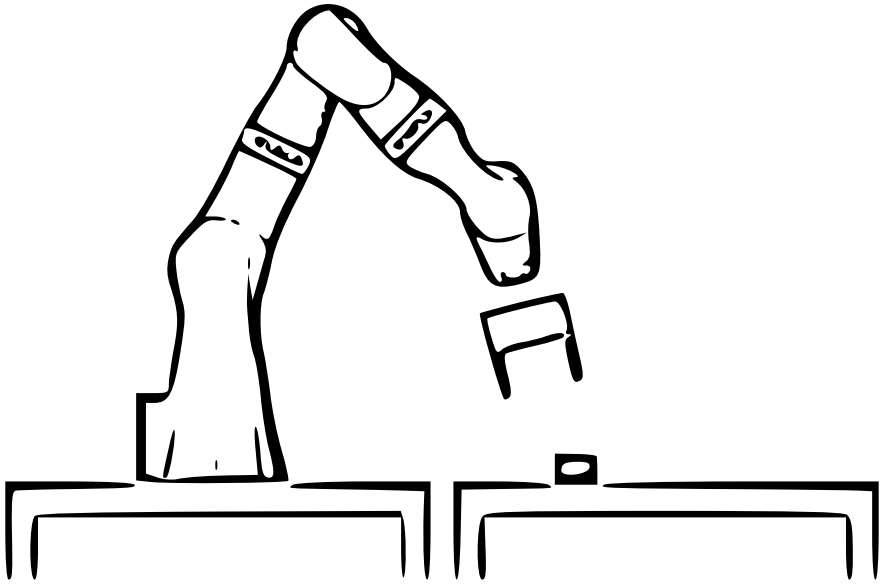
CHALMERS



Automated Tool Selection and Optimal Grasping Point Generation for Robotics

Bachelor's thesis in Electrical Engineering

Rasmus Cervin, Omar Qazzaz, Ruben Sandström, Fredrik Sörfeldt, Edwin Täck, Albin Zahnér

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2025

www.chalmers.se

Bachelor's thesis 2025

Automated Tool Selection and Optimal Grasping Point Generation for Robotics

Rasmus Cervin
Omar Qazzaz
Ruben Sandström
Fredrik Sörfeldt
Edwin Täck
Albin Zahnér

CHALMERS

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2025

Automated Tool Selection and Optimal Grasping Point Generation for Robotics

Rasmus Cervin
Omar Qazzaz
Ruben Sandström
Fredrik Sörfeldt
Edwin Täck
Albin Zahnér

Supervisor: Endre Eros, Chalmers Industriteknik, Applied AI
Examiner: Knut Åkesson, Department of Electrical Engineering
Industrial collaborator: Atieh Hanna, Research and Technology development, Volvo Trucks

Bachelors Thesis 2025
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualization of a robotic arm about to pick an object[1]. Redistributed with permission.

Abstract

This thesis addresses the challenge of automating optimal grasping point generation for robotic manipulation in unstructured environments. Traditional industrial robots rely on manually defined grasping points, limiting their flexibility and efficiency when handling diverse components with varying geometries and positions. To overcome this, an algorithm is developed to analyze CAD files of objects and generate optimal grasping points tailored to different robotic end-effectors, including suction cups, sponges, and grippers. The method involves creating an initial dataset of potential grasping points by analyzing the geometry extracted from CAD models. These grasping points are then filtered based on tool-specific constraints such as surface flatness, accepted torque, and accessibility to ensure successful and stable grasping. The algorithm's effectiveness is validated through visualizations (and to a lesser degree simple simulations) of generated grasping points on various objects of differing shapes and sizes. The developed algorithm reduces the need for human intervention, improving the automated production line, and lays the foundation for increased automation in the future.

Keywords: Robotics, CAD analysis, Automated grasping, Automated tool selection, Robotic manipulation, Unstructured environments, Robotic Grasping, Suction, Gripper, Sponge tool.

Acknowledgements

We would like to express our gratitude to Endre Eros who has been our supervisor throughout this thesis. His support, insightful guidance and feedback has been invaluable during the whole process. His expertise and encouragement has shaped both the direction and quality of our work. We appreciate the time and effort devoted to helping us throughout the thesis.

Rasmus Cervin, Omar Qazzaz, Ruben Sandström,
Fredrik Sörfeldt, Edwin Täck, Albin Zahnér
Gothenburg, May 2025

Contents

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

Acronym	**Expanded version**
ABTF	Aluminum Bolt-Together Fitting
BIP	Big Plate
BLP	Black Plate
BMDSS	Base-Mounted Dual Shaft Support
CAD	Computer-aided design
DBSCAN	Density-based spatial clustering of applications with noise
EWSBC	Extra-Wide Sleeve Bearing Carriage
GFSP	Granite Flat-Surface Plate
JSON	JavaScript object notation
LBH	Linear Bearing Housing
OCC	OpenCASCADE
OCOG	Object common grasp
RL	Reinforcement learning
STEP	Standard for the exchange of product data
STL	Standard Triangle Language

Nomenclature

Indices

i, j	indices for unique elements

Sets

U	set of sample points
P_i	set of points indexed by i
D	sample set

Functions and Operators

f_n	filter or function indexed by n
T	transform function (mapping)
$\wedge$	logical operator "and"

Others

ϵ	small margin of error
r_f	cup radius footprint
d_p	maximum depth penetration
w_f	width footprint
h_f	height footprint
p_c	partial footprint
T	torque (when used as scalar measurement)
S_{uv}	surface parameterized by indices u, v

$\mathbb{R}^n$	n-dimensional spaced room
β	parametric boundary
d_t	orthogonal distance from sample point to center of mass
A_i	area indexed by i
N	arbitrary number (scalar or integer)
A_{eff}	effective contact area

List of Figures

List of Tables

List of Algorithms

1

Introduction

Manufacturing is undergoing a significant transformation driven by the demand for customized products and compliance with diverse international standards[2],[3]. In some modern factories, such as Volvo Trucks, this transformation is apparent in the efforts to balance product diversity with production efficiency.

Historically, traditional industrial robotics has played a key role in automation. These robots are operated in highly structured environments with rigid assumptions: workspace was fixed, objects were always located in the same positions, and tasks were executed through predefined joint trajectories. Operators would manually define the joint values and the time intervals between waypoints, assuming that each part handled was delivered to a fixture at a known and repeatable location. Because the environment did not change and because there was no vision or perception involved, this approach worked well, although limited to operate only under highly controlled conditions[4].

However, such systems are increasingly inadequate in today's dynamic manufacturing contexts[2]. Traditional robots struggle to adapt when the environment becomes unstructured or when the variety of parts increases beyond what manual programming can handle. Moreover, any change in part geometry or position typically requires a human operator to reprogram the robot, introducing delays and inefficiencies. These systems also lack flexibility, are time-consuming to maintain, and often require substantial human oversight for reconfiguration.

As manufacturing shifts toward smaller batch sizes and higher product variability, there is growing interest in collaborative robotic systems where robots work safely alongside humans[5]. One key application of collaborative robots is bin picking: selecting and manipulating parts from randomly organized bins. This task requires a robot to perceive and adapt to unknown object locations and orientations, unlike traditional setups where everything is fixed. Vision systems and intelligent software are critical to enabling robots to detect objects, locate them in a 3D space, and compute how to grasp them.

This thesis focuses on a particularly challenging subset of that domain: unstructured environments where no fixtures or dispensers are used, and where objects have unknown poses and positions. In such scenarios, the robot cannot rely on assumptions about initial conditions. Instead, it perceives the environment using vision systems

and sensors, determines the location and orientation of objects, and calculates how to move its arm accordingly using inverse kinematics. This enables the robot to achieve Cartesian positioning; knowing exactly where its end effector is and how to orient it to perform a successful grasp[6].

The core problem addressed in this thesis is the generation of grasping points on specific locations on an object where a robot can reliably grip. Currently, these grasping points are manually defined by human operators for each part, which is time consuming, error prone, and unsustainable given the increasing diversity of components in modern production. To address this, the project proposes an automated pipeline that analyzes Computer-Aided Design (CAD) models of parts and outputs a set of predefined picking points suitable for robotic grasping[6].

By automating the generation of these grasping points, the system reduces the dependency on manual configuration and enhances the adaptability of collaborative robots in unstructured settings. This is a crucial step toward achieving scalable and flexible automation in modern manufacturing environments where speed, reliability, and reduced human intervention are paramount.

1.1 Purpose

As previously mentioned, traditional robotic manipulation solutions rely on predefined grasping strategies that work well in structured environments, but struggle with the complexity and variability of modern production demands. Challenges arise especially in tasks such as pick-and-place operations, shown in figure 1.1, where robots must handle a wide variety of parts with different geometries and orientation. In these settings, reliable grasp planning becomes essential. The increasing diversity of components and the unpredictability of storage methods require a more adaptive approach[6].

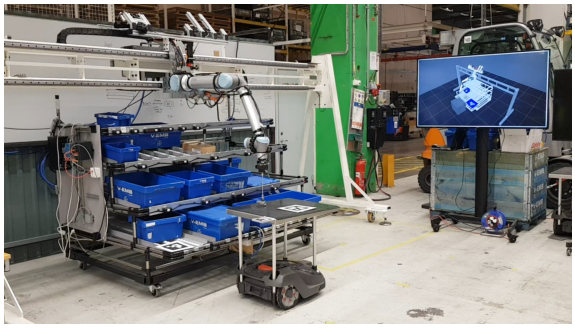


Figure 1.1: Collaborative bin picking by an industrial robot[7]. Redistributed with permission.

The objective of this project is to develop an algorithm that, given an object's CAD file as input, automatically determines and exports the object's optimal grasping points. The development of such an algorithm is a critical component in automating the pick-and-place tasks done by robotic arms as the production industry requires innovative solutions that enhance both productivity and operational flexibility.

1.2 Scope

The developed algorithm should have the following qualities:

- There should be no limit on the number of objects or tools that can be processed. This ensures that the system can scale effectively, managing a wide variety of objects without performance degradation.
- The algorithm should be able to handle new objects that have not been seen before.
- The algorithm should allow for an easy way to add new tools and end-effectors.
- The algorithm is not bound by a set run-time limit, meaning increasingly complex objects are allowed to take increasingly more time to process. That said, the algorithm will be developed to keep run-time reasonably close to the minimum allowed by chosen software.

As the focus of this project lies exclusively on the development of the algorithm for grasping point generation and optimal tool selection, the project is bound by the following constraints:

- The algorithm does not receive any feedback or data from the robotic system. This limitation prevents real-time analysis, meaning the algorithm cannot account for rotation and location of objects in real life. All grasping points are produced solely from the CAD files with respect to the tools and objects at hand.
- The algorithm is limited solely to processing and analyzing CAD files in the Standard for the Exchange of Product model data (STEP)(.stp file extension) format and tool specifications must be provided in a JavaScript Object Notation (JSON)(.json file extension) file following a pre-defined template.
- The project does not include integration with or control of robotic hardware.

1.3 Thesis Questions

The following three questions guide the method in this thesis:

- How can the geometry of CAD files be analyzed and extracted by written code?
- How can the extracted data be used to find optimal grasping points for a set of different robotic tools?
- How can these grasping points be ranked in order from the most optimal to least optimal?

2

Background

2.1 Literature review

Today, a common manual approach involves the use of a gripping library, where experienced operators identify optimal grasping points and select appropriate tools from a predefined set[8]. Although effective, this method is time-consuming, requires specialized expertise, and is prone to delays[9]. Moreover, in facilities such as Volvo Trucks that handle a high number of unique and specialized parts, the process of determining grasping points and selecting tools must be frequently revisited to handle evolving designs and specifications[5].

A more recent approach that has shown promise utilizes reinforcement learning (RL) to solve robotic pick-and-place tasks and involves integrating traditional control methods (such as Rule-Based systems) with reinforcement learning algorithms[10]. This combination allows robots to execute desired operations without the need for manually defined grasping points. However, reinforcement learning requires significant amounts of labeled data, which would have to be provided by human employees, making it ineffective at reducing the work load on employees.

Computational algorithms have been proposed to optimize grasp strategies for multiple objects. For instance, the Object Common Grasp (OCOG) algorithm configures grasping for sets of planar objects by mapping possible grasps that satisfy force-closure and quality criteria[9]. While this approach demonstrates the feasibility of computational grasp optimization, handling diverse parts in more complex, unstructured environments still requires further development.

An alternative strategy is to leverage detailed CAD models for grasp planning. A tool called "GraspIt!" uses CAD data to create accurate representations of both objects and robotic hands[11]. By processing these CAD models, the system extracts the object's surface geometry and identifies regions where the robot's fingers can make contact. This method is invalid as the project was developed back in 2009, with only minor fixes since, meaning it lacks new technological developments. It has also seemingly been abandoned since 2021, which causes compatibility issues.

Another method, MoveIt previously attempted to fulfill this project's niche but has since been discontinued. Moreover, MoveIt Grasps is exclusively built on ROS1, a framework currently being phased out, and it lacks implementation of more modern

methods and functionalities. Similarly, GraspIt!, also relies on outdated techniques and does not specifically cater to systems based purely on static CAD file analysis. Other related approaches currently available primarily focus on live data processing from sensors leaving the specific area of static CAD analysis lacking.

