A novel algorithm for fast grasping of unknown objects using C-shape configuration

Lei, Qujiang; Chen, Guangming; Meijer, Jonathan; Wisse, Martijn

DOI
10.1063/1.5006570
Publication date
2018
Document Version
Final published version
Published in
AIP Advances

A novel algorithm for fast grasping of unknown objects using C-shape configuration

Qujiang Lei,[a] Guangming Chen, Jonathan Meijer, and Martijn Wisse
Faculty of Mechanical, Maritime and Materials Engineering, Delft University of Technology, 2628CD Delft, Netherlands

(Received 26 September 2017; accepted 30 January 2018; published online 8 February 2018)

Increasing grasping efficiency is very important for the robots to grasp unknown objects especially subjected to unfamiliar environments. To achieve this, a new algorithm is proposed based on the 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 unknown object is constructed by using a single-view partial point cloud. To examine 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 this 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. © *2018 Author(s). All article content, except where otherwise noted, is licensed under a Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).* https://doi.org/10.1063/1.5006570

I. INTRODUCTION

An unknown object can be defined as an item that has neither apparent information nor geometric model.[1] Fast grasping of unknown objects is quite important for robots efficiently to 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 for fasting grasping of unknown objects.

A literature study reports five dominant fast grasping algorithms.[2–6] Among them Ref. 2 is a well acknowledged fast grasping algorithm using Hough transformation to visualize the edges of objects into a 2D image. It can detect whether the edges have sufficient length and whether the parallel edges suit the width of the used grippers. In the work of Eppner and Brock,[3] the point cloud is transformed into shape primitives (i.e., cylinder, disk, sphere and box). A pre-grasp (configuration of the hand) is chosen according to shape primitives. Using the shape primitive, the scope of grasp searching is significantly reduced. However, this may result in lots of grasp uncertainty, which may lead to grasp failure.

Reference 4 applies the contact area of the grasping rectangle to determine the suitable grasps. The problem with this algorithm is that when the contact area is too small, the grasp is likely to fail, and thus has to be replaced by another one. Reference 5 utilizes principal axis and centroid of the object to synthesize a grasping action. Pas[6] attempted to fit the shape of the parallel gripper on the point cloud of the objects. They use a detailed segmentation to pick objects from dense clutters. This algorithm promotes quite efficient grasping action. These three fast grasping algorithms have a common character of using the normal of the table plane as the grasp approaching direction, which can

[a]E-mail address: q.lei@tudelft.nl

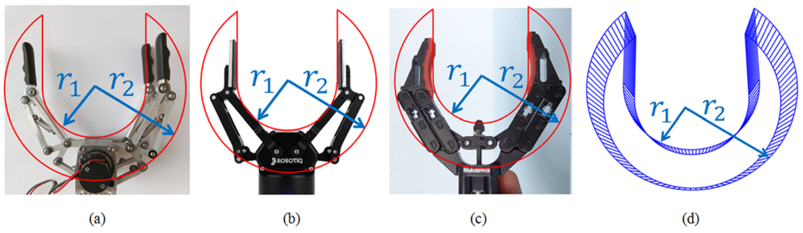


FIG. 1. Three widely used commercial under-actuated grippers (a-c) and the approximation of C-shape (d).

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.

As a summary, it is obtained that except Ref. 3, the other four fast grasping algorithms[2,4–6] are designed for parallel grippers. Furthermore, excluding Ref. 2 that uses RGB images as input of the grasping algorithm, the rest four grasping algorithms use a partial point cloud as input, which can reduce the computational time during a grasping process. In addition, four of them[2,4–6] use parallel grippers because parallel grippers have simpler geometry shape and are easier to control in comparison with dexterous hands. Nevertheless, the parallel grippers are not advocated in terms of flexibility. To make a trade-off, the under-actuated grippers are selected which have both sufficient flexibility and operational convenience. Fig. 1(a–c) shows three commonly used under-actuated grippers. All the three types of under-actuated grippers can be described as a C-shape with radii r_1 and r_2 as particularly displayed in Fig. 1(d).

Additionally, due to the fact that few grasping algorithms provided details on the actual motion planning of the robotic arm towards the unknown objects, grasping algorithms seem simply focusing on finding specific grasps on certain objects. As a result, researchers and users who plan to implement the grasping algorithms have to firstly bridge the gap of motion planning. Hence, they are required to investigate many different available motion planning methods before implementation, which introduces much time. The visual application of MoveIt!,[7] a motion planning interface in ROS, is convenient to operate and therefore widely used for robot manipulation. In order to enable researchers and users to quickly choose suitable motion planners, a comparison of motion planners available in the simulation system (Moveit!) is made. To execute grasps, the motion planner with the highest number of solved runs, lowest computing time and shortest path length found our grasping algorithm is used.

The goal of this paper is to design a fast and general grasping algorithm for unknown objects. In order to achieve this goal, the rest of this paper is organized as follows: Section II illustrates our fast grasping algorithm. Section III compares different online motion planners. Section IV presents the simulation results. Section V gives the experimental results. Finally, the conclusions are provided in section VI.

II. A NOVEL ALGORITHM

In order to understand this novel algorithm, this section firstly presents the mathematical description of the C-shape configuration. Next, the outline of our fast grasping algorithm is presented. It demonstrates that eight steps are required to execute an effective grasp. Using an example of grasping an unknown object, these eight steps are interpreted in details.

A. Mathematical description of the C-shape configuration

Fig. 2(a) shows the C-shape of the under-actuated grippers, in which w is the width of the griper. Fig. 2(b) demonstrates the area of the C-shape (C_c) from the perspective of x axis. Therefore, C_c equals the outer cylinder space (C_{out}) minuses the inner cylinder space (C_{in}) and minuses the red area (C_{red}) (equation (1)). C_{red} can be approximated as $C_{red} = \{(-0.5w \le x \le 0.5w) \wedge (-r_1 \le y \le r_1) \wedge (-r_2 \le z \le 0)\}$.

