RBE 595: Vision-based Robotic Manipulation Grasping from Top Surfaces

GitHub repo: github.com/azzamshaikh/RBE_595_Grasping_from_Top_Surfaces_Project

Azzam Shaikh Niranjan Kumar Ilampooranan Swati Shirke Alex Brattstrom Chaitanya Gokule

***Abstract*—In this report, the development of a top-surface grasping algorithm was explored. This is a key task in robotic manipulation that enhances a robot's ability to interact with objects in cluttered or constrained environments. Utilizing the work of Richtsfeld and Vincze as reference, point cloud data techniques were implemented, focusing on surface normal estimation and grasp point detection. Two implementations utilizing concave hulls approaches from different libraries were evaluated, yielding consistent results in grasp point identification and quality metrics. The challenges and future work are proposed to improve the grasping pipeline.**

***Index Terms*—Vision-based manipulation, Grasping**

I. Introduction

Grasping is a fundamental task in robotic manipulation, essential for enabling robots to interact effectively with their environments. Research in grasping and manipulation is still ongoing to tackle the problem of finding good grasps in various scenarios for multiple objects. One particularly versatile and classical approach involves grasping objects from their top surface, which provides stability and allows for manipulation in confined or cluttered spaces. Grasping from the top surfaces for various of objects usually result in acceptable to good grasps. This method is widely applicable in both industrial automation and service robotics, where efficiency and precision are key.

This report will explore the techniques and algorithms used for top-surface grasping, applying the algorithm to various object shapes, and discuss the challenges faced by robotic systems in accurately identifying grasping poses. By leveraging techniques in computer vision, robots can improve their grasping capabilities, leading to more adaptable and intelligent systems capable of operating in dynamic environments.

II. Background

The idea for this project is based on the work proposed by Mario Richtsfeld and Markus Vincze on "Grasping of Unknown Objects from a Table Top". The authors have used the pointcloud data obtained from the top view of the scene to generate the grasp points for each object on the table. The pointcloud is preprocessed to get only the top surface of the objects which is used to convert it to the problem of 2D grasping. Grasp points are then chosen systematically and then their validity is analysed. [1].

Point cloud utilization techniques were referenced to the Point Cloud Library - which includes pointcloud filtering, RANSAC (Random Sample Consensus) for major plane detection, and clustering to segment pointcloud into individual object pointclouds [2].

III. Methods

This section will discuss the methods to implement the algorithms for grasping objects from their top surfaces. The simulation setup for this project will be described. Using this setup, the general grasp detection pipeline will be presented and each step of the pipeline will be discussed in detail.

A. Simulation Setup

For the simulation setup, an environment was provided for the project that utilized ROS 2 and Gazebo 11.10.2 in Ubuntu Humble (22.04 LTS). The simulation contains a RealSense camera that outputs point cloud information of the scene. Figure 1 visualizes the overall setup.

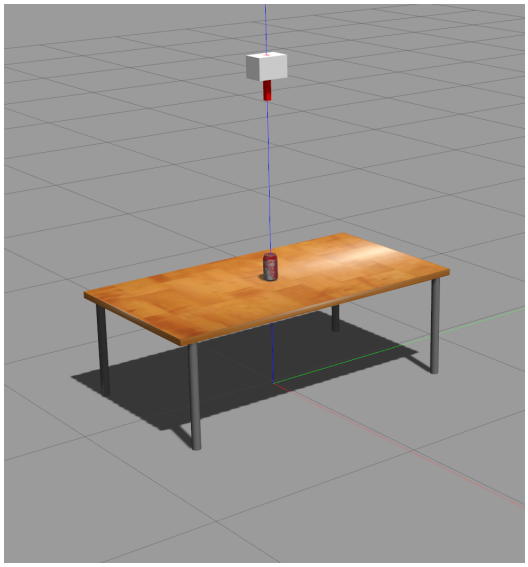

Fig. 1. View of Initial Gazebo Setup

The camera is looking downward at the scene of a table with a Coke can on top of it. Figure 3 visualizes the point cloud obtained from the RealSense camera. The top of the Coke can is shown in the center of the table.

