers shown as (a), explanation about why we choose such kind of under-actuated grippers will be given latter. After a suitable grasp is found as (b) shows, a comparison of motion planners (shown as (c)) is conducted in order to quickly execute the grasp. According to the current trend of motion planning, we compared all the motion planners available in MoveIt!. The motion planner with the highest solved runs, lowest computing time and the shortest path length is chosen to execute the grasp found by the grasping algorithm. An example of grasp execution is shown as (d).

B. Existing fastest grasping algorithms for unknown objects

Table I shows the five fastest grasping algorithms of unknown objects from literature study. These fast grasping algorithms in table I from left to right are in chronologic order. We can find some interesting things: Except [2], the other four fast grasping algorithms are designed for parallel grippers. Only [1] uses RGB images as input of the grasping algorithm, the rest four grasping algorithms employ a partial point cloud as input. The above two findings inspired us to create a more

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

Table I. The five fastest grasping algorithms from literature study

Grasping algorithms	Baumgartl [1]	Eppner [2]	Lin [3]	Ten Pas [4]	Suzuki [5]
Year	2012	2013	2014	2015	2016
Hardware	four cores with 2.66GHz	single core with 2.2GHz	unknown	four cores with 3.5GHz	four cores with 2.5GHz
Time	34ms	3s	2.352s	2.7s	4.2s
Input	RGB images	A Partial point cloud	A Partial point cloud	A Partial point cloud	A Partial point cloud
Robot hand	A parallel gripper	A Barrett Hand	A parallel gripper	A parallel gripper	A parallel gripper

general and faster grasping algorithm for simple grippers by using a partial point cloud as input.

[1] is a pretty fast grasping algorithm, which uses Hough transformation to find the edges of objects in a 2D image. A check has been done to see if the edges are long enough to be grabbed by the gripper and another check is followed to see if the parallel edges suit the gripper's width. [3] uses the contact area of the grasping rectangle to find suitable grasps. If the contact area is too small, the grasp is likely to fail, and then a better grasp need to be picked. [5] uses principal axis and centroid of the object to synthesize a grasp. The above three fast grasping algorithms have a common character, that is, they all use the normal of the table plane as the grasp approaching direction, which can accelerate grasp searching. However, this kind of simplification cannot be widely used, because grasping from top is not applicable for most objects, for example, objects in fridges or shelves.

In the work of Eppner [2], the point cloud is transformed into shape primitives (cylinder, disk, sphere and box). A pre-grasp (configuration of the hand) is chosen according to those shape primitives. This kind of shape primitives can greatly 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.

Ten Pas [4] 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 is very efficient. However, the parallel gripper is not good at flexibility comparing with dexterous hands and under-actuated grippers.

To sum up, in this paper, we aim to design a more general and faster grasping algorithm than the above five fast grasping algorithms, Meanwhile, in order to make our grasping algorithm more flexible, we will adopt under-actuated grippers.

C. Why we choose under-actuated gripper?

As we said before, among the five fast grasping algorithms, four of them choose to use parallel grippers because parallel grippers have simpler geometry shape and they are easier to control. One more thing is that parallel grippers are cheap so that their grasping algorithms can be widely used. But all of them ignore a kind of excellent robot hands, that is, under-actuated grippers.

Table II. Three popular robot hands and a short comparison of them.

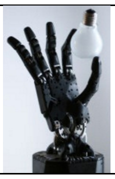
Three popular robot hands			
Flexibility	+++	+	--
Complexity	+++	-	--
price	+++	-	--

Table II shows three popular robot hands, that is, a dexterous hand, an under-actuated gripper and a parallel gripper. Even though dexterous hands are very good at flexibility, but the high complexity and high price stop them to become popular 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 are under-actuated grippers. Under-actuated grippers are a very good tradeoff between dexterous hands and parallel grippers. Fig.2 shows three popular cheap commercial under-actuated grippers.