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

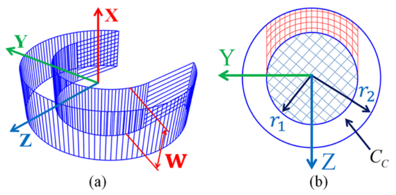


FIG. 2. Mathematical description of the C-shape (a) 3D illustration (b) YOZ projection of the C-shape.

In order to calculate C_{out} and C_{in} in the 3D space, the parametric equation for a random circle on an arbitrary plane must be firstly obtained. Assuming that P (x_0, y_0, z_0) is the center and r is the radius of an arbitrary circle, the unit normal vector is given by $N=(n_x, n_y, n_z)$ shown as the red line in Fig. 3. When the normal vector is projected to the XOY plane, XOZ plane and YOZ plane, the three project lines are obtained, i.e., the three green lines. γ, β and α are respectively the angles between the projected lines and the coordinate axes. Then the arbitrary plane can be obtained by transforming the XOY plane through this manner: first, rotating around the X axis by α; then, rotating around the Y axis by β; last, moving along the vector N to P (x_0, y_0, z_0). The entire transformation is expressed as equation (2).

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

Assuming that $(x(t), y(t), z(t))$ are the arbitrary points on the arbitrary circle, then the parametric equation of the circle can be obtained by the equation (3).

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

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)

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

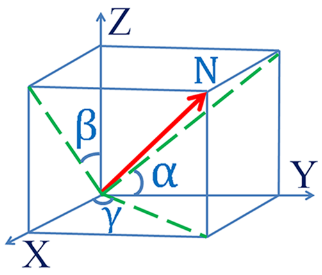


FIG. 3. Parametric definition for the space calculation of the C-shape.

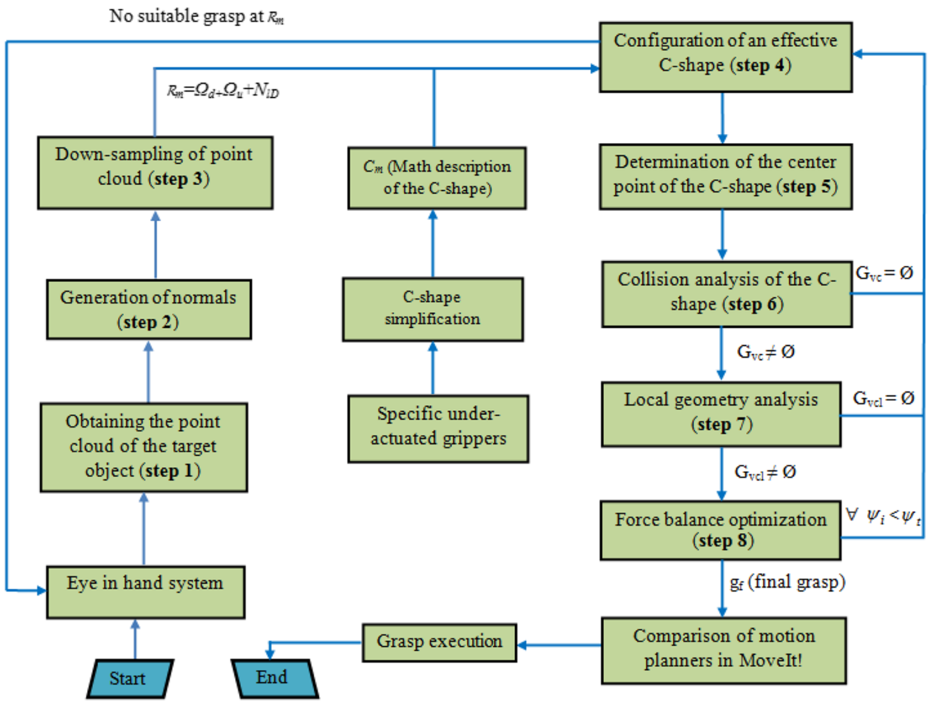

No suitable grasp at R_m
$R_m=\Omega_d+\Omega_u+N_{ID}$
Configuration of an effective C-shape (**step 4**)
Down-sampling of point cloud (**step 3**)
C_m (Math description of the C-shape)
Determination of the center point of the C-shape (**step 5**)
Generation of normals (**step 2**)
C-shape simplification
Collision analysis of the C-shape (**step 6**)
$G_{vc}=\emptyset$
$G_{vc}\neq\emptyset$
Obtaining the point cloud of the target object (**step 1**)
Specific under-actuated grippers
Local geometry analysis (**step 7**)
$G_{vcl}=\emptyset$
$G_{vcl}\neq\emptyset$
Force balance optimization (**step 8**)
$\forall\ \psi_i<\psi_t$
g_f (final grasp)
Eye in hand system
Grasp execution
Comparison of motion planners in MoveIt!
Start
End

FIG. 4. The outline of our algorithm for fast grasping of unknown objects.

In which $0 \leq s \leq w$, w is the width of the griper. Using equation (4), we can obtain equations for C_{out} and C_{in}. In combination with C_{red} and using equation (1), the C-shape configuration can be mathematically expressed.

B. Outline of our fast grasping algorithm

The outline of our fast grasping algorithm is shown in Fig. 4. It can be seen that eight steps are required to execute fast grasping using our algorithm. In the following paragraphs, these eight steps are respectively explained in detail.

1. *Step 1: Obtaining the point cloud of the target object*

Fig. 5(a) shows the experimental setup, which consists 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

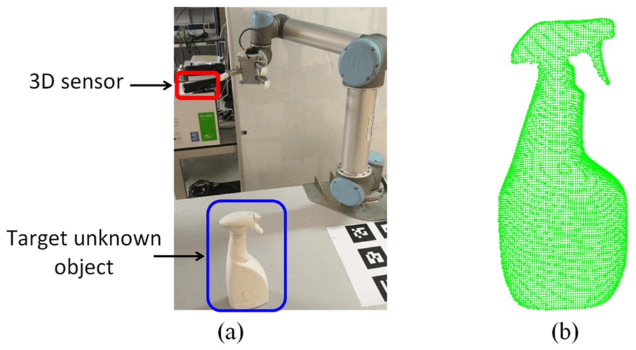


FIG. 5. Obtaining the point cloud of the target object (a) experimental setup (b) raw point cloud.

reduce the computing time and remove the points out of the reach of the robot arm. To isolated point cloud of the target object, the Random Sample Consensus (RANSAC) method is applied to remove the table plane.