For this project, the setup was modified to include multiple objects. The purpose of this is to evaluate the pipeline's ability

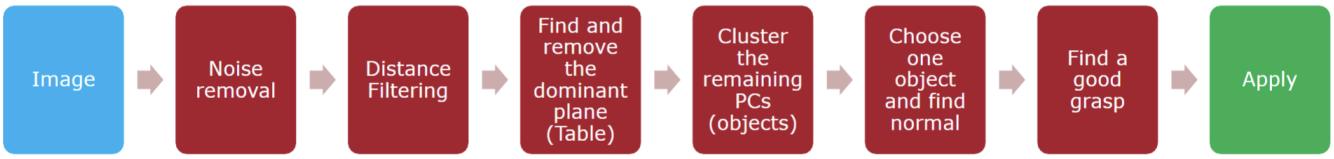

Fig. 2. Grasp Detection Pipeline

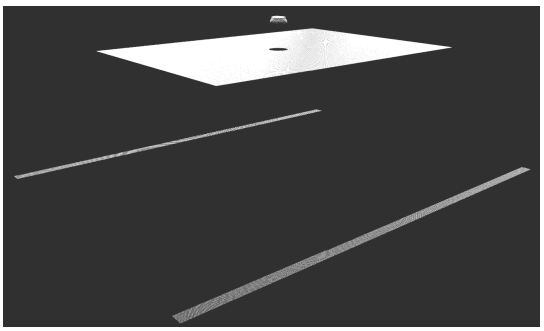

Fig. 3. Initial Point Cloud Viewing in RViz

to detect grasps for multiple objects. Figure 4 visualizes the multiple-object scene. In this case, there is a Coke can, a small box, and a bowl.

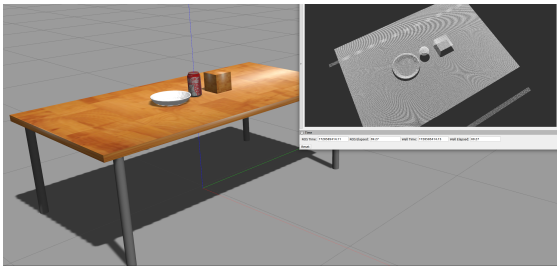

Fig. 4. Multiple-Object Setup; Top Right: Point Cloud View in RViz

B. Grasp Detection Pipeline

The general pipeline for grasp detection can be visualized in Figure 2.

Each step of the pipeline has its own implementation and are developed in separate ROS 2 packages. Each step will be discussed in the following sections.

1) Denoising and Distance Filtering: In the initial stage of the pipeline, typically noise removal is applied, as real-world point cloud data often contains noise. However, for this application, due to the simulation environment used, there isn't significant noise present. Thus, only distance filtering is applied at this step, removing points located beyond 1 meter in the Z direction relative to the camera. This is because the objects are positioned 1 meter from the camera center. By filtering out these distant points, it ensures that the focus remains on the relevant scene. Figure 5 visualizes the filtering of the ground plane from the overall view.

Refer to the *denoise* package for implementation.

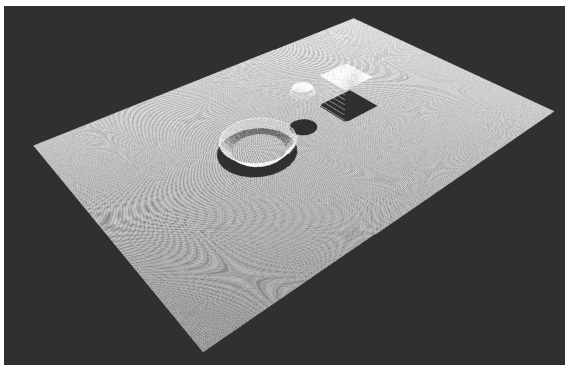

Fig. 5. Point Cloud after Noise Removal and Distance Filtering

2) Detection of the Major Plane: Next, the objective is to detect the dominant plane in the scene, which, in this case, is the tabletop where objects are placed. The goal is to remove this plane from the point cloud to isolate the objects on the table. The RANSAC method is utilized for plane fitting, where three random points from the cloud are selected to define a candidate plane. Using these points, a model is developed and the algorithm fits as many additional points to this plane as possible. Those points who fit well are classified as inliers. This process is repeated for N iterations to identify the plane that fits the most points in the cloud. The PCL's SAC-RANSAC method is utilized to implement this, as described in the Planar Segmentation tutorial [3], which returns the indices of the detected plane points. These indices are then removed from the point cloud data to focus on the objects. Figure 6 visualizes the removal of the plane from the point cloud data.