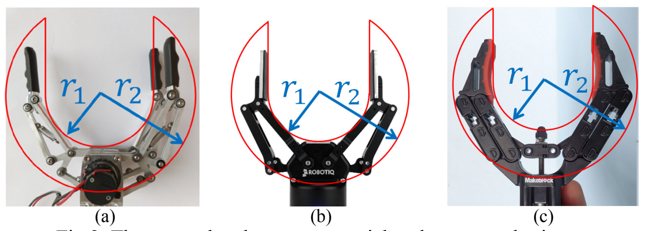


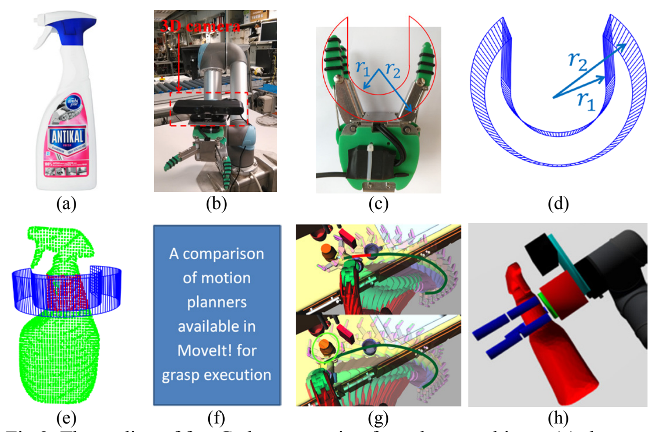
Fig.2. Three popular cheap commercial under-actuated grippers.

D. *Our fast grasping algorithm*

In this paper, we propose an original fast grasping algorithm for unknown objects. The outline of our fast grasping algorithm is shown as Fig.3. The under-actuated grasping gripper is simplified as a C-shape cylinder (shown as Fig.3 (c) and Fig.3 (d)) with radius r_1 and r_2 respectively. The algorithm will perform C-shape searching on the partial point cloud of the target object to quickly synthesize an executable grasp. Specifically, Fig.3 (b) shows a setup consisting of a robot arm equipped with a 3D camera and an under-actuated gripper. A spray bottle in Fig.3 (a) works as an example of an unknown object. The gripper in Fig.3 (b) is described as a C-shape with radius r_1 and r_2. The C-shape is used to match with the partial point cloud of the target object to work out an executable grasp. Fig.3 (e) shows an example of executable grasps found by our grasping algorithm. The red points on the object stand for the corresponding grasp area. Fig.3 (h) shows the grasp execution for the spray bottle. Details about our grasping algorithm will be explained in section III.

E. *Comparison 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


3D camera
r_1
r_2
A comparison of motion planners available in MoveIt! for grasp execution
(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
Fig.3. The outline of fast C-shape grasping for unknown objects. (a) shows an example of an unknown object. (b) shows a robot arm equipped with a 3D camera and an under-actuated gripper. (c) and (d) show the inspiration of this paper, the under-actuated gripper is simplified as a C-shape. (e) shows an example grasp found by our algorithm. In order to choose a suitable online motion planner for grasp execution, we made a comparison of existing online motion planners in (f). A good example of motion path found by the motion planner is shown in (g). (h) demonstrates the grasp found in (e) is executed.

action, we will make a comparison of different online motion planners in this paper.

F. *Organization of this paper*

The rest of this paper is arranged as follows: Section II shows the comparison of different motion planners. Section III contains a detailed explanation of our fast grasp grasping algorithm. Section IV demonstrates the simulation results. Section V is the experiment results and Section VI gives a discussion about our fast grasping algorithm and the other five popular fast grasping algorithms mentioned above. Section VII is the conclusion of this paper.

II. Comparison of Different Motion Planners for Grasp Execution

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

A. *Motion planning using MoveIt!*

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

OMPL (Open Motion Planning Library) [7] 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) [8] 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) [9] 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) [10] 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!.

Out of these four, OMPL has been chosen to use for performing motion planning in MoveIt!. OMPL gives us a wide variety of choice to solve a motion problem since it contains 23 planners. In the next part we attempt to choose one of these planners by conducting a benchmark.

B. *Comparison of OMPL planners available in MoveIt!*

In order to compare the performance of the 23 motion planners available in MoveIt!, we created two benchmarks shown in Fig.4. The first benchmark resembles a grasp between obstacles, meaning that the planner has to solve a path through a narrow passage. The second benchmark resembles a long motion grasp.