Despite the existence of various related projects and solutions, this thesis addresses a specific niche within robotic grasping point generation. The primary distinguishing factor of this project is its ability to operate solely by analyzing static CAD files, without relying on live sensor data or real-time feedback from a robotic system.

Due to this, this thesis contributes by offering a solution that focuses exclusively on CAD-based grasping analysis avoiding the limitations associated with outdated or deprecated software.

2.2 Ethics

The main ethical conflict when developing means of automation, including the generation of grasping points, is affecting job opportunities within the existing workspace. Should the grasping point generation be successfully automated, these employees could end up jobless unless preventative measures are taken. These preventative measures could include investments in re-education, not only from companies utilizing automation, but also governments, to ensure employees can be seamlessly re-assigned[12]. Another possibility could be to develop a framework for limiting automated systems to always require human supervision, even if the system is theoretically powerful and robust enough to self-supervise. However, given the high competitiveness of the manufacturing sector, this may prove difficult.

Re-assignment is also not a perfect solution to prevent the risk of increased unemployment, as one must not underestimate the impact it can have on an individuals daily routine. For many, work has deeper meaning than the financial benefit, in that it gives feelings of belonging to a community and a sense of self-worth in the skills that the individual has learned[12]. Too frequent re-assignment could destabilize the working environment, leading to increased stress, which is proven to be detrimental to employees mental health and, by extension, work performance[13].

3

Method

3.1 Overview

The algorithm developed in this project is divided into four parts:

- Point generation
- Tool-based point filtering
- Computation of the approach orientation
- Grasping Point evaluation and ranking

The point generation data set is created once and used in parallel across all tools, generating a new set of points for each tool. The resulting point sets are then ranked based on their estimated grasp quality. The general workings of the algorithm from start to end are visualized in figure 3.1

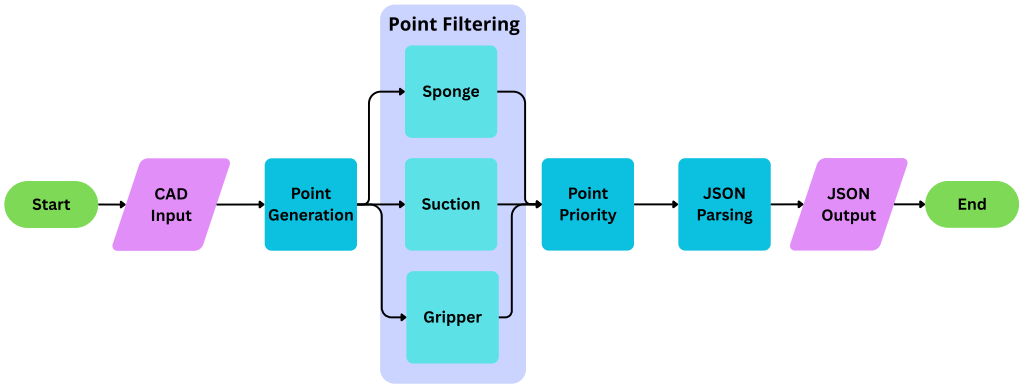
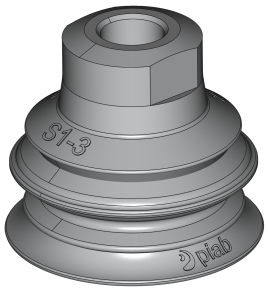
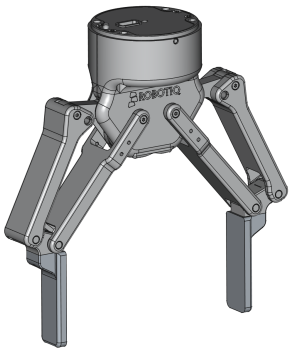


Figure 3.1: Flowchart of the algorithms different steps and processes.

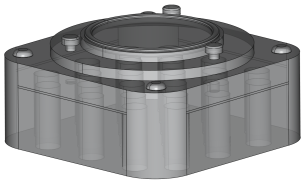
The tool filters are designed to work with three types of tools: suction cup, gripper, and sponge, shown in figure 3.2a, 3.2b and 3.2c respectively. Each tool can be customized by adjusting parameters to create new variations of the tool, i.e. a suction cup of a different diameter. The given tool parameters are central to the filtering constraints.



(a) CAD model of the suction cup end-effector, showing a single suction channel and a circular contact area



(b) CAD model of the gripper end-effector while fully opened



(c) Semi-transparent CAD model of the sponge end effector showing multiple suction channels inside a rectangular contact area

Figure 3.2: CAD models of the three different types of end-effectors.

3.2 Implementation Details

The algorithm is built on Python version 3.12. The algorithm mainly utilizes the python libraries *trimesh* and *pythonocc-core* for loading and analyzing CAD files, as well as generating and filtering points[14],[15]. *Pythonocc-core* is itself a wrapper for the OpenCASCADE (OCC) library based on the C++ programming language. The algorithm also utilizes the commonly used library *numpy* in order to perform many mathematical calculations.

Pythonocc-core describes objects via mathematical expressions and a class hierarchy. This ensures zero data loss and precise calculations. *Trimesh*, on the other hand, describes objects via a mesh. This mesh covers the surface of the object and is constructed via triangles, resulting in potential data loss. For suction cups and sponge tools, the algorithm uses *trimesh* for its improved computational efficiency. For the gripper tool, the algorithm relies solely on *pythonocc-core*.

3.3 Tool Constraint Parametrization

To enable the handling of a variety of tool types, each tool type is parametrized, allowing for the customization of specific tools. Three tool-types are addressed in this thesis: suction cup, gripper and sponge, each with its own set of parameters and constraints.

The **suction cup** tool's properties affect the constraints in two key ways. First, the maximum footprint width of the suction cup, which corresponds to the radius of the cup, denoted as r_f. Second, the maximum penetration of the cup for which the cup

can be compressed to ensure a sufficient closed surface. The maximum penetration depth is denoted as d_p.

The **sponge** tool's, which shares similarities with the suction cup mechanics, operates under two primary constraints. The footprint area, defined by a rectangle with a width of w_f and a height of h_f, and the maximum penetration depth d_p. While the sponge tool is often valued for its increased penetration depth, its main advantage is its ability to create a vacuum seal without requiring full surface contact. It can remain effective even when only a fraction p_c of its footprint is in contact with the surface.

The **gripper** tool's properties is constrained by the depth the gripper arms can grasp without colliding with the object and by the maximum width between the two arms of the gripper.

3.4 Surface Point Generation

For each face of the object, the surface geometry $S_{uv} \in \mathbb{R}^2$, defined in the (u, v) parameter space, is extracted along with its extreme points. These extreme points define the parametric boundary $\beta \subset S_{uv}$. A set of points $U = \{(u_i, v_i)\}_{i=1}^{N}$ is then generated uniformly along β at fixed distance intervals. The initial point set U is visualized in figure 3.3.

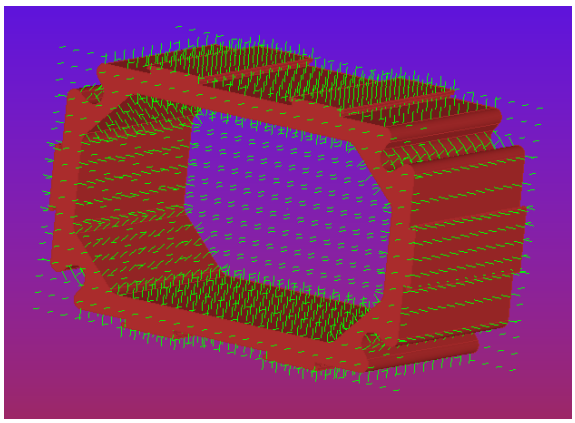


Figure 3.3: Points generated on object in the first stage of point generation.

The point set U is subsequently filtered to remove points lying outside the valid surface region. Here, the valid surface is defined as the region enclosed by the outer boundary curve, excluding any regions within inner boundaries (holes). The point set U after initial filtering is visualized in figure 3.4.

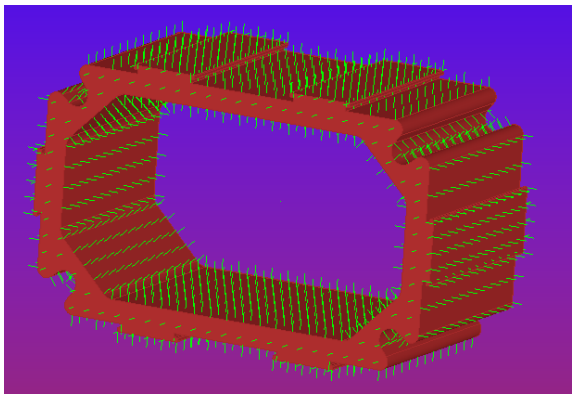


Figure 3.4: Remaining generated points on object in the second stage of point generation.

A mapping function $T : \mathbb{R}^2 \rightarrow \mathbb{R}^3$, defined as $T(u, v) = (x, y, z)$, transforms the points from local surface space to global object space. The resulting 3D points $P_i = (x_i, y_i, z_i)$ are stored along with their corresponding surface normal n_i, forming a paired sample set $\mathcal{D} = \{(P_i, n_i)\}_{i=1}^{N}$ for each location. An algorithmic description of the initial point generation process is available in algorithm 1.

Algorithm 1: Generating Samples from Surface Points.

Data: shape (geometry), point_distance (sampling distance)
Result: PointSet containing sampled points and corresponding normals
Step 1: Generate Initial Sample Points
samples ← GenerateFacePointClouds(shape, point_distance);
Initialize empty lists: sample_points, sample_normals;
Step 2: Process Each Face in the Sample Data
foreach *(face, points) in samples* **do**
    surface_points ← FilterPointsOutsideFace(points, face);
    (_points, _normals) ← UVPositionToGlobal(surface_points, face);
    Append _points to sample_points;
    Append _normals to sample_normals;
end
Step 3: Create and Return PointSet
point_array ← Convert(sample_points to numpy array);
normal_array ← Convert(sample_normals to numpy array);
return PointSet(points=point_array, normals=normal_array, cog=self.cog, mass=self.mass);

3.5 Filtering Process

Once the initial sample set $\mathcal{D}$ is generated, a series of filters are applied to remove invalid grasping points and tools.

3.5.1 Suction Cup Tool

To be considered valid for suction, each point in the generated set $\mathcal{D}$ must satisfy three constraints specific to the suction cup tool:

- The local surface deviation around the point must be within the tool's allowed tolerance.
- The suction cup's full footprint, centered at the point, must lie entirely within the surface boundary.
- The approach path of the tool toward the point must be free of collisions with the object.

To evaluate these constraints, a local height map is constructed around each point. The height map is defined as a square grid centered on the point, with a side length of $2r_f + \epsilon$, where ϵ is a small margin. The grid is generated by projecting points along the surface normal direction from a fixed height h above the surface. Height values are recorded relative to the sampled point, which is treated as the local origin ($z = 0$) in the height map. An algorithmic description is available in algorithm 2, and an example surface and the corresponding height map at sample point $(x, y, z) = (0, 0, 0)$ are shown in figure 3.5 and 3.6.

Algorithm 2: Z-Map Generation using *Trimesh* and Transformations.

Data: positions, normals, size, mesh (triangles), sampling_rate, z_max_threshold
Result: Z-max maps and transformation matrices
Step 1: Initialize Transformation Matrices
foreach *(normal, point) in (normals, points)* **do**
    TR ← TransformMatrixToZAxis(normal, point);
    Add TR to the list of transformation matrices;
end
Step 2: Generate Sampling Grid
x_samples ← linspace(-size/2, size/2, sampling_rate);
y_samples ← linspace(-size/2, size/2, sampling_rate);
grid_x, grid_y ← meshgrid(x_samples, y_samples);
samples_flat ← stack(grid_x, grid_y);
Step 3: Filter Triangles by Size
origin_triangles ← mesh.triangles;
valid_indices ← RemoveTinyTriangles(origin_triangles, min_area=0.5);
Step 4: Generate Z-Max Maps for Each Transformation
foreach *TR in transformation matrices* **do**
    large_triangles ← origin_triangles[valid_indices];
    transformed_triangles ← ApplyTransformToTriangles(large_triangles, TR);
    Shift triangles by -z_max_threshold along the Z-axis;
    filtered_triangles ← FilterOutsideTriangles(transformed_triangles, size/2);
    point_mask_batch ← FastPointInTriangle(samples_flat, filtered_triangles);
    Step 5: Compute Plane Parameters
    normals_batch, D, valid_planes ← ComputePlaneBatch(filtered_triangles);
    filtered_triangles ← filtered_triangles[valid_planes];
    point_mask_batch ← point_mask_batch[valid_planes];
    if *no valid triangles* **then**
        continue;
    end
    Step 6: Calculate Z-Values
    x_grid ← repeat(samples_flat[:, 0], len(filtered_triangles));
    y_grid ← repeat(samples_flat[:, 1], len(filtered_triangles));
    z_values ← ZFromXY(x_grid, y_grid, normals_batch, D);
    Set invalid Z-values to NaN;
    z_values ← z_values + z_max_threshold;
    z_map ← nanmax(z_values, axis=0);
    Append z_map to z_max_maps;
end
return (z_max_maps, transformation matrices);

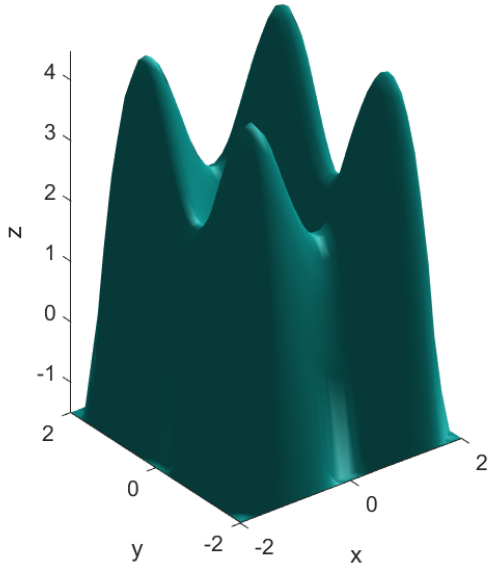


Figure 3.5: Example of an item surface, scale in centimeter.