2. 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 paper, normals are used to guide the configuration of the C-shape to accelerate grasp searching. The solution for estimating the surface normal is to analyze 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:

$$C = \frac{1}{k}\sum_{i=1}^{k}(P_i - \bar{P})\cdot(P_i - \bar{P})^T \; C\cdot\vec{V}_j = \lambda_j \cdot \vec{V}_j \quad j \in \{0,1,2\} \tag{5}$$

Where k is the number of points in the neighborhood of P_i, $\bar{P}$ 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 Fig. 6, 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 can constrain all the normals towards the camera viewpoint to obtain all normals (shown as all the red lines in Fig. 6) of the target object.

3. Step 3: Down-sampling of point cloud

As can be observed in Fig. 6, the normals of the target object are too dense. In order to accelerate the speed of grasp searching, the normals must be down-sampled. To down-sample the normals, the K-d tree is used. Fig. 7(a) shows the green points of the original point cloud (Ω) that is used to compute the normal, and the red points are the down-sampled point cloud Ω_d. 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. 7(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 combined to form the down-sampled normals shown as Fig. 7(c).

4. Step 4: Effective configuration of a C-shape

Configuration of the C-shape is abided by a SE (3) group, thus it can mean 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). In this step, we will explain how to configure the C-shape to find a

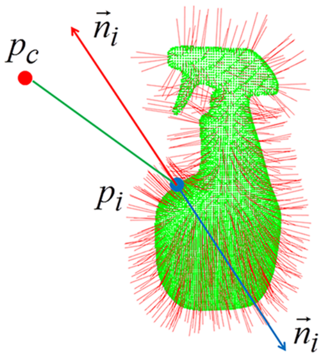


FIG. 6. Normals of the target object.

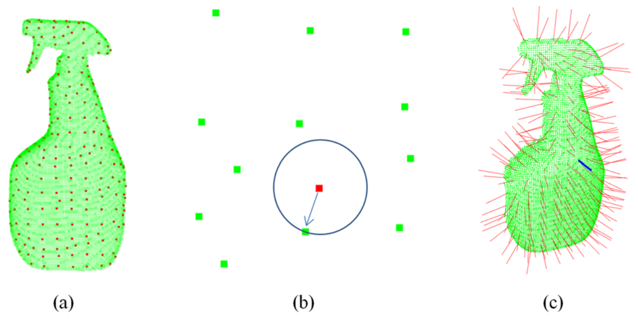


FIG. 7. Down-sampling of normals of the target object. (a) original point cloud (b) KNN search (c) reduced normals.

suitable grasp. Besides, we will also illustrate the method to tackle the unseen part of object because we cannot see the back side of the object when we use a single-view point cloud.

Fig. 8 shows the procedures to configure the C-shape. The blue line in Fig. 8(a) is a random normal. Fig. 8(b) is an partially enlarged image of Fig. 8(a). When 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 Fig. 8(a) and (b). Then the C-shape can solely rotate around the normal, thus that the C-shape are configured around the normal with an incremental angle δ (shown as Fig. 8(b)). Each red line in Fig. 8(a) and (b) 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. 8(c) illustrates an example of a potential grasp candidate corresponding to the black axis in Fig. 8(b), in which the red points in Fig. 8(c) represent the points of the object covered by 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, we may get two totally different allocations of C-shape axis shown in Fig. 9(a) and (b). The three cylinder axis in Fig. 9(a) will lead to no appropriate grasp, because all the three C-shapes will collide with the object. However, the C-shape axis in Fig. 9(b) can promote a suitable grasp candidate (shown as Fig. 9(c)). This difference is ascribed to the position of the first axis. Hereby it is suggested to use the principal axis of the local point cloud to work as the first C-shape axis.

Nevertheless, using a single-view partial point cloud of the object can also result in grasp uncertainty. For instance, if the C-shape is configured as Fig. 10(a), the gripper will collide with the target object. To overcome this, the boundary of the object is employed to eliminate the uncertainty. Specifically, the point cloud in the camera frame is used to work out the boundary points Ω_b (shown as Fig. 10(b)). Fig. 10(c) illustrates our method 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 with the two red points. These two orange dashed lines are obtained by extending the two orange lines.

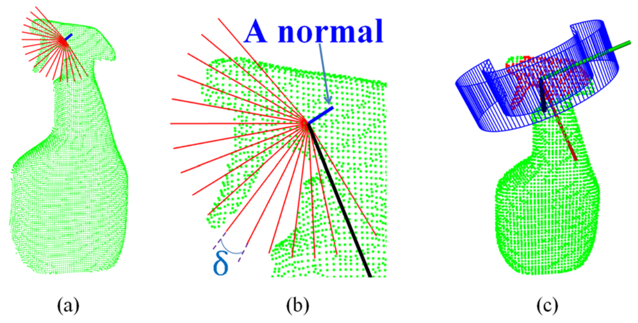


FIG. 8. Configuration of the C-shape (a) selection of a random normal (b) identification of the normal (c) a potential grasp candidate.

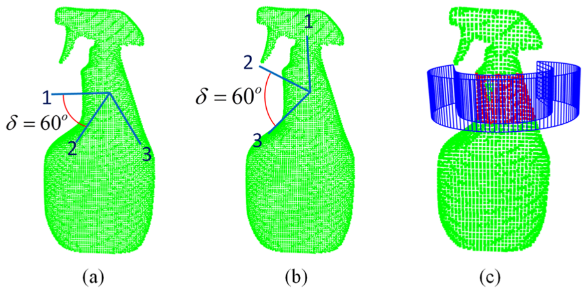


FIG. 9. Determination of the first configuration of the C-shape (a) and (b) illustrate two different allocations from one big angle (c) a suitable grasp.