The planners are analyzed by looking at the solved runs, computing time and path length. Solved runs is expressed as a percentage of the amount of runs resulting in a valid path, presented in Fig.5 (a) as a bar plot (high is better). Computing time, the time it takes for the planner to produce a valid path, is presented as a boxplot in Fig.5 (b) (lower is better). Path length, the length of the created path in the configuration space, is presented as a boxplot in Fig.5 (c) (lower is better).

For each planner, 30 runs are executed with a maximum computing time of 3s. In the two defined benchmarks, we find that BiTRRT [11] yields the best performance considering solved runs, computing time and path length. Therefore BiTRRT is chosen to produce paths for the UR5 robot in order to execute the final grasp in this paper.

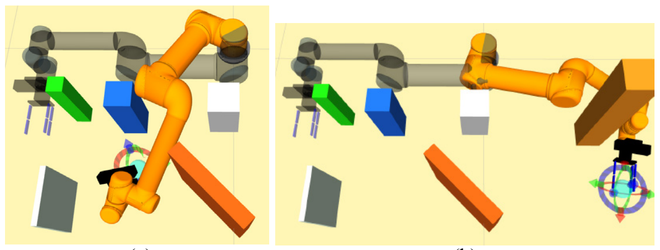


Fig.4. Simulation setting for comparison of different motion planners in MoveIt!. (a) is used to compare the performance under the circumstance of dense obstacles. (b) is used to compare the performance where the robot arm needs long motion path.

III. Details of Fast C-shape Grasping

This section contains a detailed explanation of the fast C-shape grasping algorithm.

A. *Math description of the C-shape*

As mentioned before, in this paper, we specially designed a fast and general grasping algorithm for under-actuated grippers. The under-actuated gripper is simplified as a

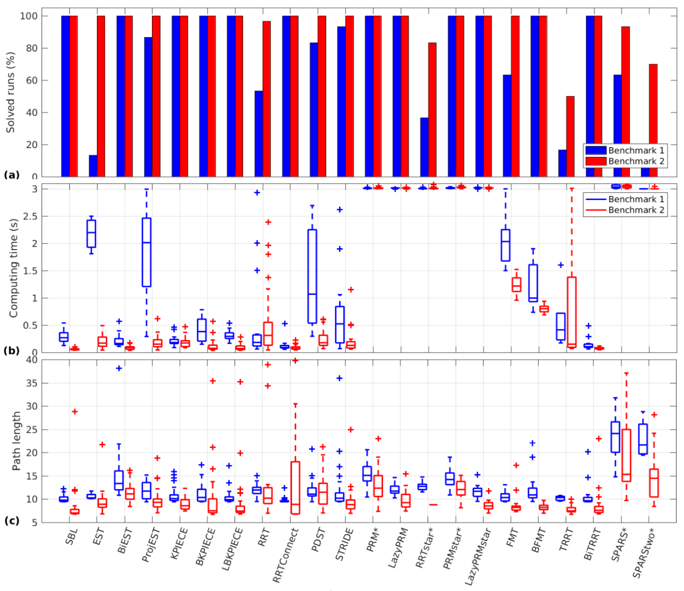


Fig.5. Comparison results of 23 motion planners in MoveIt!.

C-shape. Then, the algorithm will do C-shape searching on the single point cloud of the target object to quickly synthesize an executable grasp.

Fig.6 (a) shows the C-shape of the under-actuated gripper in Fig.3 (c), w is the width of the griper. From Fig.6 (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 (1). C_{red} can be approximated as $C_{red} = \{(-0.5w \le x \le 0.5w) \wedge (-r_1 \le y \le r_1) \wedge (-r_2 \le z \le 0)\}$.

$$C_c = C_{out} - C_{in} - C_{red} \tag{1}$$

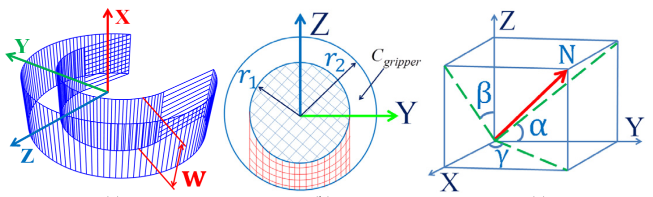


Fig.6. How to obtain the math description of the C-shape.