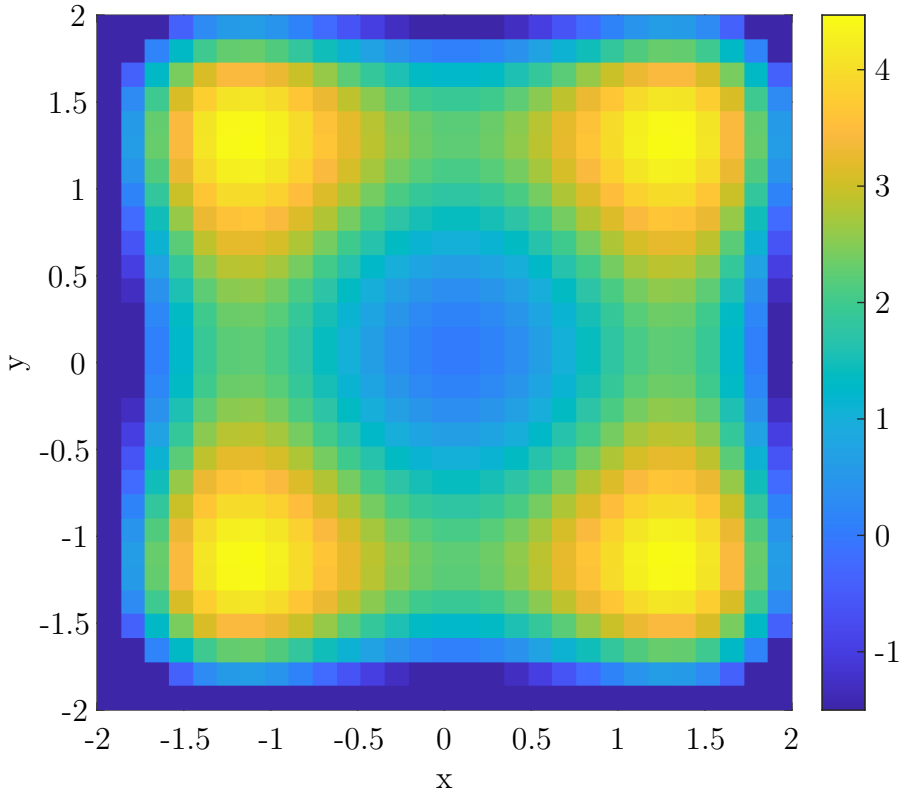


Figure 3.6: Height map of surface in figure 3.5 at point $p = (0, 0, 0)$, scale in centimeter.

To ensure that points are suitable for grasping with a suction cup, each point undergoes three constraint checks, both evaluated over the same local circular region of radius r_f, centered at the point.

Surface Deviation Filter f_1

Enforces the constraint on allowable surface deviation. Using the local height map, the absolute height at each value within the circular region is compared to the maximum allowable penetration depth d_p.

The penetration depth corresponds to the suction cup's malleable tip. This allows the tool to achieve a sufficient suction over slightly uneven surfaces. Such actions are visually represented in figure 3.7. If any value exceeds d_p, the sample is considered invalid and discarded. An algorithmic description is available in algorithm 3.

Algorithm 3: Surface Flatness Evaluation and Filtering.

Data: samples with Z maps, penetration threshold, radius
Result: Filtered list of flat samples
Step 1: Initialize an empty list for samples on a flat surface;
foreach *sample in samples* **do**
    Extract a Z map patch from the sample based on the specified radius;
    Check Flatness:
    if *all absolute values in the patch are below the penetration threshold* **then**
        Add the sample to the list of flat samples;
    end
end
return list of flat samples;

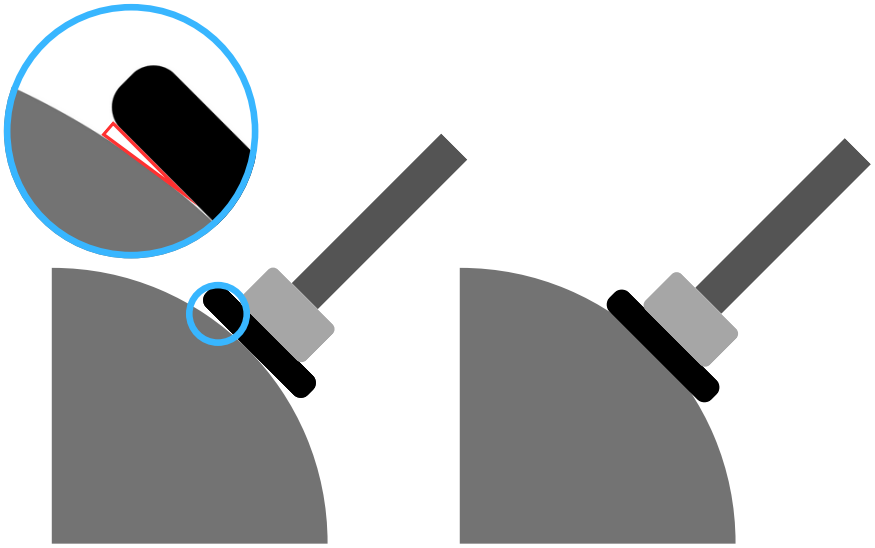


Figure 3.7: Suction cup tolerances and minor ability for adjustment.

Surface Boundary Filter f_2

Ensures that the suction cup footprint remains fully within the valid surface. If any part of the circular region of radius r_f extends beyond the outer boundary or intersects an inner hole, the point is discarded. This condition is evaluated by verifying whether any value within r_f of the height map has a value less than $-d_p$. A visualization of this filter is shown in figure 3.8.

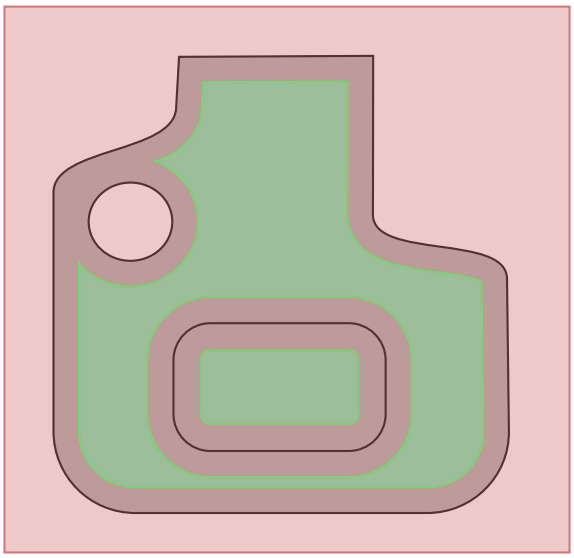


Figure 3.8: Points laying no closer than the suction cup's radius to any boundaries.

Free Approach Filter f_3

Removes all points for which the suction cup footprint intersects any part of the object geometry along the surface normal direction. Using the same surface evaluation process, if any values within r_f exceeds d_p the points approach path is obstructed and is discarded. A visual demonstration of the filter is represented in figure 3.9.

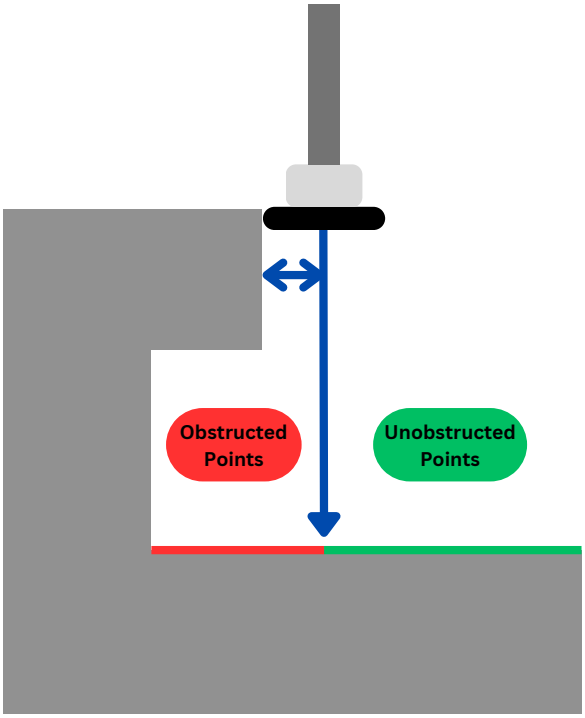


Figure 3.9: Suction cup tool obstructed approach path.

Maximum Torque Load Filter f_4

Eliminates points that experience a torque load that exceeds the maximum torque restriction specified by the tool. The torque load experienced by the point is computed using the orthogonal distance between the point position and the object's center of mass along the point normal.

Complete Filter

The filters are applied to enforce the constraints of the various suction cup tools across the object. Since all surface evaluation filters use the same evaluation process on the same area and values, the surface evaluation can be computed in a single pass F_s. Once the surface has been validated, the torque filter is applied to eliminate samples with excessive maximum torque load.

$$F_s = f_1 \wedge f_2 \wedge f_3 \tag{3.1}$$

$$F_c = f_4 \wedge F_s \tag{3.2}$$

3.5.2 Sponge Tool

The sponge tool operates similarly to the suction cup in that it requires a vacuum seal to generate force. However, due to having multiple suction channels, it does not require full coverage and can function with partial contact with the surface. Additionally, because the sponge tool is defined as a rectangle, it allows for multiple rotations. This provides greater flexibility in positioning. To ensure sufficient suction, each point in the generated set $\mathcal{D}$ must satisfy the following constraints:

- A minimum fraction p_c of the sponge tool footprint must maintain sufficient coverage when attempting to remain within the acceptable deviation range.
- The approach path of the tool toward the point must be free of collisions with the object.

The constraints are evaluated using the same surface evaluation process as the suction cup in section 3.5.1. For each sample, a height map is generated with the side length equal to $max(w_f, h_f)$ around the point.

Sufficient Contact Filter f_1

The sufficient contact filter ensures that the point provides adequate contact for the sponge tool to generate enough suction force to lift the object. First, the height map values within the footprint area are extracted. These values are then evaluated to ensure that a sufficient number of values are within the acceptable deviation range. Points that do not meet the minimum coverage requirement are discarded. An algorithmic desciption is available in algorithm 4

Algorithm 4: Coverage Check for Patch Validity.

Data: coverage_threshold, penetration_threshold, patch
Result: Boolean indicating whether the patch is covered
Step 1: Determine Peak Value
peak_value ← maximum height in the patch (ignoring NaN);
Step 2: Check Peak Value against Threshold
if *peak_value > penetration_threshold* **then**
  return False `// Patch is not covered`
end
Step 3: Calculate Deviation and Valid Mask
difference ← absolute difference between patch and peak_value;
valid_mask ← (difference ≤ penetration_threshold);
Step 4: Calculate Coverage Fraction
n_valids ← number of True values in valid_mask;
total_values ← number of elements in patch;
fraction ← n_valids / total_values;
Step 5: Return Coverage Check
return (fraction ≥ coverage_threshold);

Free Approach Filter f_2

Similar to the height map evaluation method described in Section 3.5.1, the sponge uses the same approach, but analyzes the footprint of the rectangular sponge instead.

Combinatory Filtering

These filters are applied to enforce the constraints of the different sponge tools on all objects. The complete filtering function is defined mathematically according to equation 3.3.

$$F_s = f_1 \wedge f_2 \tag{3.3}$$

Tool Rotation Analysis

Testing multiple rotations of the tool for each point increases the likelihood of finding valid points. To evaluate different rotations, the tool footprint on the height map is rotated. If the rotated patch meets the validity criteria, a new rotated point is generated.

3.5.3 Sample Clustering of Suction and Sponge

After filtering the complete set of valid grasping points, the next step is to refine the point set to include only the samples with the highest probability of success. This is accomplished by applying the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm in two stages.

First, the normals are clustered to ensure representation of all directions. Next, the grasping point positions are clustered to group spatially close samples. This two-step clustering process improves future ranking and usability of the point set. An algorithmic description is available in algorithm 5.

Algorithm 5: Sample Clustering Using DBSCAN.