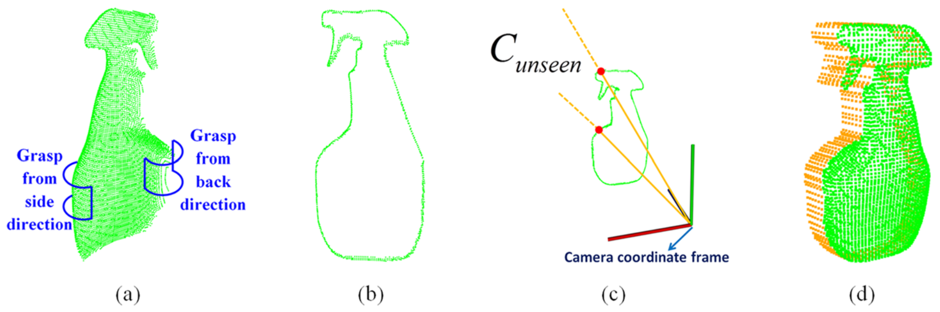


FIG. 10. Illustration of the solution to deal with the unseen part of the target object (a) grasp collision with the object (b) the boundary points (c) illustration on dealing with the unseen part (d) a point cloud which includes the unseen part.

When applying this method to all the points on the boundary, we can obtain a point cloud shown as Fig. 10(d). After the unseen part is generated, the configuration space (C space) of the target object (C_{obj}) is divided into two parts, which are C'_{obj} (the green points in (d)) and C_{unseen} (the orange points in Fig. 10(d)), as shown in equation (6).

$$C_{obj} = C'_{obj} + C_{unseen} \tag{6}$$

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

As mentioned in the previous step, the under-actuated gripper will approach the target object along the normal direction. Then the correct contact position for initiating a grasp must be determined. Therefore it is required 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

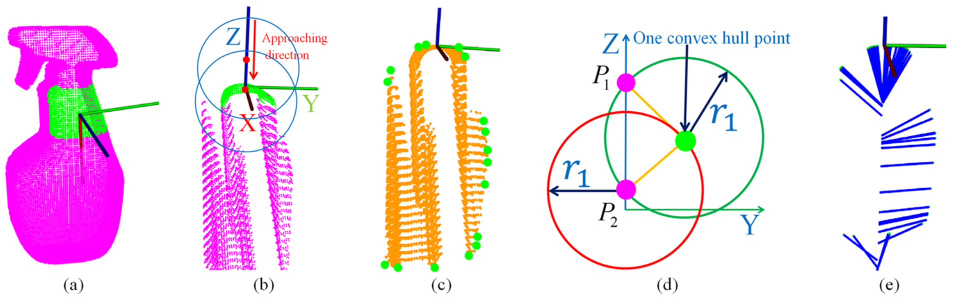


FIG. 11. Determination of the center point of the C-shape. (a) a possible grasp candidate (b) abstracted point cloud (c) convex hull of the projected point cloud (d) one obtained point of the convex hull (e) all obtained best center points.

the corresponding C-shape. Due to the fact that the center point can go down, meaning that the two examples of center points of the C-shape are not useful. Fig. 11(c), (d) and (e) elaborate on the method to find out the best center point, which is given below.

When the abstracted point cloud in Fig. 11(b) is projected to the YOZ plane to get the projected point cloud (orange points shown as (c)), the convex hull of the projected point cloud is extracted shown as the green points in (c). The green point in Fig. 11(d) gives one point of the convex hull obtained in (c). When drawing 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. In the same way, we can get all the center points $Z_c = (Z_{c1},Z_{c2},\cdots,Z_{cn})$ (shown as (e)) for all the green points in (c). The maximal Z_c is used as the final C-shape center (shown as the equation (7)). The maximal Z_c means the earliest contact point with the object when the C-shape tries to approach the object.

$$Z_{c\_\max} = \max(Z_{c1}, Z_{c2}, \cdot\cdot\cdot, Z_{cn}) \tag{7}$$

6. Step 6: Collision analysis of the C-shape

After the configuration of the C-shape is obtained, it is necessary to predict whether this configuration will collide with object. 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 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 Fig. 12, 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.

7. Step 7: Local geometry analysis

Step 6 ensures that the C-shape will not collide with the object, which also means that the C-shape can envelope the object at this configuration. In this step, we will account for the local geometry of the points enveloped by the C-shape. Fig. 13(a) shows a grasp candidate, in which the local geometry shape may lead to grasp uncertainty. To tackle this scenario, two grasp sides are abstracted shown as the red points in Fig. 13(b). The distance between one red point and the blue line is defined as d_i ($0 < i \leq n$, and n is the total number of the red points). Thus a summation of the distances can be used to evaluate the variance v of the grasp ($v = \sum_{i=1}^{i=n} d_i$). For the variance is smaller than the threshold set by the system the grasp is saved. Otherwise, it is removed.

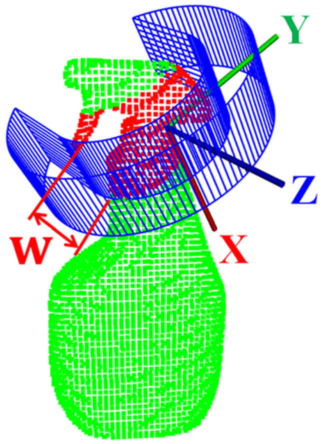


FIG. 12. Analyzing one grasp formed by a C-shape.

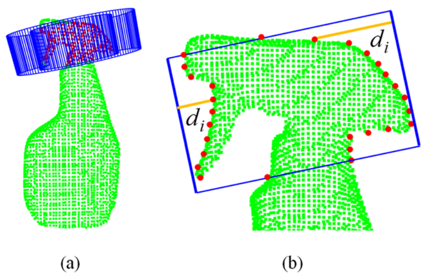


FIG. 13. Local geometry analysis (a) a grasp candidate (b) two abstracted grasp sides.

8. Step 8: Force balance optimization

All grasp candidates passed previous steps can form a new vector $G_j = (g_{j1}, g_{j2}...g_{jn})$, in which all the grasps can be executed without collision with the target object. Generally, researchers employ the physic property to do force balance computation, for instance, the friction coefficient. However, it is inapplicable in this research because the physic property is unknown. To choose the final grasp, we propose to use force balance optimization based on the local geometry shape.