In order to get the outer cylinder space (C_{out}) and the inner cylinder space (C_{in}), we need to 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 Fig.6 (c). If the normal vector is projected to the XOY plane, XOZ plane and YOZ plane, we can get 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 summarized as equation (2).

$$T=\begin{bmatrix}1&0&0&0\\0&\cos a&\sin a&0\\0&-\sin a&\cos a&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}\cos\beta&0&\sin\beta&0\\0&1&0&0\\\sin\beta&0&\cos\beta&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\x_0&y_0&z_0&1\end{bmatrix}\qquad(2)$$
$$=\begin{bmatrix}\cos\beta&0&-\sin\beta&0\\\sin a\sin\beta&\cos a&\sin a\cos\beta&0\\\cos a\sin\beta&-\sin a&\cos a\cos\beta&0\\x_0&y_0&z_0&1\end{bmatrix}$$

If ($x(t)$, $y(t)$, $z(t)$) are used to stand for an arbitrary points on the arbitrary circle, the parametric equation of the circle can be obtained by the equation (3).

$$\begin{pmatrix}x(t)\\y(t)\\z(t)\\1\end{pmatrix}=\begin{pmatrix}r\cos t\\r\sin t\\0\\1\end{pmatrix}^T*T=\begin{cases}x(t)=x_0+r\cos t\cos\beta+r\sin t\sin a\sin\beta\\y(t)=y_0+r\sin t\cos a\\z(t)=z_0+r\sin t\sin a\cos\beta-r\cos t\sin\beta\end{cases}\qquad(3)$$

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 (4).

$$\begin{cases}x(s,t)=x_0+r\cos t\cos\beta+r\sin t\sin a\sin\beta+s\cos a'\\y(s,t)=y_0+r\sin t\cos a+s\cos\beta'\\z(s,t)=z_0+r\sin t\sin a\cos\beta-r\cos t\sin\beta+s\cos\gamma'\end{cases}\qquad(4)$$

$0\le s\le w$, w is the width of the griper. Using equation (4), we can get equations for C_{out} and C_{in} , then we can obtain the math description of the C-shape using equation (1).

B. *Obtaining the point cloud of the target object*

The raw point cloud from the 3D sensor contains the environment (for example the table plane). In order to quickly isolate 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 (shown as the green points in Fig.7 (b)).

C. *Configuration of the C-shape*

In this subsection, 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.

C.1 How to configure the C-shape efficiently

If we want to locate a C-shape in 3D space, it means many possibilities. How to reduce the possibilities in order to save computing time? 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). Fig.7 shows how to configure the C-shape. (a) shows an example of a C-shape. (b) shows a normal (the blue line). (c) is an enlarged image of (b). 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 (b) and (c). Then the C-shape can only rotate around the normal, so we rotate the C-shape around the normal with an incremental angle δ (shown as Fig.7 (c)). Every red line in (b) and (c) means 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. Fig.7 (d) shows an example of a potential grasp candidate corresponding to the black axis in (c). The red points in (d) mean the points of the object covered by the C-shape.

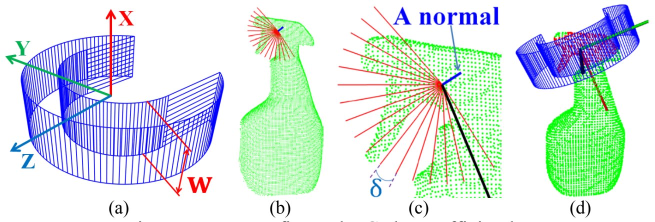


Fig.7. How to configure the C-shape efficiently.

C.2 How to deal with the unseen part

If the C-shape is configured as Fig.8 (a), then the gripper will collide with the target object. Because we only use a single-view partial point cloud of the object in this paper, the unseen part of the target object will result in grasp uncertainty. Here, we propose to employ the boundary of the object to eliminate the uncertainty. Specifically, the point cloud in the camera coordinate system is used to work out the boundary points Ω_b (shown as Fig.8 (b)). Fig.8 (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 coordinate system and the two red points. The two orange dashed lines are obtained by extending the two orange lines. This method will go through all the points on the boundary, and then we can obtain a point cloud shown as Fig.8 (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 (5).

$$C_{obj}=C_{obj}^{'}+C_{unseen}\qquad(5)$$

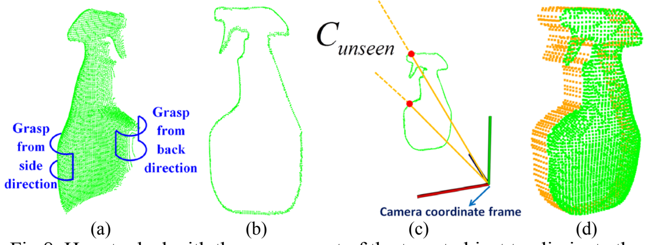


Fig.8. How to deal with the unseen part of the target object to eliminate the grasp uncertainty.