Data: sample normals, sample positions, angle_threshold, position_threshold, min_cluster_size
Result: List of clustered sample indices
Step 1: Cluster by Normal Direction
Normalize the sample normals to unit vectors;
Compute the angular distance matrix using cosine similarity between normalized normals;
Convert angle_threshold from degrees to radians to obtain radian_threshold;
Apply DBSCAN with parameters: `eps=radian_threshold`, `min_samples=1`, `metric='precomputed'`;
normal_clusters ← resulting cluster labels from DBSCAN;
Step 2: Cluster Positions within Each Normal Cluster
Identify unique normal cluster labels from normal_clusters, excluding noise (-1);
foreach *cluster_label in unique normal cluster labels* **do**
    Extract sample indices associated with the current cluster;
    Select corresponding positions from the position data;
    Apply DBSCAN on the position subset with parameters: `eps=position_threshold`, `min_samples=min_cluster_size`;
    position_clusters ← resulting cluster labels from DBSCAN;
    foreach *position_label in unique position cluster labels* **do**
        Collect indices corresponding to the current position cluster;
        Append the clustered indices to the final list of clusters;
    end
end
return final list of clusters;

3.5.4 Gripper Tool

For a point to be considered a valid grasping point for a gripper tool, the points must fulfill the following constraints:

- The gripper must be able to reach the point without colliding with itself or the object.
- The thickness of the gripping surface corresponding to the point must not exceed the maximum gripping width of the gripper.

These constraints are evaluated via filters, where the algorithm will output points that satisfy all requirements.

Surface Boundary Filter f_1

This filter removes points where the gripper's footprint would overlap with the object or where the gripper cannot reach. It works by applying a minimum and maximum allowable distance constraint, illustrated in figure 3.10, and described algorithmically in algorithm 6.

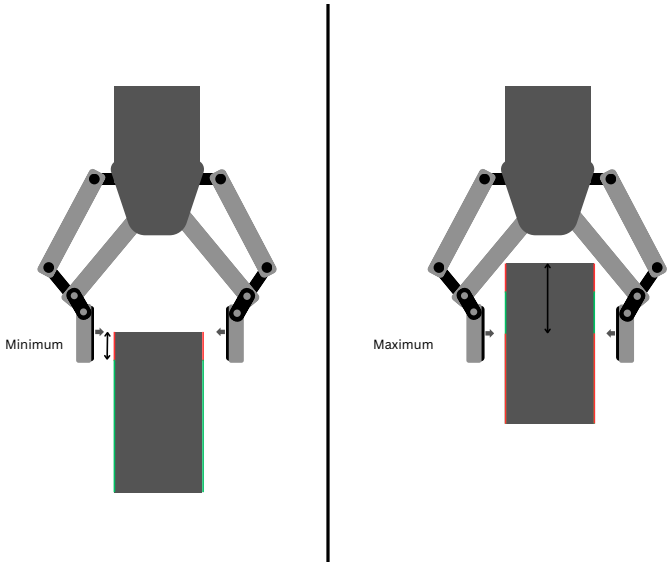


Figure 3.10: Minimum and maximum surface boundary filtering for gripper tools.

Algorithm 6: Surface Boundary Filter

Data: $SurfacePoints$
Result: $FilteredPoints$
foreach $Points, Face \in SurfacePoints$ **do**
  $OuterWire \leftarrow OuterWire(Face)$
  for $Point \in Points$ **do**
    if $MinDist < Distance(Point, OuterWire) < MaxDist$ **then**
      $CandidatePoints.Append(Point)$
    end
  end
end
return $CandidatePoints$

Point Pair Filter f_2

For each point, this filter finds a potential paired point. It removes any points where the pair is too far away, shown in figure 3.11, or where the surface normals of the two points are not sufficiently aligned, shown in figure 3.12. An algorithmic description of the filter is available in algorithm 7.

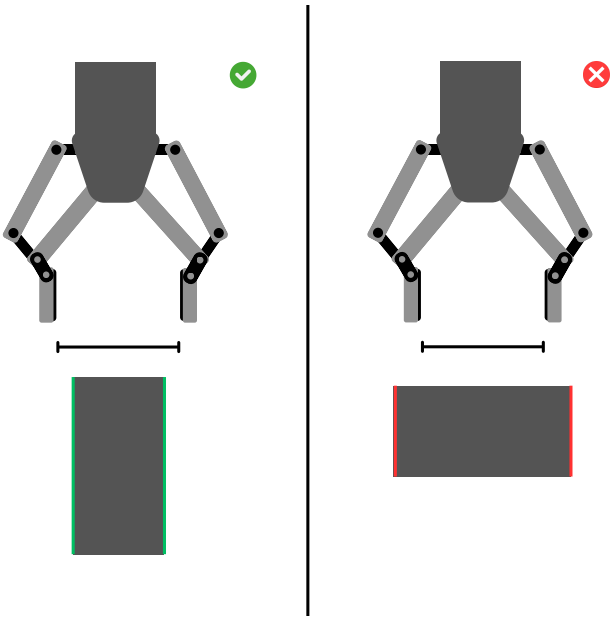


Figure 3.11: Filtering based on maximum point pair distance for gripper tools.

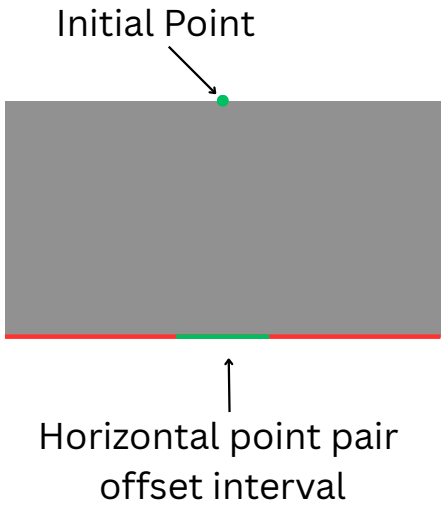


Figure 3.12: Maximum point pair offset filtering for gripper tools.

Algorithm 7: Point Pairs Filter.

Data: $Points$, $MaxAngle$, $MinDist$, $MaxDist$
Result: $FilteredPointPairs$
foreach $Point \in Points$ **do**
  $Front, Back \leftarrow$ intersect with ray from $Point$ in the negative direction of normal to the shape in $Point$
  if *angle between normal for* $Front$ *and* $Back$ *is less than* $MaxAngle$ **then**
    $CandidatePointPairs \leftarrow (Point, Back)$
  end
end
return $CanditatePointPairs$

Closest Orthogonal Filter f_3

This filter identifies the midpoint between each valid point pair and searches for the nearest orthogonal path that leads outward from the object, helping to determine a viable gripping direction. The filter is visualized in figure 3.13 and an algorithmic description is available in algorithm 8.

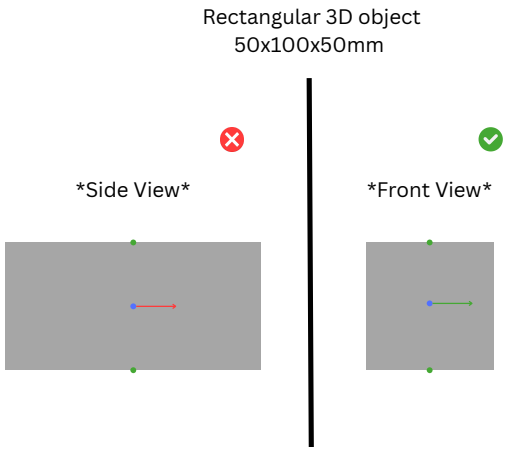


Figure 3.13: Illustrating closest orthogonal path leading outward from the object

Algorithm 8: Closest Orthogonal Filter.

Data: *PointPairs*, *Shape*
Result: *FilteredPoints*
foreach *PointPair* ∈ *PointPairs* **do**
    Front, *Back* ← *PointPair*
    Mid ← middle of *Front* and *Back*
    Plane ← plane through *Mid* with *Normal* same as *Front*
    Wire ← intersect between *Plane* and *Shape*
    CandidatePoint ← point on *Wire* closest to *Mid*
end
return list of *CandidatePoints*

Collision Filter f_4

By generating a 3D model of the gripper, this filter checks for potential collisions with the object, as shown in figure 3.14. It ensures that there is a clear, unobstructed path for the gripper to approach an object. An algorithmic description is available in algorithm 9.

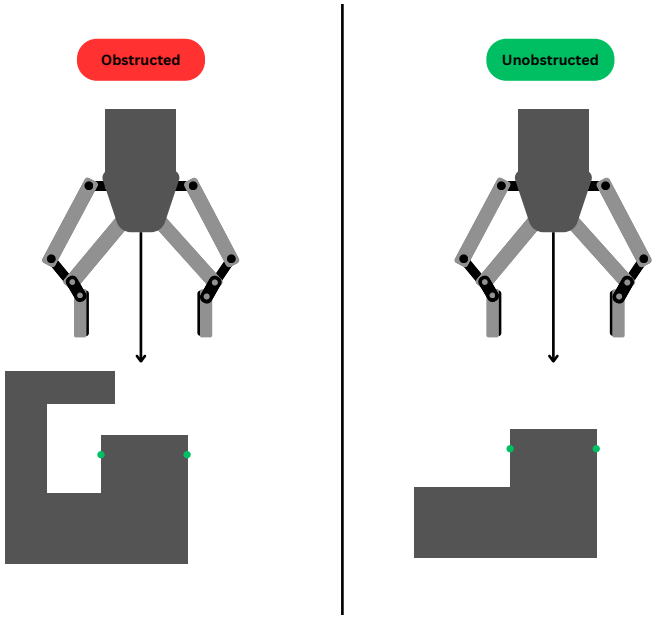


Figure 3.14: Example of a blocked and a free approach path for a gripper.

Algorithm 9: Collision Filter.

Data: $Breadth$, $Width$, $Depth$, $Points$, $Shape$
Result: $FilteredPoints$
foreach $Point \in Points$ **do**
    $Model \leftarrow$ simplified swept volume based on $Width$, $Breadth$ and $Depth$ with a small tolerance
    move model to $Point$ position and orientation
    $Distance \leftarrow$ distance between $Model$ and $Shape$
    if $Distance > 0$ **then**
        $CandidatePoint \leftarrow Point$
    end
end
return *list of* $CandidatePoints$

Filter Clustered Points f_5

Removes points located too close to each other to remove clusters. Points that are closer to center of gravity are prioritized. This filter operates differently from the clustering filter for suction cups and sponge tools. An algorithmic description is available in algorithm 10.

Algorithm 10: Filter Clustered Points.

Data: $Points$, $CenterOfGravity$, $MinDist$
Result: $FilteredPoints$
$SortedPoints \leftarrow Points$ sorted by distance to $CenterOfGravity$ in ascending order
foreach $P1 \in Points$ **do**
    foreach $P2 \in CandidatePoints$ **do**
        if *distance between* $P1$ *and* $P2 \geq MinDist$ **then**
            $CandidatePoint \leftarrow P1$
        end
    end
end
return list of $CandidatePoints$

3.6 Ranking Picking Points

To increase the success rate of grasps, the filtered point clusters are ranked on the basis of their stability. The primary factor influencing grasp success is the minimization of external forces, with the most significant external force being the maximum torque exerted by the object at the contact point.

$$T_{max} = d_t \cdot m \cdot g$$

Here, d_t represents the orthogonal distance from the point to the center of mass along the point normal, m is the mass of the object, and g is the acceleration due to gravity. Lower torque values indicate a reduced likelihood of destabilizing forces during lifting.

3.7 Export Structure

The export structure contains the essential components required to accurately represent the processed data, enabling the robot to perform the necessary movements and alignments for successful grasping. It includes the position, quaternion orientation, and associated ranking scores represented by the maximum torque.

Additionally, the export includes metadata related to both the object and the tools, enabling post-processing and better utilization of the data.

1. **Metadata**
   - **Export Time:** Timestamp of the data export.
   - **Object Properties:**
     - **Mass:** Total mass of the object.
     - **Center of Mass:** Position coordinates (x, y, z) representing the object's center.
     - **Matrix of Inertia:** A 3×3 matrix representing the object's inertia relative to its coordinate system.
   - **Descriptions:**
     - **Units:**
       - Length: mm
       - Mass: g
       - Torque: g*mm
     - **Coordinate Frame:** Step file origin.
     - **Position:** Global coordinate in the step file space.
     - **Normal:** Direction pointing towards the approach vector.
2. **Point Sets**
   - **Tool Specifications:**
     - Name: Tool identifier (e.g., Kenos KCS Foam-110).
     - Type: Type of tool (e.g., sponge, suction).

  - Maximum Width: Suction or contact width.
  - Maximum Penetration: Allowed depth for grasping.
  - Maximum Torque: Upper torque limit for stability.
  - Minimum Coverage: Required contact ratio for successful grasping.
- **Sampled Points:**
  - **Position:** Coordinates (x, y, z).
  - **Quaternion Orientation:** (x, y, z, w) representing the rotation.
  - **Maximum Torque:** Calculated torque for the point.

3.8 Evaluation Method

To assess the efficency of the generated grasps, a simulation in PyBullet was implemented. Algorithmically generated grasps were compared against a baseline of randomly sampled points on the surface of the mesh to validate results. The benchmarking ran on a diverse range of object to avoid biasing towards a specific shape or size.