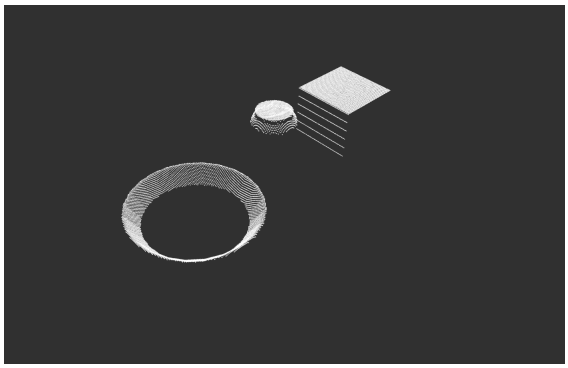

Fig. 6. Removal of Common Plane from Point Cloud

Refer to the *major_plane_detection* package for implementation.

3) Clustering: In the next phase, clusters are created out of the objects from the point cloud data. PCL's KdTree method was utilized as the base data structure for clustering. The KdTree creates an appropriate data structure to map the data points based on distance. Using PCL's EuclideanClusterExtraction class, each individual cluster is extracted from the tree and created into separate point clouds. Each cluster in then published to individual topics to be utilized in downstream packages.

Refer to the *clustering* package for implementation.

4) Determine the Object Normal: After segmenting the point cloud, removing the dominant plane (tabletop), and clustering the scene, the next critical step is to determine the normal vector of the top surfaces of the segmented objects. The normal vector represents the direction perpendicular to the surface and provides vital information about the orientation of the object in 3D space. In the implementation, the RANSAC algorithm is used to detect the planar surfaces on top of the objects. The RANSAC algorithm fits a plane model to the point cloud data, allowing the algorithm to segment the surface with the most points, which is assumed to represent the top surface of the object. By extracting the normal vector components, a mathematical representation of the surface's orientation is obtained. The determined normal vector also plays a role in further filtering the point cloud data. Using this representation, the points below a specified distance (3 mm) from the detected plane are filtered out to ensure the pipeline focuses on the most relevant surface points for grasp planning. Figure 7 visualizes the current point cloud for a can before determining the object's normal.

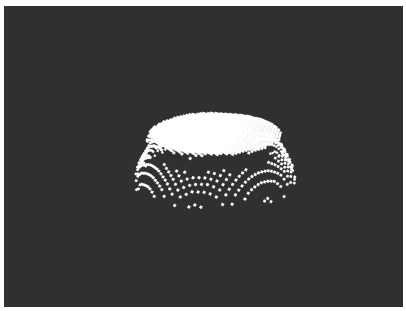

Fig. 7. View of the Coke can point cloud.

After determining the can normal and filtering all points within 3 mm of the normal to the plane, the points are projected to the normal plane. Figure 8 visualizes the can data on the left and the projected normal on the right. Figure 9 visualizes the surface normals of the three objects.

Using this data, there were two implementations tackled. One implementation is utilizing the PCL library while the other one utilizes the alphashape library. In the PCL implementation, using the surface normal data, the PCL's concave hull class was utilized to obtain the concave hull of the objects. The concave hull represents the outer boundary of the point

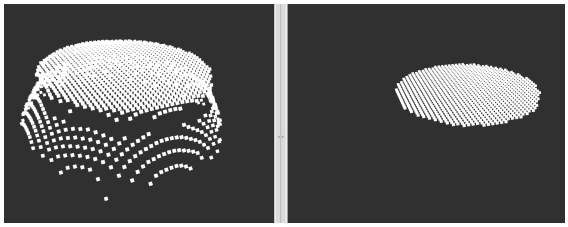

Fig. 8. Left: View of the Coke can point cloud. Right: Top 3mm of the pointcloud projected to surface normal.