D. *Generation and down-sampling of normals*

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

D.1 Generation of normals

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:

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

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.

But one normal has two possible directions (the red and green arrow lines) shown as Fig.9 (a), how to determine 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, they need to satisfy the equation: $\vec{n}_i \cdot V_i > 0$. Using this equation can constrain all the normals towards the camera viewpoint to obtain all normals (shown as all the red lines in Fig.9 (a)) of the object.

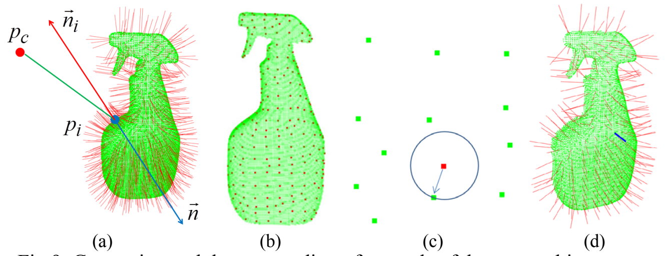


Fig.9. Generation and down-sampling of normals of the target object.

D.2 Down-sampling of normals

Normals in Fig.9 (a) are pretty dense. In order to accelerate the speed of grasp searching, the normals need to be down-sampled. K-d tree is used to down-sample the normals.

The green points in Fig.9 (b) 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 Fig.9 (b)). 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 Fig.9 (c)). Then the corresponding normal (n_i) of P_i can be looked up in the dense normals obtained in section of *D.1*. All the corresponding noramls are put together to get the down-sampled normals shown as Fig.9 (d).

E. Determination of the first axis of the C-shape

As mentioned in section of C.1, the C-shape axis is allocated around the normal with an incremental angle δ. Then a question comes out, that is, how to decide the first axis of the C-shape to increase the chance to find a suitable grasp?

If δ is a big angle, for example 60^o in Fig.10 (a) and (b), then we may get two totally different allocations of C-shape axis. In Fig.10 (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 Fig.10 (b) corresponds to a very good grasp candidate (shown as Fig.10 (c)). The difference is generated because of the position of the first axis. In this paper, we propose to use the principal axis of the local point cloud to work as the first C-shape axis.

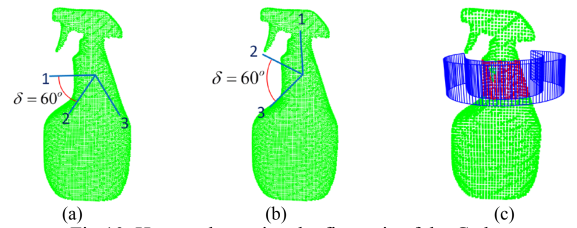


Fig.10. How to determine the first axis of the C-shape.

F. Determination of the center point of the C-shape

As we mentioned in section of C.1, the under-actuated gripper will approach the object along the normal direction. Then a question comes out, that is, where to stop?

Fig.11 is used to explain how to determine the center point of the C-shape. Fig.11 (a) shows a possible grasp candidate, the green points stand for the points covered by the C-shape. Fig.11 (b) is the abstracted point cloud, and the red arrow stands for the approaching direction of the C-shape. The two red points in Fig.11 (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) are used to explain 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)). And then the convex hull of the projected point cloud is extracted shown as the green points in (c). The green point in Fig.11 (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 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)). The maximal Z_c means the earliest contact point with the object when the C-shape tries to approach the object.

$$Z_{c\_max} = \max(Z_{c1}, Z_{c2}, \cdots, Z_{cn}) \tag{6}$$

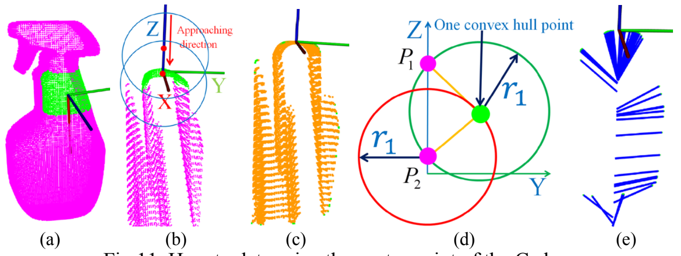


Fig.11. How to determine the center point of the C-shape.