3.8.1 Suction Cup Simulation

1. STEP to Standard Triangle Language (STL) Conversion
Each CAD file (`.step`/`.stp`) is converted to an STL mesh to ready it for simulation.

2. Mass & lift-force calculation
Compute the object mass as the mesh volume multiplied by material density (7850 kg/m^3 for steel), subject to a minimum of 0.01 kg. Calculate the downward force: F = mg. Optionally apply an additional lateral force to test stability under side loads.

3. Sampling "random" grasps Uniformly select faces from the mesh, then:

- Sample a point on each chosen face.
- Assign the face normal, a default orientation, the suction-cup diameter, and compute a torque limit from the vacuum pressure.

4. Jittering approach vectors
For randomly sampled grasps, tilt each approach direction up to $\pm 5^\circ$ around a random tangent axis to simulate misalignment.

5. Initial check test
Perform an initial contact check (fail if no collision).

6. Computing Effective Contact Area

Project each triangle onto the cup plane, keep those within the cup radius, and $\sum A_i|\cos\theta_i|$ to get A_{eff}.

7. Vacuum check & lift test
Compute the required vacuum pressure:
If the available vacuum p_{vac} is lower than p_{req}, the grasp fails immediately. Otherwise, lift the cup by the specified threshold in small increments, checking at each step for torque overflow or loss of contact.

8. Lateral Perturbation
Apply a constant sideways force equal to the same forcevalue (mg unless specified) over a series of steps. Fail if torque limit is exceeded (`torque_exceeded_perturb`) or contact is lost (`lost_at_end`).

9. Recording & summarizing results
Each trial logs one of five failure codes or "success," including relevant parameters.

Algorithm 11: Sample Random Grasps

Input: stl_path, json_path, max_samples, vacuum_pressure
Result: random_grasps list
mesh ← load_mesh(stl_path);
face_ids ← pick_random_faces(mesh, max_samples);
data ← read_JSON(json_path);
random_grasps ← [];
for each *suction_tool in tools* **do**
  diam ← compute_diameter(suction_tool);
  for each *face_id in face_ids* **do**
    pt ← sample_point_on_triangle(mesh.triangles[face_id]);
    normal ← mesh.face_normals[face_id];
    approach_vector ← jitter_vector(normal,5°);
    ori ← identity_quat();
    `/* Allow cup rotation to match surface */`
    pos ← scale(pt);
    random_grasps.append((pos,ori,normal,approach_vector,diam));
return random_grasps;

As shown in Algorithm 11, potential suction-cup grasps are generated by sampling surface triangles, computing approach vectors, and collecting pose information for each tool. First, the mesh is loaded and a fixed number of face IDs are sampled uniformly at random to ensure broad coverage of the object's surface. For each suction cup, we compute its physical dimensions (notably the cup diameter) and then iterate over each selected face. A point is sampled within the triangle, and the outward face normal at that point defines the nominal approach vector. We then apply a small angular jitter (max. 5°) around this normal to introduce variability and assess how stable the grasp remains under slight misalignment. Each grasp is saved as a tuple containing the world-frame position, a default orientation quaternion that

aligns the tool's axis with the surface normal while leaving its rotation about that axis unconstrained, the tool diameter, the original surface normal, and the jittered approach vector. Gathering all of these tuples yields a set of grasp poses that can be evaluated for stability in the next stages.

Algorithm 12: Compute Mass From STL.

Input: stl_path, density
Result: mass_kg
mesh ← load_mesh(stl_path);
mesh.scale(SCALE);
vol ← mesh.volume;
return max(vol × density, MIN_MASS);

Algorithm 12 begins by loading the mesh and converting its units from millimeters to meters via a uniform scaling factor. It then calculates the mesh volume, multiplies by the material density to derive the mass in kilograms, and enforces a minimum mass threshold to avoid zero values as thin meshes can suffer from rounding errors that would otherwise produce massless objects.

Algorithm 13: Compute Effective Contact Area.

Input: mesh, contact_center, normal, radius
Result: area
centroids ← mesh.triangles_center;
areas ← mesh.area_faces;
normals ← mesh.face_normals;
proj_pts ← project_onto_plane(centroids, contact_center, normal);
in_radius ← within_radius(proj_pts, contact_center, radius);
$cos\_theta \leftarrow \left| normals[in\_radius] \cdot \frac{normal}{\|normal\|} \right|$;
return $\sum(areas[in\_radius] \times cos\_theta)$;

Algorithm 13 computes an effective suction contact area used to evaluate hold strength. First, it projects each face centroid onto the plane at the candidate contact point. It discards any faces whose projected points lie outside the tool's radius. For each remaining face, it multiplies its area by the absolute cosine of the angle between the face normal and the reference normal. Finally, it sums those weighted areas to produce the effective contact area.

Algorithm 14: Run Suction Trial.

Input: stl, pos, normal, approach_vector, mass, lift_threshold, diam, penetration, vacuum_pressure, mu
Result: (success or error code)
start ← now();
mesh_tm ← load_mesh(stl);
mesh_tm.scale(SCALE);
grasp_ori ← compute_quaternion(normal, approach_vector);
eff_area ← compute_effective_area(mesh_tm, world_pt(pos, grasp_ori), approach_vector, diam/2);
`/* Compute torque limit based on suction cup area and vacuum */`
torque_lim ← vacuum_pressure × π × $(\frac{diam}{2})^3$;
if $mass \times gravity/eff\_area > vacuum\_pressure$ **then**
  return (False, insufficient_vacuum);
end
create_suction_cup(pos, grasp_ori, diam, mu);
if *not initial_contact* **then**
  return (False, no_initial_contact);
end
steps ← ⌊ lift_threshold / lift_rate ⌋;
for *step* ← 1 **to** *steps* **do**
  lift_cup(client, step);
  if *torque_exceeded(client)* **then**
    return (False, torque_exceeded_lift);
  end
  if *lost_contact(client)* **then**
    return (False, lost_during_lift, elapsed);
  end
end
return (True, none);

Algorithm 14 computes the grasp orientation quaternion from the surface normal and the approach vector. It then calculates the suction cup's torque limit using its diameter and the current vacuum pressure, ensuring all subsequent checks use accurate tool parameters. Next, the algorithm determines the effective contact area at the grasping point and evaluates whether the required lift force is within the vacuum's capacity.

$$F_{\text{lift}} \;=\; Pressure \;\cdot A_{\text{eff}}$$

If this feasibility check passes, the suction cup is instantiated and initial attachment is confirmed. The procedure then performs a controlled, incremental lift. After each step, it verifies that neither the torque limit is exceeded nor contact is lost. The trial terminates immediately with the appropriate error code upon any failure, or returns success once the target lift height is reached.

Algorithm 15: Batch Suction Verification.

```
Input: cad_files, rand_log
Result: Printed success rates, JSON
rand_succ ← 0;
for each (step, stl) in cad_files do
    name ← basename(step);
    json_path ← join(args.json_folder, name + "_points.json");
    if not exists(json_path) then
        continue
    end
    mass ← compute_mass_from_STL(stl, density);
    gens ← load_suction_grasps_from_JSON(json_path);
    for each (idx, grasp) in gens do
        gen_tot++;
        (success, code) ← run_suction_trial(stl, grasp, mass);
        gen_succ += success;
        record_result(args.log, name, idx, success, code, args.params);
    rands ← sample_random_grasps(stl, json_path, args.random,
     args.vacuum);
    for each (idx, grasp) in rands do
        rand_tot++;
        (success, code, dur) ← run_suction_trial(stl, grasp, mass);
        rand_succ += success;
        record_result(rand_log, name, idx, success, code, dur, args.params);
print_summary(gen_succ, gen_tot, rand_succ, rand_tot);
```

Algorithm 15 processes each CAD model in turn. For each model, it computes the object mass, loads all pregenerated suction grasps, runs the simulation for each pose while updating total and success counters, and logs each result. It then generates a new set of random grasps, repeats the same testing and logging steps, and finally reports the overall success rates for both the generated and random grasp sets.

3.9 Usage of AI

The project has used two large language models, ChatGPT and Deepseek[16],[17]. The models were used in order help brain ideas and to improve understanding of certain functions within used python libraries that lacked official documentation. They were also used to ensure that provided context was sufficient and as spelling control.

4

Results

The gathered results of this thesis consists of visualizations of the point generation process as well as the resulting output grasping points. Appendix 1 contains an example JSON file of the algorithms output when ran on object *Silver Box*. The results also include the estimated quality of grasping points from a lift simulation for grasping points generated for suction cups of varying sizes (25mm, 35mm and 50mm diameter). The estimated quality of generated grasping points is compared to the estimated quality of randomly generated grasping points both in an undisturbed lift, and one with an added disturbance. Objects were assumed to be made uniformly of steel, and mass was calculated based on objects' volume and the density of steel both in grasping point generation and simulation.

4.1 Picking Point Generation

The initially generated points on the object *Silver Box* are shown in figure 4.1. The points are uniformly distributed per individual surface and do not cluster near the center of surfaces. There are minor indications of clustering near edges of where two or more surfaces connect to each other (notice points near the 45 degree bends on the inside of the object).

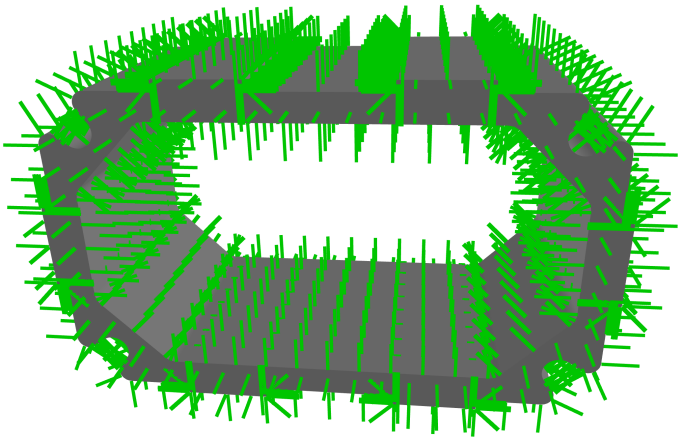


Figure 4.1: Initially generated points on object *Silver Box*.

4.1.1 Suction Cup

Figures 4.2 through 4.5 visualize the algorithm's different filtering stages using a suction cup with a 15 mm diameter and a maximum acceptable torque of 0.491 Nm on the object *Silver Box*.

Figure 4.2 shows the remaining points after removing points which do not satisfy the tool's maximum acceptable torque requirement. The remaining point's normal vector's all align towards, or away, (within acceptable margin) from the object's center of mass, meaning torque filtering was successful.

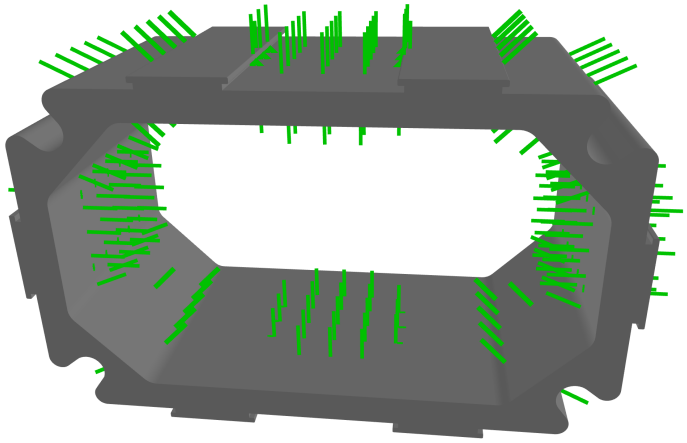


Figure 4.2: Remaining points after applying torque filtering with a maximum acceptable torque value of 0.491 Nm.

Removing points which underlying surfaces are not sufficiently flat and points which have their approach vector blocked instead gives results shown in figure 4.3. The filter has removed points generated on the curved edges and points deemed to be located too close to an edge.

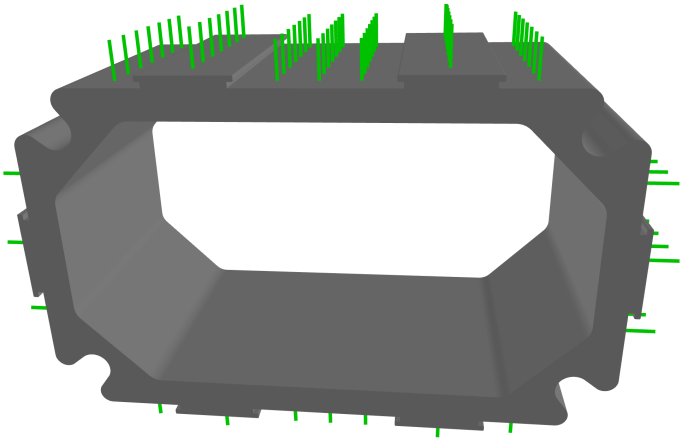


Figure 4.3: Remaining points after applying surface deviation and free approach filtering for a suction cup with 15 mm diameter.

Applying both filters in succession gives the results shown in figure 4.4. The filters operate together and the remaining points need to satisfy the conditions of both. The figure clearly shows that only points on sufficiently large, flat faces, with normal vectors pointed at the center of mass remain.