Fig. 14(a) shows seven lines which stand for the C-shape axis of the grasps in vector G_j. It is inferred that all the grasps from g_{j1} to g_{j7} can be executed. The blue points in Fig. 14(b) stand for the grasp candidate 1 (g_{j1}). It is projected to the XOY plane to get the projected point cloud shown as Fig. 14(c). The two grasp sides are abstracted to shown as the red points in Fig. 14(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. Fig. 14(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. 14(f) displays a line graph of the vector ψ, the grasp with the largest ψ is chosen as the final grasp. Fig. 14(g) shows the returned best grasp, which corresponds to the 4th grasp in Fig. 14(a) and (f).

The above steps 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. This was also identified in the outline of the algorithm in Figure 4.

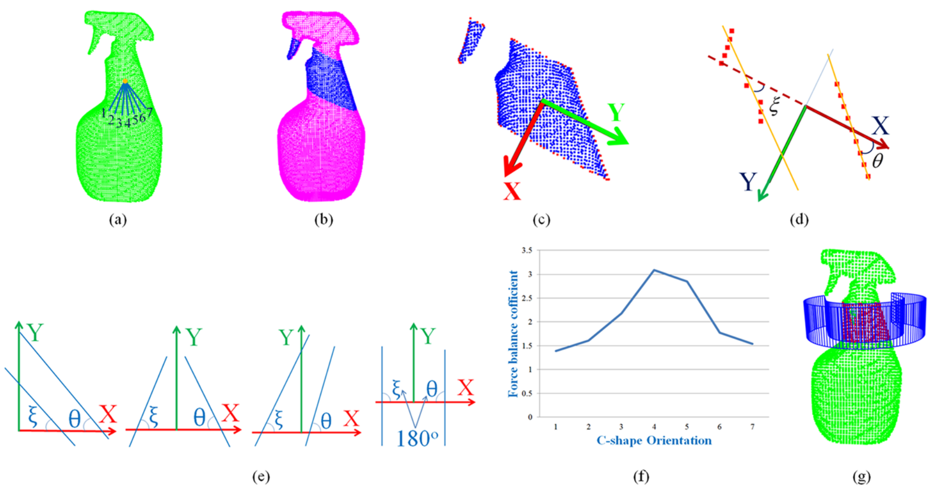


FIG. 14. Choose the best grasp using force balance optimization. (a) C-shape axis of the grasps (b) grasp candidate 1 (c) projected point cloud (d) the abstracted two grasp sides (e) three cases of allocation (f) a line graph (g) the returned best grasp.

III. SELECTION OF MOTION PLANNERS

Performance of motion planning depends on the chosen motion planner so that motion planning is a very important part for grasp execution. In this section, we will investigate the choice of motion planner based on the results of benchmark after a series of motion planner comparisons and parametric selections in MoveIt!.

A. Motion planning using MoveIt!

MoveIt! itself does not provide motion planning. Instead, it is designed to work with planners or planning libraries. Currently four main planners/planning libraries can be configured to use: namely, OMPL (Open Motion Planning Library),[8] STOMP (Stochastic Trajectory Optimization for Motion Planning),[9] CHOMP (Covariant Hamiltonian Optimization for Motion Planning)[10] and SBPL (Search-Based Planning Library).[11]

OMPL[8] 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[9] 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[10] mainly operates by using two terms, which are dynamical quantity term and obstacle term. 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!. SBPL[11] 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 the discussed four planning libraries, OMPL will be used to perform motion planning in MoveIt! to compare different motion planners. This planning library also gives a wide variety of choices to solve a motion planning problem because it contains 23 planners.

B. Overview of OMPL planners available in MoveIt!

Table I lists the 23 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. Sampling-based motion 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.[20] The 23 sampling-based motion planners can be divided into two categories, i.e. none-asymptotically optimal planners and asymptotically optimal planners. None-asymptotically optimal planners include SBL,[12] EST,[13] BiEST based on Ref. 13, ProjEST based on Ref. 13, KPIECE,[14] BKPIECE based on Ref. 14, LBKPIECE based on Refs. 14 and 15, RRT,[16] RRTConnect,[17] PDST,[18] STRIDE,[19] PRM[20] and LazyPRM.[15] Asymptotically optimal planners contain RRTstar,[21] PRMstar based on Refs. 20 and 21, LazyPRMstar based on Refs. 15 and 21, FMT,[22] BFMT,[23] LBTRRT,[24] TRRT,[25] BiTRRT,[26] SPARS[27] and SPARStwo.[28]

C. Methodologies of comparing motion planners in MoveIt!

To compare the performance of the 23 motion planners available in MoveIt!, we created two benchmarks shown in Fig. 15. The first benchmark resembles a grasp among dense obstacles and the second resembles a long motion grasp. 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 are 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. Finally, planners with a small standard deviation from the average path length and small interquartile range are considered as better performance.

TABLE I. Available planners of OMPL in MoveIt!.

Planner name	Reference	Asymptotically optimal	Time-invariant goal
SBL	12		√
EST	13		√
BiEST	Based on Ref. 13		√
ProjEST	Based on Ref. 13		√
KPIECE	14		√
BKPIECE	Based on Ref. 14		√
LBKPIECE	Based on Refs. 14 and 15		√
RRT	16		√
RRTConnect	17		√
PDST	18		√
STRIDE	19		√
PRM	20		
LazyPRM	15		
RRTstar	21	√	
PRMstar	Based on Refs. 20 and 21	√	
LazyPRMstar	Based on Refs. 15 and 21	√	
FMT	22	√	√
BFMT	23	√	√
LBTRRT	24	√	√
TRRT	25	√	√
BiTRRT	26	√	√
SPARS	27	√	
SPARStwo	28	√	

The benchmarking experiments are performed using one thread on a system with an Intel i5 2.70GHz processor and 8GB of memory. 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.