G. Collision judgement of the C-shape

Fig.12 (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 object or not? If the C-shape will not collide with object, then it means this configuration is possible to be an executable grasp candidate, otherwise this configuration should be ignored.

In order to judge whether one figuration 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 Fig.12 (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 (the math description of the C-shape is obtained in section A), that means the C-shape will collide with the object, then the grasp candidate g_i should be removed, otherwise g_i is reserved for following analysis. Using this method goes through all the C-shape configurations, we can get a vector $G=(g_1,g_2...g_n)$ which is used to store all grasp candidates without collision with the object.

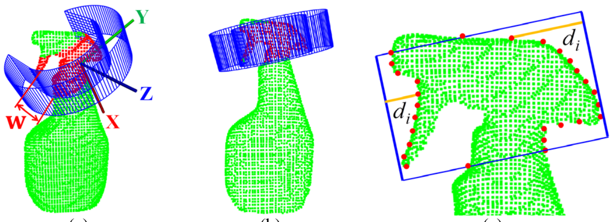


Fig.12. How to judge one grasp formed by a C-shape.

H. 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 subsection, we will consider the local geometry of the points enveloped by the C-shape. Specifically, a grasp candidate is shown as Fig.12 (b), the local geometry shape will lead to 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.

I. Force balance optimization

All grasp candidates passed step G and step H form a new vector $G_j=(g_{j1},g_{j2}\cdots g_{jn})$, all the grasps in this vector can be executed without collision with the object. If the lines 1,2,3,4,5, 6 and 7 in Fig.13 (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 papers will employ the physic 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 paper are unknown. We propose to use the local geometry shape to do force balance computation. The blue points in Fig.13 (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 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 $G_j=(g_{j1},g_{j2}...g_{j7})$. Fig.13 (f) is a line graph of the vector ψ, the grasp with the largest ψ is chosen as the final grasp. Fig.13 (g) shows the best grasp returned, which corresponds to the 4th grasp in (a) and (f).

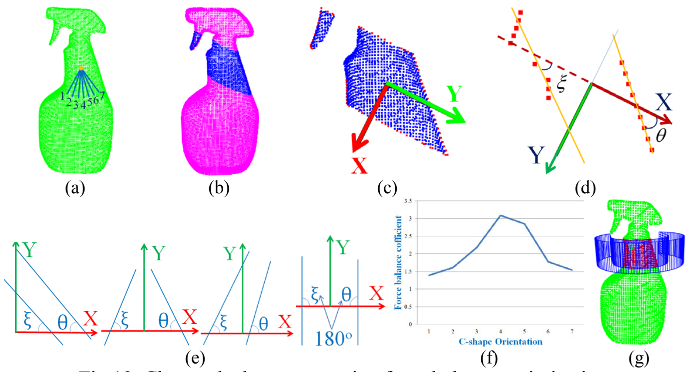


Fig.13. Choose the best grasp using force balance optimization.

The above steps from subsection of C to I 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.