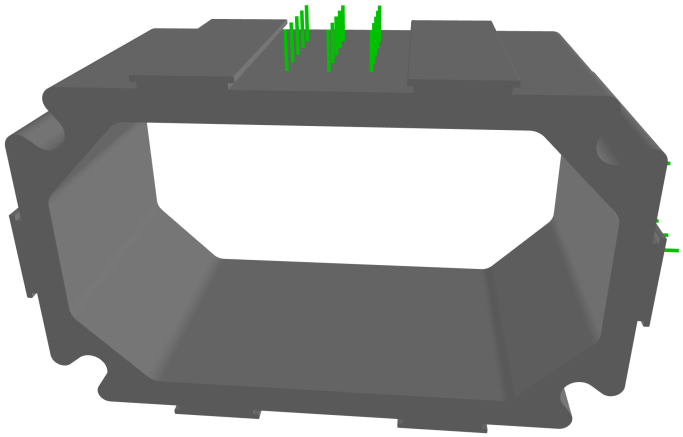


Figure 4.4: Remaining points after applying torque filtering, flat surface filtering, and free approach filtering for suction with a 15 mm diameter and a maximum acceptable torque values of 0.491 Nm.

Figure 4.5 shows the final remaining grasping points after applying best point clustering on grasping points visualized in figure 4.4. The filter keeps only the best grasping point, in regards to torque, for each cluster of grasping points in order to minimize excessive data export. In this case, where points are located on surfaces encapsulating the center of mass, the remaining grasping points are near the center of surfaces.

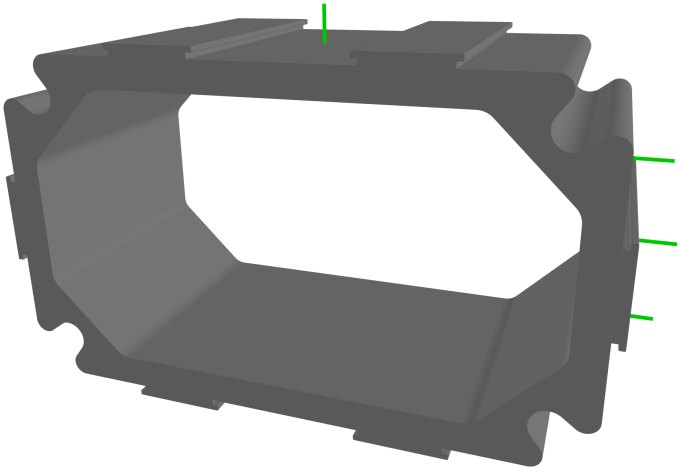


Figure 4.5: Final generated grasping points to be exported on object *Silver Box*.

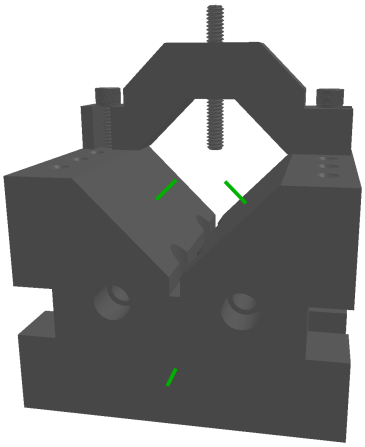
The algorithm has successfully produced eight (four grasping points obstructed from view) optimal grasping points on the *Silver Box* object for a 15 mm diameter suction

cup with a maximum acceptable torque of 0.491 Nm. The algorithm has generated no points on edges where the suction cup would not fit on the surface, or on uneven surfaces where surface contact is uncertain.

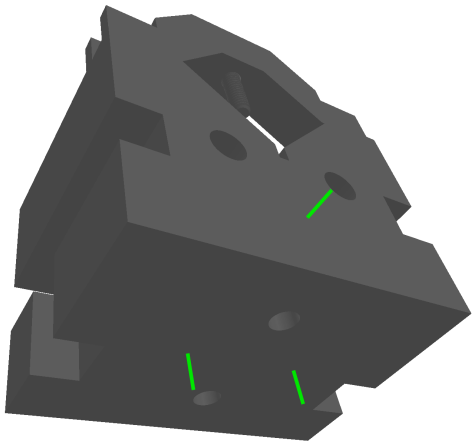
Plug Connector is an object that differs massively in size from *Silver Box* in that it is very small, weighing only 89 grams, and has very limited flat surfaces. Running the algorithm on *Plug Connector* with a 15 mm suction cup, results shown in figure 4.6, demonstrates a potential weakness of the algorithm. The flat surfaces are too small to fit a 15 mm diameter suction cup, resulting in the algorithm generating zero grasping points.

Figure 4.6: The object *Plug Connector* and its lack of generated grasping points using a 15 mm suction cup.

Running the algorithm with a 35 mm suction cup on the *V-Block* object, a much bigger and heavier object weighing 3.7 kg, results shown in figure 4.7, yields results in line with those on *Silver Box*. The algorithm has generated a total of six grasping points, all located on large flat surfaces, staying clear of any holes or surface deviations. The suction cup's maximum force (in other words: mass of the object) was left unspecified, meaning the suction cup was assumed to be able to handle the weight of 3.7 kg.



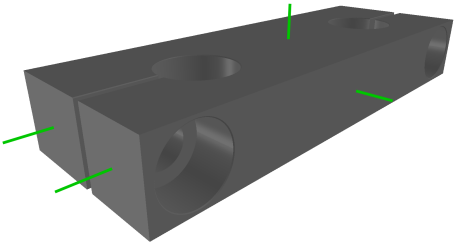
(a) Front of object *V-Block*, showing three generated grasping points.



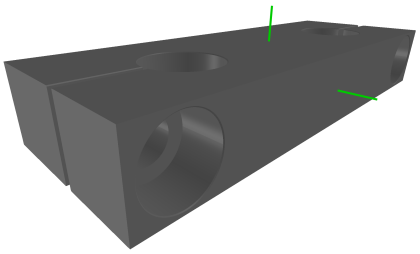
(b) Back of object *V-Block*, showing three additional generated grasping points.

Figure 4.7: Six points generated on CAD object *V-Block* using a 35 mm suction cup with a maximum acceptable torque value of 0.982 Nm.

Figure 4.8 shows how the resulting grasping points differ based on the maximum acceptable torque of the suction cup. Results shown in figure 4.8a and 4.8b use the same 15 mm diameter suction cup and only differ in maximum acceptable torque values. The lower value of maximum acceptable torque causes points on the short side of the object to be removed, as the points are unable to be placed sufficiently near the center due to a channel splitting the surface.



(a) Grasping points generated on object *Base-Mounted Dual Shaft Support* using a 15 mm suction cup with a maximum acceptable torque value = 0.196 Nm.



(b) Grasping points generated on object *Base-Mounted Dual Shaft Support* using a 15 mm suction cup with a maximum acceptable torque value = 0.049 Nm.

Figure 4.8: Difference in grasping point generation on object *Base-Mounted Dual Shaft Support* using different values of maximum acceptable torque.

4.1.2 Sponge Tool

The sponge tool does not require the entire tool footprint to fit on a surface and thus is able to generate grasping points closer to edges than a suction cup. It also has a maximum acceptable torque value significantly higher than what any grasping point generated on the objects showcased requires. Important algorithm parameters are the sponge tool's size and minimum acceptable surface coverage.

Visualized in figure 4.9 is the results of running the algorithm using a 110x110 mm sponge tool with a minimum acceptable surface coverage of 50%. The algorithm has removed points too close to corners, where surface coverage would drop below 50%.

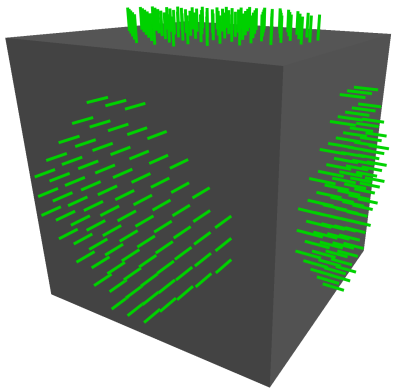


Figure 4.9: Result of generating grasping points on a 120x120x120 mm cube using a 110x110 sponge tool with a minimum acceptable surface coverage of 50%.

Figure 4.10 shows the resulting grasping points generated on object *Silver Box* using a 80x80 mm sponge tool with a minimum acceptable surface coverage of 60%. The grasping point clusters have not been removed to demonstrate the sponge tool's ability to handle uneven surfaces. The resulting grasping points after removing clusters are instead shown in figure 4.11.

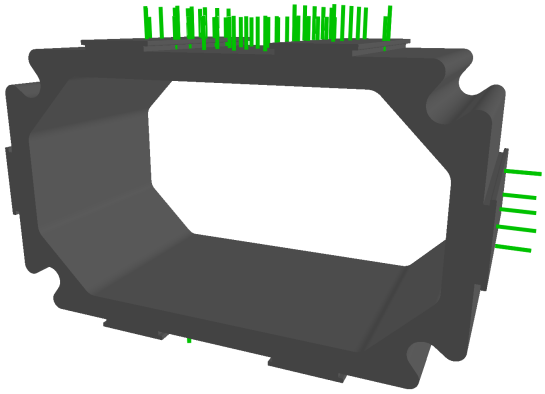


Figure 4.10: Result of generating grasping points on object *Silver Box* using a 80x80 mm sponge tool with a minimum acceptable surface coverage of 60% filtering clusters.

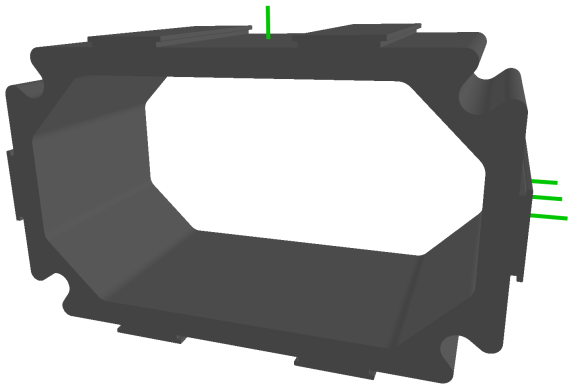


Figure 4.11: Result of generating grasping points on object *Silver Box* using a 80x80 mm sponge tool with a minimum acceptable surface coverage of 60%.

Using a small 30x30 mm sponge tool with minimum acceptable surface coverage of 50% the algorithm is able to generate grasping points on object *Plug Connector*, which it was unable to do using a 15 mm suction cup, as seen in figure 4.6. The resulting grasping points with cluster filtering turned off and turned on are shown in figures 4.12a and 4.12b respectively and demonstrate the sponge tool's non-reliance on flat surfaces.

(a) Grasping points generated without applying cluster filtering.

(b) Grasping points generated when applying cluster filtering.

Figure 4.12: The results of running the algorithm on object Plug Connector using a 30x30 mm sponge tool with a minimum acceptable surface coverage of 50%.

Figure 4.13 shows the resulting grasping points of running the algorithm on object *V-Block* using a 80x80mm sponge tool with a minimum acceptable surface coverage of 60%. Note how grasping points are located closer to edges than those generated by a suction cup, shown in figure 4.7.

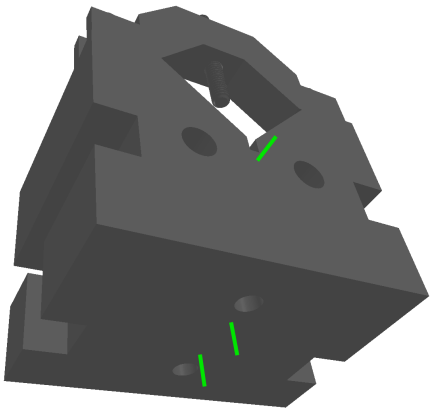


Figure 4.13: Result of algorithm on object *V-Block* using a 80x80 mm sponge tool with a minimum acceptable surface coverage of 60%.

In contrast to figure 4.8, figure 4.14 shows that the sponge tool does not generate grasping points on very small surfaces unless the sponge tool is also very small like in figure 4.12.

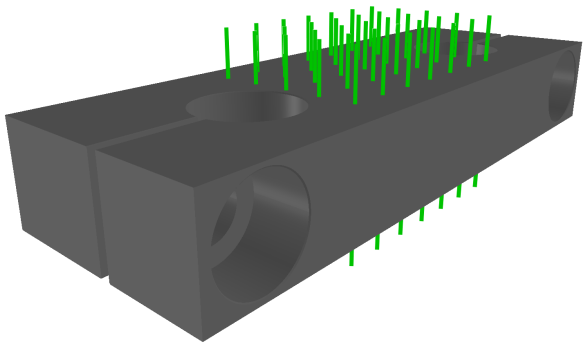
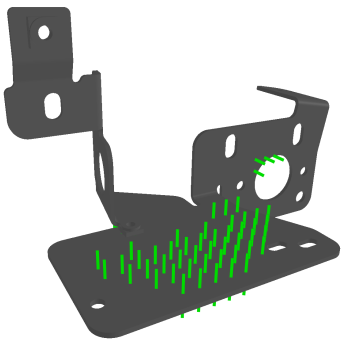
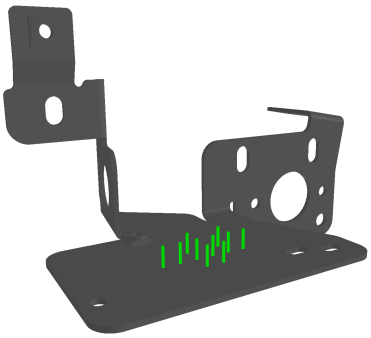


Figure 4.14: Result of algorithm on object *Base-Mounted Dual Shaft Support* using a 60x60 mm sponge with a minimum acceptable surface coverage of 50%. Cluster filtering is disabled.