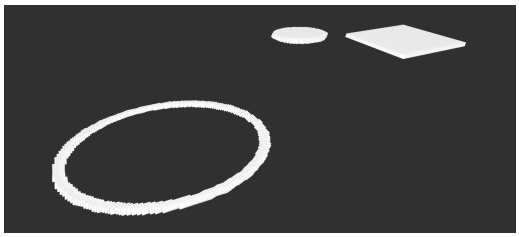

Fig. 9. Surface normals of the three objects.

cloud data. This outer boundary can be used to obtain the grasp points for the object. Figure 10 visualizes the concave hull found the multiple objects.

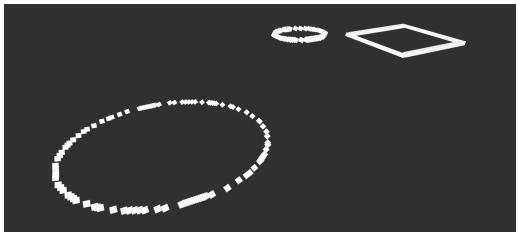

Fig. 10. Concave hull of the three objects.

Refer to the *planar_surface* package for implementation.

In the alphashape implementation, the point clouds shown in Figure 9 are fed directly to the downstream, *planar_grasp* package for processing.

5) Grasp Synthesis: For grasp synthesis, as mentioned, two implementations will be presented.

For the PCL implementation, the package subscribes to the surface normal point cloud as well as the concave hull point cloud. First, the top surface's center of mass was obtained by averaging the x, y, and z values of the point cloud. For grasp point 1, a KdTree of the point cloud was setup, and the closest point to the center of mass was found. Using this point and the center of mass, a vector was created and was projected to the center of the object. The tip of the vector was utilized as a guess point for grasp point 2. This guess was then used as a query point in the KdTree to obtain the equivalent point from the point cloud. This point would be grasp point 2. For both grasp points, the orientation of the vector was needed. First the normal had to be found. This was accomplished by taking the three closest neighbors to each grasp point, creating a covariance matrix from the points, computing the eigenvectors

and eigenvalues of the matrix, and then selecting the smallest eigenvalue. Using this vector, the arctangent of the normal was taken to obtain the vector's orientation. This orientation would be required for evaluating the grasp. Once the grasp points and angles were found, the data could be published to visualize the grasp points and evaluate the grasp quality.

For the alphashape implementation, first the point cloud data is converted to a concave hull as an alphashape Polygon object. From this object, the center of mass is obtained through the object's centroid property. To obtain grasp point 1, the closest neighbor of the center of mass was found. In this case, there was no KdTree setup. Rather, the object's exterior property contained the exterior points of the shape, which was used to obtain the closest point. From grasp point 1 and the center of mass, a vector was drawn. This vector was then reflected to the other side of the center of mass and a LineString object was created. Using this LineString, an intersection was found between the LineString object and the exterior of the shape. This intersection would be grasp point 2. For each grasp point, surface normals were found to obtain the orientation of the grasp point. The surface normals were found by creating lines between the closest points to each grasp point, i.e. the tangent, and creating a vector that was perpendicular to that line, i.e. the normal. From this normal, the arctangent was taken to obtain the orientation. This orientation would be required for evaluating the grasp. Once the grasp points and angles were found, the data could be published to visualize the grasp points and evaluate the grasp quality.

The outcomes of the two implementations will be presented in the Results section. Refer to the *planar_grasp* package for implementation.

6) Evaluating the Grasp: For the evaluation of the two grasp points for each object, an initial approach was utilized based on grasp quality metrics. To accomplish this, first the grasp matrix of the object was computed. By obtaining the center of mass, contact locations, and orientations (converted to rotation matrices), the grasp matrix can be computed. Since this is a 2D grasping evaluation, the grasp matrix shape reflects that. After obtaining the G matrix, three metrics are computed.

The first metric will be based on the minimum singular value. This metric shows how far away the grasp is from losing its closure properties. Thus, the grasp matrix with the largest minimum singular value will be the best grasp as it is furthest from losing its closure properties. It can be computed as shown in Equation (1).

$$Q_{MSV} = \sigma_{min}(G) \tag{1}$$

The second metric will be based on the volume of the ellipsoid in the wrench space. This metric shows the distance from the singularity in every contact configuration. Thus, the grasp with the largest ellipsoid will be the best grasp as it is furthest from the singularity. It can be computed as shown in Equation (2).

$$Q_{VEW} = \sigma_1\sigma_2\sigma_3...\sigma_d \tag{2}$$

The third metric will be based on the grasp isotropy index. This metric shows the uniform contribution of the contact forces in the total wrench. Thus, the closer the index value is to 1, the better the grasp. It can be computed as shown in Equation (3).

$$Q_{VEW} = \frac{\sigma_{min}(G)}{\sigma_{max}(G)} \tag{3}$$

The outcomes of the grasp metrics for the two implementations will be presented in the Results section. Refer to the *grasp_quality* package for implementation.