D. Parameter selection

One important parameter can affect all planners, which is the distance parameter (longest_valid_segment_fraction). This parameter is accounted when the planner detects collisions between two nodes. Collision detection is not detected when the distance between the nodes is smaller than 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 the value of this parameter did not play an immediate effect on the solved runs for both benchmark problems in Fig. 15. Among the 23 available planners in MoveIt!, 19 of them have their own parameters. For the two benchmarks, parameters of the 19 planners were

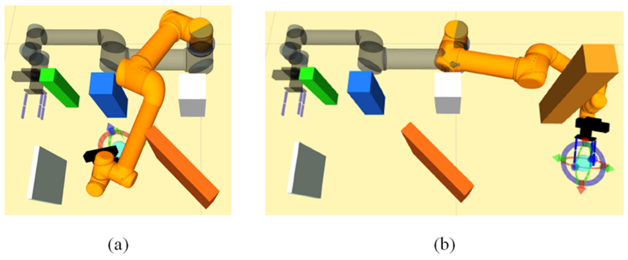


FIG. 15. Simulation setting for comparison of different motion planners in MoveIt!. (a): Benchmark 1: Grasp among dense obstacles. (b): Benchmark 2: The robot arm needs long motion path for grasping.

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			

set to values that benefit one or more performance measures, and the corresponding values are given in Table II.

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. As a result, we are unable to provide benchmark data for this particular planner. There is an important parameter for the used UR5 robot, i.e., joint limit settings for each joint. The parameter can be set as π or 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.

E. Comparison results

Results of benchmark 1 are shown in Fig. 16 and Table III. 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 one second. 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.

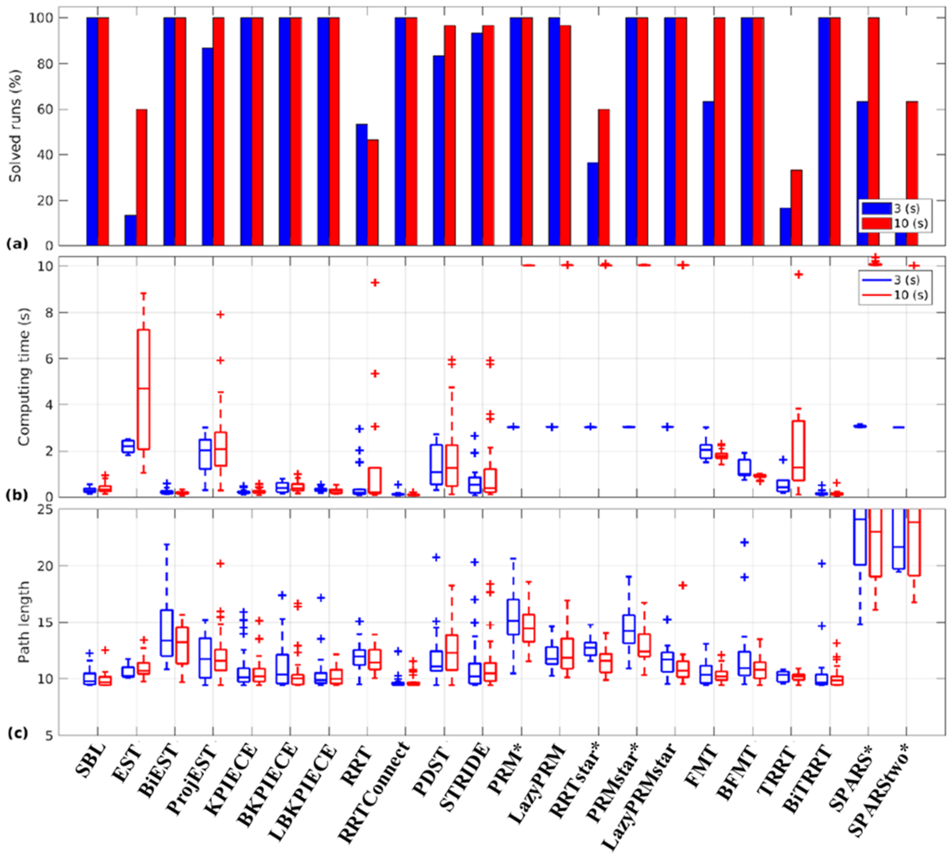


FIG. 16. Comparison results of 23 motion planners in MoveIt! for benchmark 1.

TABLE III. Average values for benchmark 1.

	Max. 3s computing time		Max. 10s computing time	
Planner name	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)