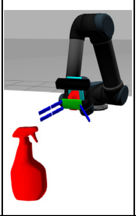
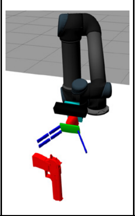
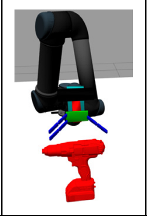
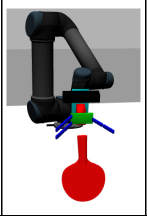
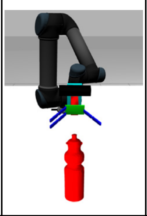
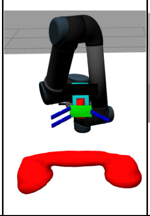
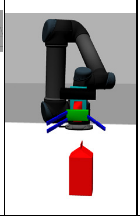
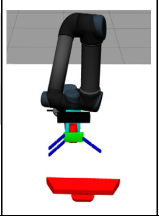
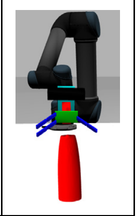
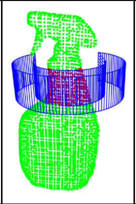
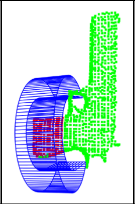
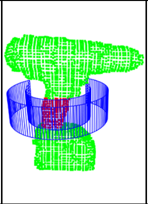
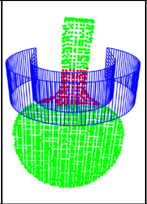
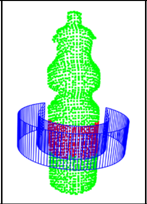
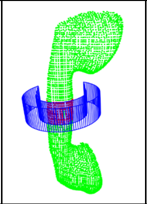
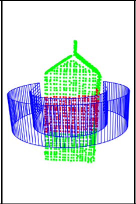
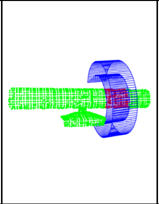
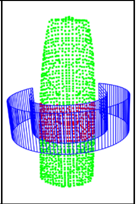
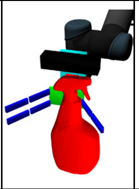
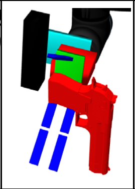
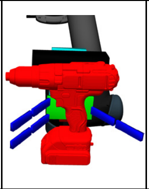
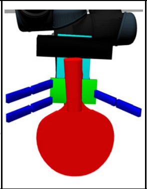
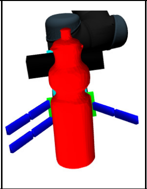
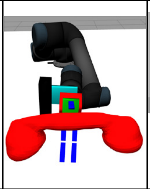
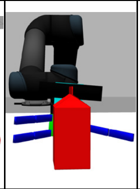
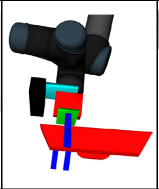
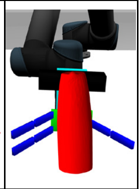
IV. Simulation

In order to verify our grasping algorithm, simulations are performed using a personal computer (2 cores, 2.9GHz). Several objects with different geometry shapes are used in the simulation. All the tested objects can be seen in the second row of table III. 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.

V. 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 shown as Fig.14. The results of experiments are shown as table IV. The second row shows the experiment setup for every object. The third row shows the example grasp found by the grasping algorithm. The fourth

Table III: Simulation results

Object name	Cleaner spray bottle	Pistol	Electric drill	Table tennis racket	Water bottle	Telephone horn	Milk carton	Kinet	Shampoo bottle
Intial setup									
Example grasp foud									
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

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 IV 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 IV, 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 in table I, our algorithm shows much improvement at the speed of grasp searching.

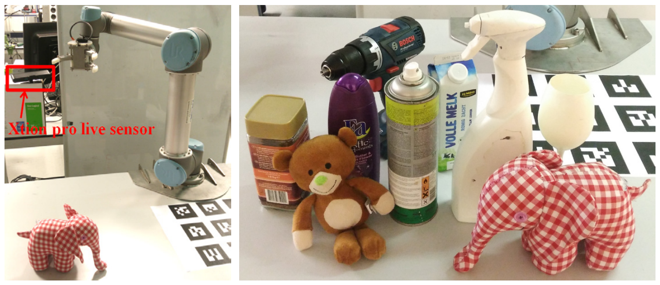


Fig. 14. Experiments setup and objects used for experiments.

VI. Discussion

In this section, we will discuss the characteristics of our grasping algorithm compared with the grasping algorithms in table I.

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 [1,3,4,5]. Meanwhile, the price of the under-actuated gripper is much cheaper than [2] which uses a barrett hand.