Using a 80x80 mm sponge tool, the algorithm generates multiple grasping points on the flat surfaces of object *Black Plate*. Figure 4.15b and 4.15b shows the grasping points generated using minimum acceptable surface coverages of 50% and 70% respectively without applying cluster filtering. Grasping points generated using higher surface coverage requirements are located closer to the center of surfaces, with the exception of where the center would be close to a hole. Using 50% coverage allows for grasping points much closer to the edges of holes within a surface, evident by the four grasping points on the vertical surface on the right side of the object.



(a) Grasping points generated using a minimum acceptable surface coverage of 50%.

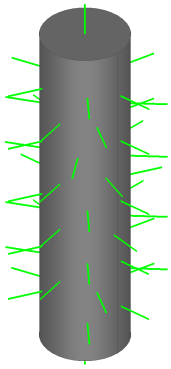


(b) Grasping points generated using a minimum acceptable surface coverage of 70%.

Figure 4.15: Generated grasping points on object *Black Plate* using a 80x80 mm sponge tool without applying clustering using differing values of minimum acceptable surface coverage.

4.1.3 Gripper Tool

Figure 4.16 shows the results of the algorithm using a 140 mm wide gripper on a 200 mm tall cylinder with a diameter of 50 mm. The algorithm has successfully generated grasping points on the curved surface of the cylinder, but perhaps an excessive amount. However, as demonstrated specifically by figure 4.16b, the algorithm is unable to generate grasping points on the curved surfaces should the cylinder contain a hole, indicating a potential issue.



(a) Grasping points generated on a 200 mm tall cylinder.

(b) Grasping points generated on a 200 mm tall cylinder with a hole.

Figure 4.16: Difference in grasping points generation for a 140 mm wide gripper on two 200 mm tall cylinders with diameters of 50 mm depending on whether or not there is a hole.

Figure 4.17 shows the grasping points generated using a 50 mm wide gripper on object *Silver Box*. The algorithm has successfully identified where there are matching parallel surfaces and generated grasping points in those location.

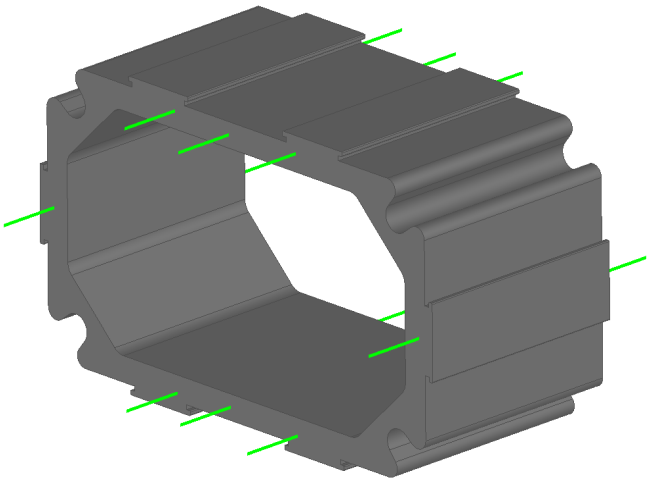


Figure 4.17: Grasping points generated on object *Silver Box* using a 50 mm wide gripper.

Running the algorithm uisng a 140 mm wide gripper on object *Black Plate* gives results shown in figure 4.18. Unlike results shown in figure 4.17, however, the algorithm has now generated two grasping points on the top flat surface with vertical approach vectors. These grasping points are not anomalies as the gripper reaches across the length of the surface, evident by their disappearance when using a smaller 30 mm wide gripper as shown in figure 4.19, but they may not be considered optimal. Comparing the results to those from a sponge tool in figure 4.15 clearly illustrates the difference in approach strategy between grippers and suction based tools.

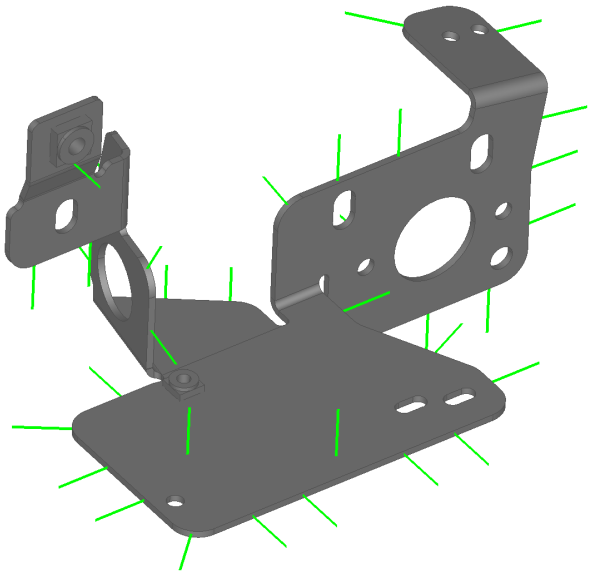


Figure 4.18: Grasping points generated on object *Black Plate* using a 140 mm wide gripper.

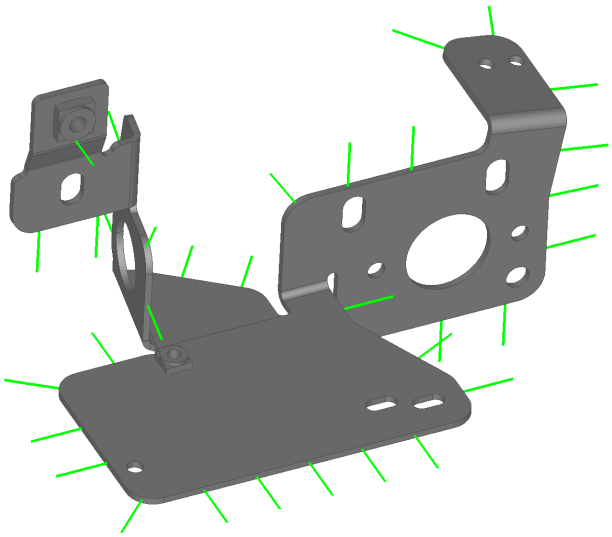


Figure 4.19: Grasping points generated on object *Black Plate* using a 30 mm wide gripper.

Generating grasping points using a 140 mm wide gripper on object *Base-Mounted Dual Shaft Support* generates a potential anomaly. One of the grasping points is located very close to an edge which is not optimal. It is also located close to a hole which the algorithm is known to struggle with, as demonstrated in figure 4.16b.

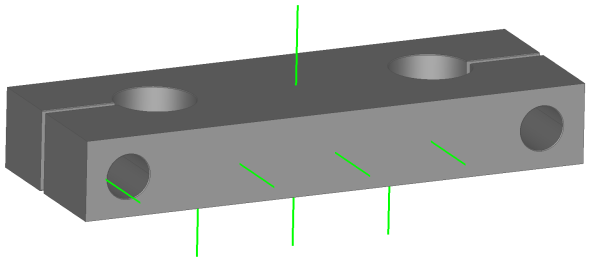


Figure 4.20: Grasping points generated on object *Base-Mounted Dual Shaft Support* using a 140 mm wide gripper.

4.2 Suction Cup Simulations

This section details the results of benchmarking simulations using pybullet. Tables show the amount of generated grasping points N, and the success rates using

the generated grasping points. The algorithms generated points are compared to grasping points that have been randomly generated, as described in section 3.8.1.

In order to simplify references to specific object, abbreviations were created for all objects used in the simulations. These abbreviations are listed in table 4.1.

Table 4.1: Abbreviations of object names.

Object name	**Object abbreviation**
Granite Flat-Surface Plate	GFSP
Base-Mounted Dual Shaft Support	BMDSS
Extra-Wide Sleeve Bearing Carriage	EWSBC
Aluminum Bolt-Together Fitting	ABTF
Linear Bearing Housing	LBH
Big Plate	BIP
Black Plate	BLP

These objects were partly supplied by Volvo Trucks, with the remainder randomly selected to ensure coverage of a broad range of geometries, sizes, and surface characteristics. Tables 4.2 and 4.3 shows the results of simulations performed with the suction cups.

Table 4.2: Success rates for different objects for both algorithmically and randomly places grasping points during simulations in pybullet with simple gravity.

	Algorithm		**Random**	
Object	**N points**	**Success rate [%]**	**N points**	**Success rate [%]**
GFSP	20	0	192	5.24
BMDSS	6	100.0	192	21.4
EWSBC	12	83.3	192	46.9
ABTF	4	50.0	192	24.5
LBH	15	13.3	192	22.4
BIP	8	75.0	192	44.3
BLP	12	66.7	192	19.3

Table 4.3: Success rates for different objects for both algorithmically and randomly placed grasping points during simulations in pybullet with simple gravity combined with equal force applied horizontally.

	Algorithm		Random	
Object	**N points**	**Success rate [%]**	**N points**	**Success rate [%]**
GFSP	20	0	192	1.0
BMDSS	6	100.0	192	11.5
EWSBC	12	75.0	192	37.0
ABTF	4	50.0	192	23.4
LBH	15	13.3	192	14.6
BIP	8	75.0	192	47.4
BLP	12	66.7	192	28.1

In Table 4.2, algorithm-generated grasping points were compared to randomly sampled grasping points in undisturbed lift simulations. For objects with large, flat surfaces (BMDSS, EWSBC, BLP, and BIP) the algorithm produced between 4 and 20 grasp candidates, achieving success rates of 50%–100% (with an outlier at 13%). The random sampler ran 192 trials per object and yielded lower success rates (5.24%–46.9%). On the heaviest object, GFSP, the algorithm generated 20 grasping points and achieved a 0% success rate. Random grasping points achieved a 5.24% success rate.

Under a 20 N lateral perturbation (see Table 4.3), success rates decreased for both methods but the advantage of the geometry-based filter remained. The algorithm maintained 75% success on EWSBC (12 grasping points) and 100% on BMDSS (6 grasping points), whereas random sampling on those objects dropped to 37.0% and 11.5%, respectively. On GFSP, the success rate of random grasping points dropped to 1%.

In summary, the algorithm consistently yielded fewer but far more reliable grasp candidates than random sampling, especially under lateral perturbations.

4.2.1 Complexity Analysis

To accurately assess the performance of each tool algorithm, two types are analysis are performed. First, a theoretical Big-O analysis is performed on each tool algorithm to estimate time complexity.

Next an experimental analysis is performed to validate the theoretical estimation. The analysis is conducted using varying point densities and sampling rates to observe their impact on computational performance.

Although the number of triangles could theoretically influence the complexity, varying triangle count independently is impractical. This is because the number and size of triangles directly affect the generated points. The number of generated points can be estimated as:

$$point\ count = \frac{surface\ area}{(point\ distance)^2} \tag{4.1}$$

Due to this relationship, the experimental analysis focuses on varying point density and sampling rate rather than the triangle count.

To simplify the analysis, the following variables are defined:

- N: Number of points generated by the algorithm.
- T: Number of triangles in the mesh used for surface evaluation.
- S: Sampling Rate of the height map.

Both the **Suction Cup Algorithm** and the **Sponge Tool Algorithm** uses a combination of two filters: the excessive torque filter and the surface evaluation filter. The final time complexity of both algorithm's is dominated by the height map generation step, resulting in:

$$O(T \times N \times S^2)$$

To experimentally evaluate the performance of the algorithm's, an analysis was conducted using a fixed sampling rate of 20. The results are presented in figure 4.21, illustrating the linear relationship between point count and execution time.

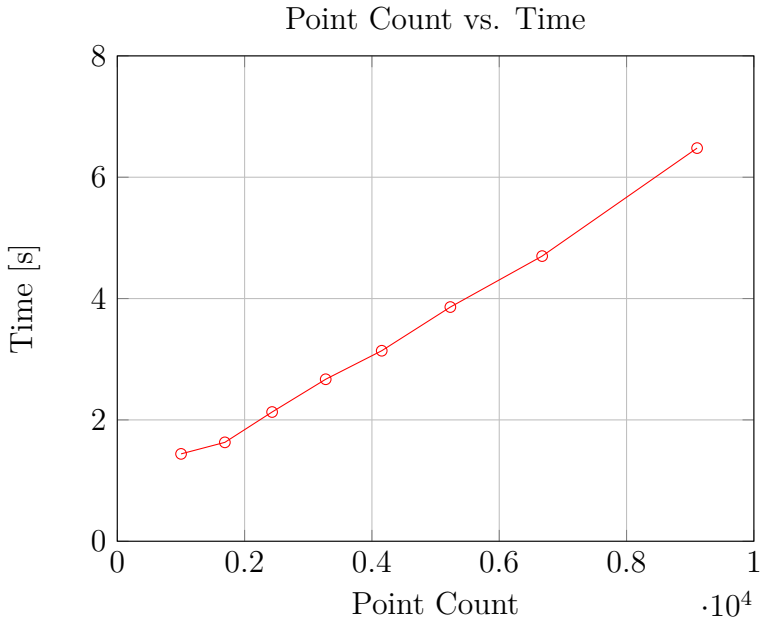


Figure 4.21: Relationship between Point Count and Time.