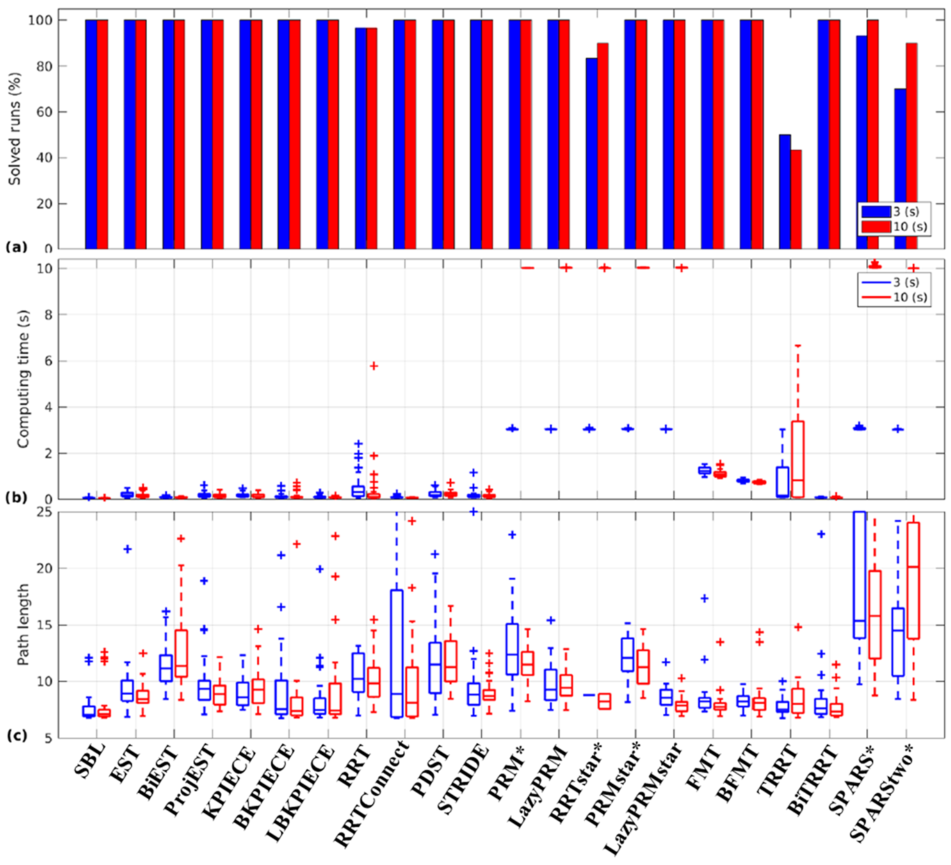


FIG. 17. Comparisotan results of 23 motion planners in MoveIt! for benchmark 2.

TABLE IV. Average values for benchmark 2.

	Max. 3s computing time		Max. 10s computing time	
Planner name	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)

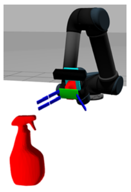
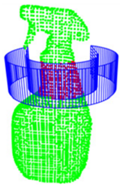
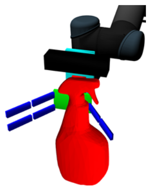
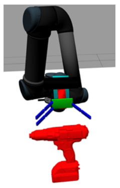
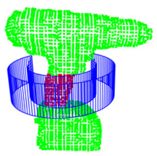
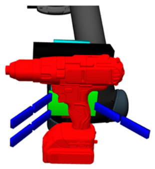
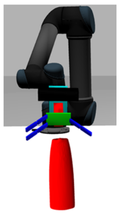
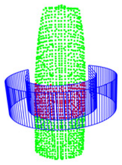
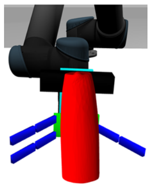
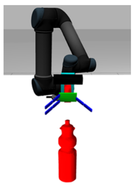
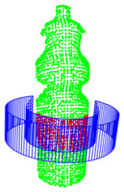
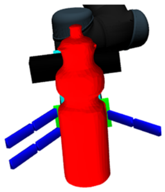
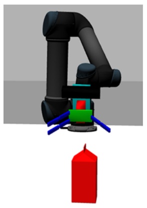
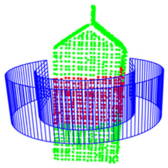
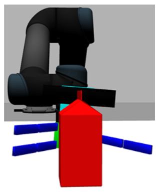
Results of benchmark 2 are shown in Fig. 17 and Table IV. RRTstar, TRRT and SPARStwo have a lower solved runs compared to the other planner algorithms. SBL, BiEST, BKPIECE, LBKPIECE, RRTConnect and BiTRRT compute paths in under 0.1s, SBL is the fastest planner. Planners that have a time invariant stopping goal, except for FMT and TRRT, are producing valid paths within one second. 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.

To sum up, SBL, BKPIECE, LBKPIECE, RRTConnect and BiTRRT can achieve better performances than the other planners based on the comparison results of benchmark 1 and benchmark 2. These five planners can work better with respect to 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.

IV. SIMULATION TESTS

In order to verify our grasping algorithm, the motion planner BiTRRT for the UR5 manipulator is used to perform grasp action. The simulation tests were performed using a personal computer (2 cores, 2.9GHz). Several objects with different geometry shapes are used in the simulation. Table V lists the simulation testes for five objects. The first column shows the setup of each test. The second column shows an example grasp found by the grasping algorithm. The third column shows the robot arm arrived at the grasp point by using BiTRRT as motion planner. The fourth column shows the number of points of the input partial point cloud. The last column shows the average computing time (10 trials for each object). For all the tested objects, it demonstrates that the algorithm can quickly find out a suitable grasp within two seconds.

TABLE V. Experiment results.

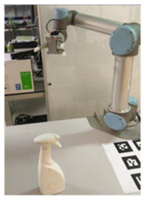
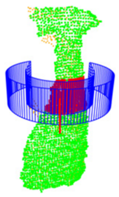
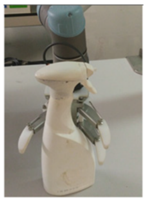
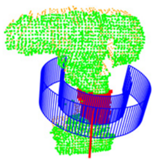
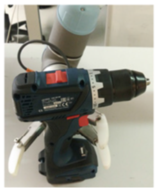
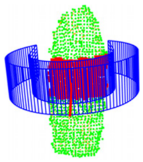
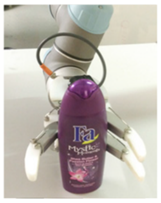
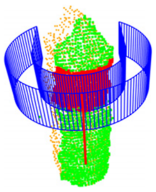
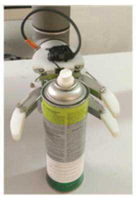
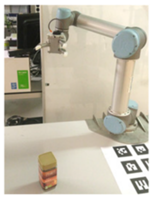
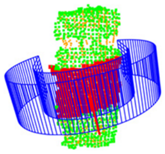
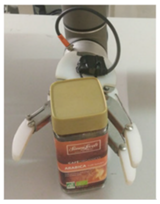
Intial setup	Example grasp found	Grasp execution	Points	Time (s)
			8154	1.95
			7678	1.83
			5274	0.58
			7270	0.87
			4710	0.73

V. EXPERIMENTAL TESTS

The experimental tests were 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 the first column of Table VI. The second column shows the example grasp found by the grasping algorithm. The third column shows the robot arm arrives at the grasp position by using BiTRRT as motion planner. The fourth column shows the number of points of the input partial point cloud. The last column shows the computing time (10 trials for each object).