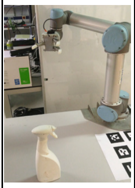
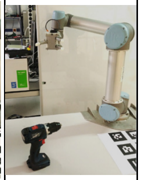
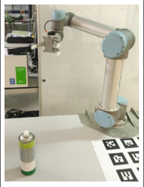
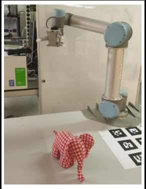
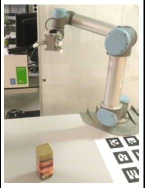
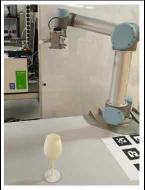
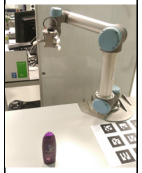
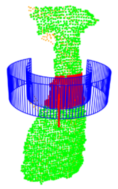
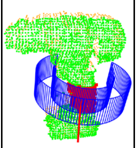
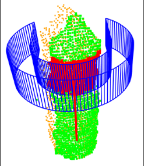
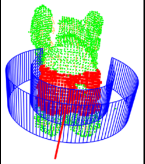
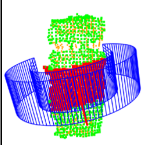
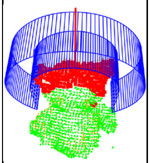
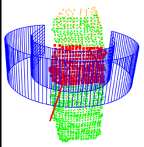
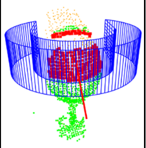
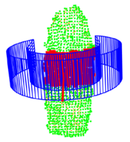
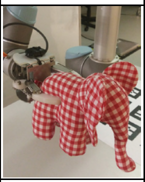
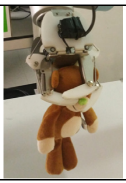
Object complexity: The presented grasping approach is able to find grasp for complex objects like, teddy bear, elephant, electric drill and the cleaner spray bottle. This makes it better than [1,3,5], which only considers simple objects. [2] transforms the objects into simple shapes (cylinder, disk, 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 [2,3,4,5]. [1] is able to find a grasp faster since it only uses a RGB image at the cost of losing depth information of the object.

Grasping direction: [1,3,5] 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 [4] 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.

Table IV: Experiment results

Object name	Cleaner spray bottle	Electric drill	Spray can	Elephant	Coffee jar	Teddy bear	Milk carton	Wineglass	Shampoo bottle
Initail setup									
Example grasp foud									
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

VII. Conclusion

In this paper, 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 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 object to enhance the grasp security. For the robot arm to quickly execute the grasp found by the grasping algorithm, a suitable motion planner has to be selected. We made comparison of the motion planners available in MoveIt!. The motion planner, BiTRRT, is chosen for motion planning due to its high solved runs, low computing time and short path length. In order to verify the effectiveness of our algorithm, several objects commonly used by other grasping algorithms with different geometric shapes were used to do simulations and experiments. And successful results are obtained.

Acknowledgment

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.

References

[1] Johannes Baumgartl and DominikHenrich, "Fast Vision-based Grasp and Delivery Planning for unknown Objects," in proceeding of 7th German Conference on Robotics (ROBOTIK 2012), pages 1–5, 2012.

[2] Clemens Eppner and Oliver Brock, "Grasping unknown objects by exploiting shape adaptability and environmental constraints," in proceeding of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 4000–4006, 2013.

[3] 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 proceeding of IEEE Conference on Automation Science and Engineering (CASE), pages 1258–1263, 2014.

[4] Andreas ten Pas and Robert Platt. Using Geometry to Detect Grasps in 3D Point Clouds," in proceeding of International Syposium on Robotics Research (*ISRR)*, pages 1–16, 2015.

[5] Toshitaka Suzuki, Tetsushi Oka, "Grasping of unknown objects on a planar surface using a single depth image. In AIM, pages 572–577, 2016.

[6] A. Sucan and S. Chitta, "MoveIt!" , 2013.

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

[8] M. Kalakrishnan, S. Chitta, E. Theodorou, Peter Pastor, and Stefan Schaal, "STOMP:Stochastic trajectory optimization for motion planning," in proceeding of IEEE International Conference on Robotics and Automation (ICRA), pages 4569–4574, 2011.

[9] 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, May, 2013.

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

[11] D. Devaurs, T. Sim´eon, and J. Cort´es, "Enhancing the transitionbased rrt to deal with complex cost spaces," in proceeding of IEEE International Conference on Robotics and Automation (ICRA), pages 4105–4110, 2013.