A second approach was studied to determine force stability based on the grasp positions. Due to time constraints, a flushed out solution was able to be developed. The overall approach is mentioned in the Discussion section.

IV. Results

For the overall grasping pipeline discussed above, for the PCL and alphashape implementations, an rqt_graph visualizes the pipeline, as shown in Figure 11 and 12, respectively. It can be seen that the series of nodes and topics implemented are similar to the pipeline shown in Figure 2.

From these two implementations, the results of the grasp point detection and metrics will be presented.

For the PCL implementation, the grasp points can be seen in Figure 13. The center point is the center of mass and the outer points are the 2 grasp points. The grasp metrics for these points can be seen in Figure 14.

For the alphashape implementation, the grasp points can be seen in Figure 15. It can be seen that the grasp points are virtually identical. Only the bowl has different grasp points rotated by 90 degrees. The grasp metrics for these points can be seen in Figure 16. It can be seen that the metrics are also the same, even though the points for the bowl are slightly different. This is expected as the grasp points are identical.

The usage of grasp quality metrics in this case does not provide meaningful results since they are identical. If different grasp points were found using different approaches, then the metrics would provide information about the better grasp.

Based on the implementations, the grasp point detection is made purely off of the closest node to the center of mass. This inherently defines the two points that get selected for the grasps. Even if there is a better grasp available, the algorithm does not select based on that criteria. The grasp selection process can be heavily improved by incorporating some feedback from a grasp quality metric or a force stability metric to guide the selection process.

V. Discussion

Overall, in this project, the development and implementation of a grasp detection pipeline was presented and evaluated. Using point cloud data from a simulated scene, various aspects of data preprocessing were applied, such as filtering, major plan removal, clustering, etc. After determining the surface normals of an object, two implementations of obtaining grasp points were presented. In both cases, the grasp points were

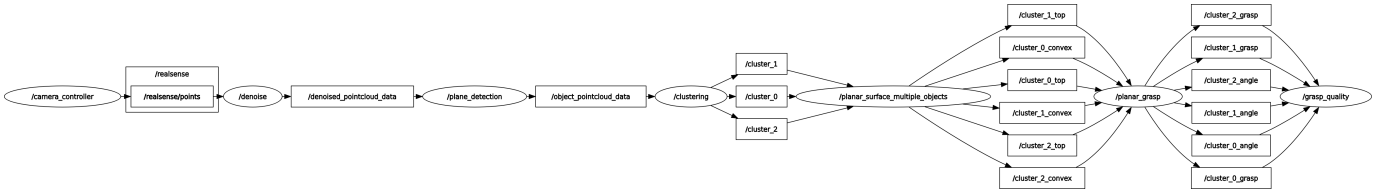

Fig. 11. rqt_graph of the PCL implementation.

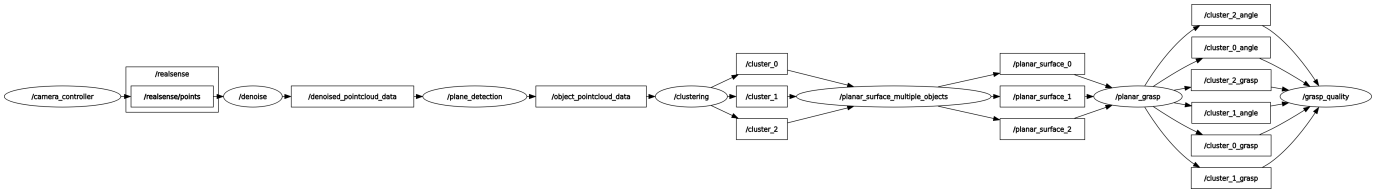

Fig. 12. rqt_graph of the alphashape implementation.