In comparison with the simulation tests in Table V, the point cloud in experiments may lose some points. For instance, the coffee jar in the sixth column of Table VI lost certain amount of points because the Xtion pro live sensor cannot detect transparent part. The neck of the coffee jar is transparent which is therefore we cannot find the points for neck of the coffee jar. From Table VI, 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 two seconds. Therefore, it demonstrates that the experimental tests are consistent with the simulation tests of using the grasping algorithm. Comparing with the five fast grasping algorithms,[2–6,29–32] it is summarized that our algorithm is distinguished from others according to the following five points:

TABLE VI. Experiment results.

Intial setup	Example grasp found	Grasp execution	Points	Time (s)
			10596	1.74
			9929	1.56
			5267	0.67
			7127	0.91
			4345	0.68

- Grasp adaptiveness: Our grasping algorithm is specially designed for under-actuated grippers, which adds compliance and dexterity without the need of adding additional actuators and sensors. In combination with the low price of under-actuated grippers, our grasping algorithm is more adaptive than others.[2,4–6] Meanwhile, the grasping algorithm proposed in this paper does not rely on object features, which means more adaptiveness than our previous work.[32]
- Object complexity: The presented grasping approach is applicable for relatively complex objects such as electric drill and the cleaner spray bottle. This makes it better than Refs. 2, 4, and 5, which only considers simple objects. Reference 3 transforms the objects into simple shapes (cylinder, disk, sphere and box), which may result in loss of details of objects. Caging based grasping algorithms[29–31] are becoming popular in recent years because of their high flexibility, however, they are only applicable for simple flat objects.
- Computing time: Our algorithm finds a suitable grasp for complex object within two seconds, which is lower in comparison with the results by Refs. 3–6. It is noted that Ref. 2 uses a much lower time because it only uses a RGB image at the cost of losing depth information of the object that can involves grasp risks.
- Grasping direction: Refs. 2, 4, and 5 only consider grasping from top, which can result in unreliable grasp, for instance, picking up the wineglass. In some cases, it is unallowable 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. Moreover, we use the normal of the object for approaching direction, which resembles a human-like grasp.

- Grasp execution: For the available fast grasp algorithms, only Ref. 6 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.

VI. CONCLUSIONS

This paper introduces a novel algorithm for fast grasping of unknown object grasping is based on the C- shape configuration. For the grasping algorithm, the under-actuated grippers is simplified as a C-shape. To accelerate the computing speed, this algorithm uses a single view partial point cloud as input, which is also used for C-shape searching on the target object. The effectiveness and reliability 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. In comparison with other fast grasping algorithms, our algorithm shows significant improvement in terms of the grasp speed and the applicability in practice.

ACKNOWLEDGMENTS

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.

[1] J. Jeannette Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-driven grasp synthesis: A survey," IEEE Transactions on Robotics **30**(2), 1–21 (2013).

[2] J. Baumgartl and D. Henrich, "Fast vision-based grasp and delivery planning for unknown objects," 7th German Conference on Robotics (ROBOTIK 2012), 1–5 (2012).

[3] C. Eppner and O. Brock, "Grasping unknown objects by exploiting shape adaptability and environmental constraints," In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 4000–4006 (2013).

[4] Y.-c. Lin, S.-t. Wei, and L.-c. 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), 1258–1263 (2014).

[5] T. Suzuki and T. Oka, "Grasping of unknown objects on a planar surface using a single depth image," In proceeding of IEEE International Conference on Advanced Intelligent Mechatronic (AIM), 572–577 (2016).

[6] A. t. Pas and R. Platt, "Using geometry to detect grasps in 3D point clouds," 2015 International Symposium on Robotic Research (ISRR), 1–16 (2015).

[7] S. Bhattacharya, M. Likhachev, and V. Kumar, "Topological constraints in search-based robot path planning," Autonomous Robots (AURO) **33**(3), 273–290 (2012).

[8] I. Sucan, M. Moll, and L. Kavraki, "Open motion planning library: A primer," IEEE Robotics & Automation Magazine **19**(4), 72–82 (2014).

[9] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," In IEEE International Conference on Robotics and Automation (ICRA), 4569–4574 (2011).

[10] 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 **32**(9-10), 1164–1193 (2013).

[11] S. Bhattacharya, M. Likhachev, and V. Kumar, "Topological constraints in search-based robot path planning," Autonomous Robots (AURO) **33**(3), 273–290 (2012).

[12] G. Sánchez and J.-C. 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), (6): 403–417 (2003).

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

[14] I. 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), (57): 449–464 (2008).

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

[16] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., (1998).

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

[18] A. 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)*, 233–241 (2005).

[19] 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), 2437–2443 (2013).

[20] 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), 566–580 (1996).
[21] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” CoRR, vol. abs/1105.1186 (2011).
[22] 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).
[23] 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).
[24] O. Salzman and D. Halperin, “Asymptotically near optimal RRT for fast, high-quality, motion planning,” CoRR, vol. abs/1308.0189 (2013).
[25] L. Jaillet, J. Corts, and T. Simon, “Sampling-based path planning on configuration-space costmaps,” IEEE Transactions on Robotics **26**, 635–646 (2010).
[26] 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), 4105–4110 (2013).
[27] A. Dobson, A. Krontiris, and K. E. Bekris, “Sparse roadmap spanners,” *Algorithmic Foundations of Robotics X* (2013), pp. 279–296.
[28] A. Dobson and K. E. Bekris, “Improving sparse roadmap spanners,” In IEEE International Conference on Robotics and Automation (ICRA), 4106–4111 (2013).
[29] W. Wan and K. Harada, “Achieving high success rate in dual-arm handover using large number of candidate grasps, handover heuristics, and hierarchical search,” Advanced Robotics **30**(17-18), 1111–1125 (2016).
[30] W. Wan and F. Rui, “Efficient planar caging test using space mapping,” IEEE Transactions on Automation Science and Engineering **15**(1), 278–289 (2016).
[31] W. Wan and F. Rui, “Finger-position optimization by using caging qualities,” Signal Processing **120**, 814–824 (2015).
[32] Q. Lei, G. Chen, and M. Wisse, “Fast grasping of unknown objects using principal component analysis,” AIP Advances **7**(9), 1–21 (2017).