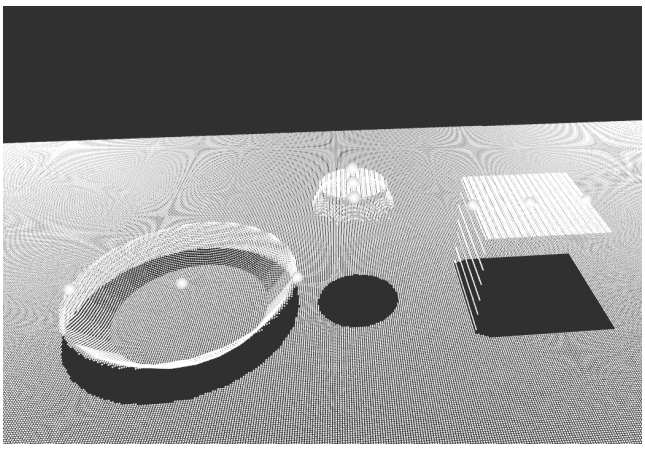

Fig. 13. Grasp points from the PCL implementation.

```
[grasp_quality-10] For cluster 0
[grasp_quality-10]      Singular Value: 0.13691787777244033
[grasp_quality-10]      Ellipsoid: 0.27383672475035986
[grasp_quality-10]      Isotropy Index: 0.09681521717389287
[grasp_quality-10] For cluster 1
[grasp_quality-10]      Singular Value: 0.07001842238739221
[grasp_quality-10]      Ellipsoid: 0.14003684490794405
[grasp_quality-10]      Isotropy Index: 0.04951050123102997
[grasp_quality-10] For cluster 2
[grasp_quality-10]      Singular Value: 0.03739242172447755
[grasp_quality-10]      Ellipsoid: 0.07478485440842332
[grasp_quality-10]      Isotropy Index: 0.02644043109160867
```

Fig. 14. Grasp quality metrics from the PCL implementation.

found using concave hulls from different packages. In both cases, the resulting points and metrics were virtually identical.

This approach is suitable for objects that have relatively planar surfaces and are symmetric. Since the algorithm picks the first grasp point as the closest node to the center of mass, having a uniform or symmetric shape makes that select process for the second grasp point trivial. For objects that have unique shapes, such as a banana, the grasp selection process is not trivial. The utilization of a concave hull results in a shape that

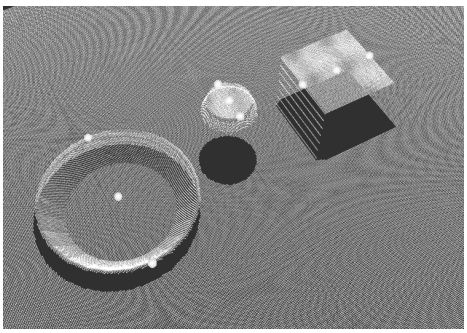

Fig. 15. Grasp points from the alphashape implementation.

```
[grasp_quality-10] For cluster 0
[grasp_quality-10]      Singular Value: 0.13593160115509562
[grasp_quality-10]      Ellipsoid: 0.2718632034617753
[grasp_quality-10]      Isotropy Index: 0.09611815654716684
[grasp_quality-10] For cluster 1
[grasp_quality-10]      Singular Value: 0.07001805896162822
[grasp_quality-10]      Ellipsoid: 0.14003611792718013
[grasp_quality-10]      Isotropy Index: 0.0495102442958996
[grasp_quality-10] For cluster 2
[grasp_quality-10]      Singular Value: 0.037418545044715305
[grasp_quality-10]      Ellipsoid: 0.0748370958889338
[grasp_quality-10]      Isotropy Index: 0.02645890489281863
```

Fig. 16. Grasp metrics from the alphashape implementation.

may not be ideal.

A. *Objects with Complex Geometry*

To evaluate the algorithm's capability on such a shape, a banana model was added to the simulation. Using the alphashape implementation, the grasp points for a banana are shown in Figure 17. It can be seen that the leftmost grasp point is not in contact with the banana. This representation is indicating that the alphashape developed is actually a convex shape, due to where the grasp point is.

An improved approach for obtaining the proper shape of an object is possible in future implementations. One approach would be to utilize the RGB data from the sensor and

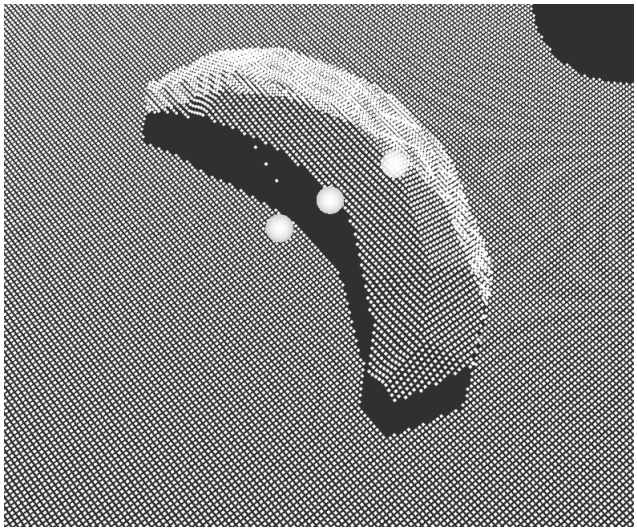

Fig. 17. alphashape algorithm on a banana.

incorporate functions to find contours of an object. This would significantly improve the extraction of an objects shape. Coupled with depth data, a significantly improved point cloud could be found for selecting grasps.

An alternate approach would be to utilize the approach discussed in the original paper, 2D DeLaunay triangulation. This approach provides an advantage when dealing with irregular point distributions as it generates a flexible mesh that covers the full surface area of the object. This would assist in generating better models of the object. An implementation of triangulation was attempted, however, due to time restrictions, a feasible solution could not be rolled out in time.

Another implementation could be to frame this as an optimization problem of finding the best contour for a use case. The objective would be to choose a set of points in the reduced pointcloud (top 3mm) and try to form edges using these points, eventually forming a polygon. A polygon that could provide the maximum area of the surface would be the perfect representation of the object contour. This contour would provide improved point cloud data to generate grasp points with, especially for edge cases like a banana.

B. Major Plane Grasp Point Detection Implementation

In another approach, grasp points were identified using the principal axis method. First, a covariance matrix is created for each object's point cloud, and the principal axis is determined by computing the eigenvalues of the covariance matrix. Next, the normal vector to the principal axis is calculated, and all points lying on it are extracted. Two points are then filtered from this set, specifically those closest to the minimum and maximum x and y values of the points on the normal vector. Due to time restrictions, this implementation could not be rolled out in time.

C. Force Stability Analysis Implementation

As an additional approach to evaluate the quality of grasps, the implementation of a 2-contact force closure grasp analysis

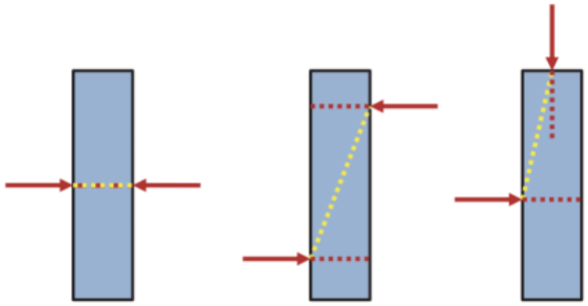

Fig. 18. Approximation for 2-Contact Force Closure Grasp

was attempted. Using the grasp points detected and their surface normals, a line could be drawn between the two grasp points. Using the line as a reference, the angle between the normal direction and the line can be computed. By summing the two angles for the two grasp points, a simple force balance can be generated. The closer the sum of the angles is to zero, the better the force balance. This is visualized in Figure 18.

This grasp metric was attempted to be implemented but could not be rolled out in time. This metric could provide a feasible method for determining a balanced grasp for irregular objects.

REFERENCES

[1] M. Richtsfeld and M. Vincze, "Grasping of Unknown Objects from a Table Top," in *Workshop on Vision in Action: Efficient strategies for cognitive agents in complex environments*. Marseille, France: Markus Vincze and Danica Kragic and Darius Burschka and Antonis Argyros, Oct. 2008. [Online]. Available: https://inria.hal.science/inria-00325794

[2] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China: IEEE, May 9-13 2011.

[3] Point Cloud Library, "Planar segmentation tutorial." [Online]. Available: https://pcl.readthedocs.io/projects/tutorials/en/latest/planar_segmentation.html