To further assess the impact of sampling rate on the algorithm's performance, the execution time was recorded for various sampling rates while keeping the point

density constant. The results are shown in figure 4.22, demonstrating the quadratic relationship between sampling rate and execution time.

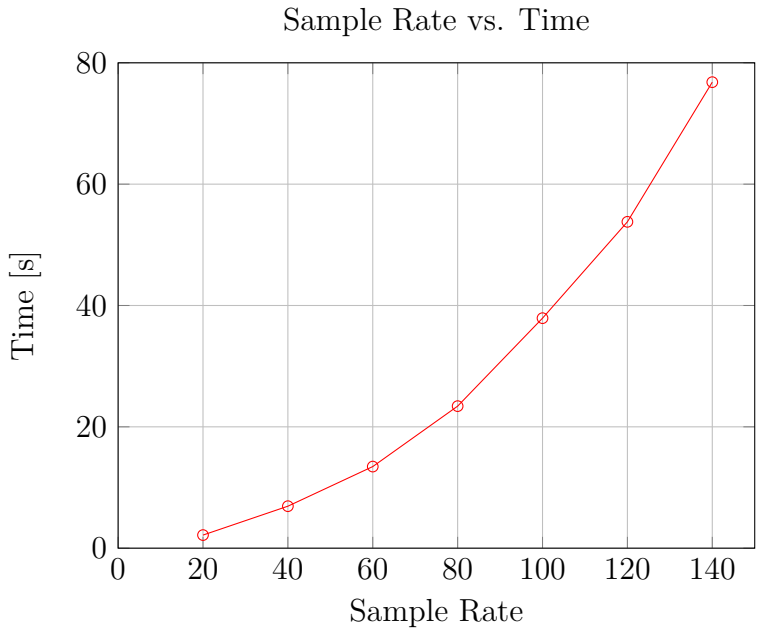


Figure 4.22: Relationship between Sample Rate and Time.

The **Sponge Tool Algorithm** follows a similar structure to the Suction Cup Algorithm, where the computational cost is also dominated by the height map generation step, resulting in the same complexity:

$$O(T \times N \times S^2)$$

The **Gripper Tool Algorithm** works differently from the suction and sponge tool algorithms. No height map is used, and the shape analysis is done on parametric form. The complexity is thus dependent only on point count. One of the steps, the clustering filter is quadratic in complexity, however due to being last in the filtering process, the overall runtime is instead dominated by the very high linear time factor of the shape analysis as shown in figure 4.23 resulting in the effective complexity:

$$O(N)$$

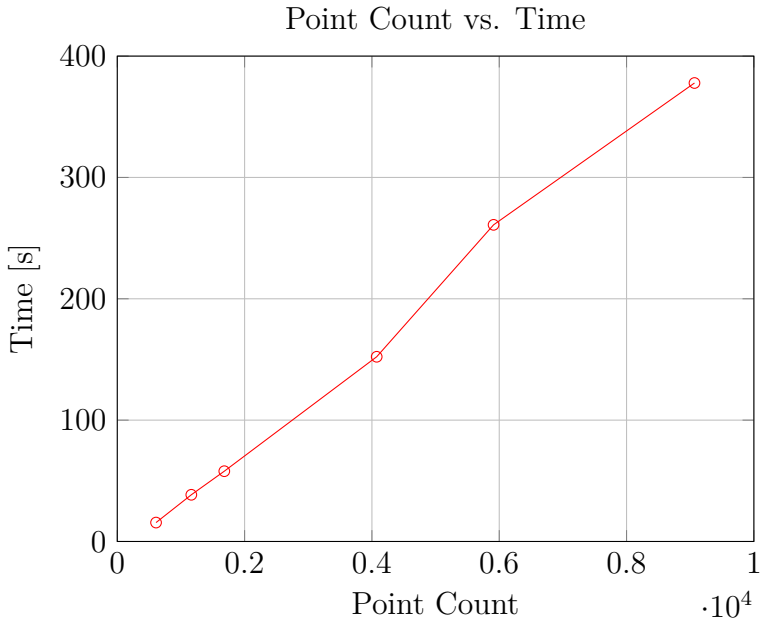


Figure 4.23: Relationship between Point Count and Time.

5

Discussion

The algorithm produces good results for suction cups and sponge tools and decent results for the gripper tool. This is expected as the gripper is inherently more complex, in part due to suction cups and sponge tools only needing a single contact surface, while the gripper needs two.

In regards to suction cups, the algorithm is able to properly identify points which exceed the maximum torque limit and points where the suction cup would not fit on the surface. It also successfully removes points which lack a free approach vector. Visual inspection of results shows no unexpected anomalies, which is a good indication about the real-life viability of generated grasping points. However, while the filters, given correct parameters in regards to maximum lifting force, torque, and size constraints, should only ever allow viable grasping points to be exported, there may be real-life factors which the algorithm is unable to account for. This real-life unpredictability, however, is mainly a limitation with not having access to any live sensor data from the robots' computer vision system (in other words: predefining grasping points based on static data), and not within the algorithm itself. A limitation that also affects manually placed grasping points.

Sponge tools also produce good results. The algorithm correctly identifies points which are not approachable and points which do not provide sufficient surface contact given a sponge tool's acceptable surface height deviation. Similar to suction cups, the sponge tool has generated no obvious anomalies, which should indicate high real-life success rates. However, once again ,much like the suction cup, real-life unpredictability is something which cannot be accounted for due to the nature of the algorithm.

Gripper tools produce reasonable grasping points on real-life objects such as *Black Plate* and *Silver Box*. That said, the algorithm clearly presents issues when generating grasping points using gripper tools on objects that have significant openings, such as objects open on both ends or those with a sufficiently deep opening on one side. This issue is likely due to the *Closest Orthogonal* filter, described in algorithm 8. In code, the filter operates by looking for the closest surface from the generated center point between point pairs when searching for viable gripping directions. This works for solid objects, as reaching a surface from within a solid means exiting the solid. In the case of previously referred to as open-ended objects, the closest surface could still be on the inside of the object. Points generated on inside surfaces are

correctly removed by subsequent filters as the points are inaccessible, meaning the algorithm does not produce invalid points. This is a major issue that could significantly impact real-life performance. A minor issue is the perhaps excessive number of generated points. However, as the final grasping points are defined very similarly to grasping points generated by suction cups and sponge tools, one could theoretically apply the cluster filtering from suction cups and sponge tools on grasping points generated for the gripper without issues.

5.1 Observed behavior

The point generation does not include any randomness, meaning it is reproducible for each object. This means that generated points will be consistent and grasping points which allow robots to successfully grasp objects will remain even if the algorithm is run again after adding a new tool. It also means that grasping points the algorithm has potentially missed will be missed again. This could be combated via increasing the sampling rate when generating points, reducing the risk of missing potentially viable grasping points. However, this would drastically increase the computational resources required by the algorithm.

One potential disadvantage of the algorithm is that the algorithm is very strict in what it considers acceptable grasping points. This might mean that on some objects, the algorithm may not produce any grasping points at all. This could be solved by always exporting at least one grasping point, which would be the most optimal grasping point given the circumstance, even if that grasping point does not reach an acceptable quality. As randomly generated grasping points failed to achieve a 50% success rate on even the simplest of objects, it was ultimately decided that even the single most optimal grasping point should only be generated if it reaches acceptable quality. This decision was also made due to safety concerns of collaborative robots potentially dropping objects while operating close to employees.

5.2 Simulation

As some of the objects selected were meant to challenge the algorithm, some objects could not be handled with the selected tool set. Specifically, some objects required smaller suction-cups or a significantly higher vacuum pressure. However observing the randomly generated grasping points success rate shows that some objects were valid in the simulation despite very likely not being possible in practice, shows bias or inaccuracies of the simulation.

While generating the grasping points, the maximum lifting force was left unspecified (i.e. infinite), meaning it is assumed that tools could handle the weight of any object. This is unlikely to accurately represent real-life scenarios, but had to be done as the CAD objects lacked material data, meaning weight could not be accurately determined. Objects were therefor assumed to be uniform and densely packed steel, meaning the center of mass, which is important during torque calculations, does not account for any potential material differences in a single part. While it could be

argued that parts typically consist of a single material, and that multi-material parts are in actuality a combination of many single-material parts, it cannot be taken for granted that parts only consist of a single material.

5.2.1 Biases For Random Grasping Point Generation

The comparison between the algorithm's generated grasping points and grasping points that are randomly generated includes factors that affect that reliability of the results. Such factors include the randomness itself, as well as how the simulation handles certain points and surfaces.

5.2.2 Biases From Randomness

The algorithms performance is deterministic for all objects, while random grasping points are not. While an attempt was made to minimize the effect of random variance by running the simulation multiple times, some results do not correlate with what is expected.

Another result of randomized grasping point generation, grasping points are more likely to be placed on larger surfaces. This gives way to larger possibilities for the randomized process to generate successful grasping points. This skews the results when comparing with the developed algorithm since the algorithm clustered grasping points on the same surfaces together.

5.2.3 Biases From Mesh Parsing

The PyBullet library's use of meshes to represent the objects opens up for certain issues. During the grasping point generation for the randomized selection, some grasping points are likely to be placed on small surfaces. When the picking simulation is run on the generated grasping points, the points on small surfaces are disregarded by the simulation and are not counted. This means that the result of these small area grasping points are withheld from the results. Combined with such small areas being very likely to fail due to size differences with respect to the suction cup, this results in a bias where some grasping points likely to fail are disregarded making success rates from the random selection higher than they should.

5.3 Discussion of Method

Over the span of this project, parts of the method were rethought and refined as further insights into the unique challenges grew. The framework established at the beginning, shown in section 3.1, worked as a solid foundation, but some elements needed to be reworked over the scope of the project.

One of the changes during development was the implementation of meshes for both the suction cups and the sponge tools. The algorithm initially started out using only parametrized calculations and did not utilize meshes in order to provide grasping points with zero data loss or error margins. However, as the algorithm developed it

quickly became very slow and resource intensive, reaching unacceptable run-times for even the simplest of objects and occasionally crashing due to insufficient memory. Meshes would be able to fix this issue, albeit at the cost of minor data loss, and was deemed the better alternative. Once the code for suction and sponge tools had been reworked to utilize meshes the code not only became much faster, but also massively reduced the number of lines needed for achieving the same functionality. An attempt was made to migrate the gripper tool to mesh based analysis as well, however, it proved to be a significant challenge. While the minor data loss did not affect suction and sponge tools, it did affect the gripper, as some of the data loss when converting to a mesh includes the relationships between surfaces. The attempt was therefor abandoned, and the gripper remains on parametrized calculations, meaning it is much slower than running the algorithm with only suction and sponge tools.

The choice of using a rule based implementation has shown great advantages. One such has been the deterministic character of the developed algorithm which greatly helped during development. If a machine learning approach was used, fixing of errors and testing of general behavior would have been significantly more difficult. The algorithm would have been working as a black box without the ability for the developer to follow the progress in the same respect. That said, had more labeled data been readily available, a machine learning approach would have been given greater consideration.

5.3.1 Tool Parameter Simplification

To achieve a more general solution, both the suction cup and the sponge tool were simplified into two area footprints. The suction cup footprint was represented as a circle with a radius of r_f, and the sponge tool footprint is represented as a rectangle with dimensions w_f and h_f.

This simplification may result in the loss of some valid grasping points. However, all generated grasping points will remain within the valid range of the tool, ensuring compatibility with the tool's constraints.

5.3.2 Data Loss During Height Map Construction

During the process of constructing the local height map for points, two key parameters are specified: the sampling rate and the local area size. Since all tools use the same sampling rate, the size of the evaluated height map is determined by the maximum size of the tool. Consequently, the size of each element within the height map is calculated as:

$$elem\_size = \frac{tool\_size}{sampling\_rate} \tag{5.1}$$

This relationship implies that the tool size directly influences the resolution of the height map. As the tool size increases, the resolution decreases, which may result in data loss. Therefore, consideration of the tool size and sampling rate is required to ensure sufficient data for accurate surface evaluation.

5.3.3 Grasping Point Density Considerations

A potential issue when generating points is achieving an appropriate point density. If the point density is too low, there is an increased risk of data loss, while an excessively high point density can lead to computational overload.

As mentioned in the complexity analysis, the estimated point count is given by:

$$point\_count = \frac{surface\_area}{point\_distance^2}$$

This relationship highlights the need to carefully consider the object size and the point distance to ensure that sufficient number of points are generated without creating excessive computational time.

5.4 Further Development

There first steps to be taken in future development is to make the algorithm more efficient and to solve some of the edge cases that the algorithm is currently unable to properly process.

5.4.1 Uniformity in Integration

The algorithm for the gripper tool has been developed separately from the algorithm for the suction cup and the sponge tool. Ideally, the gripper should be integrated in a uniform manner. An example of this is the collision filter described in algorithm 9 which could be replaced by a z-map analysis, as it is done in the suction cup and sponge tool algorithms. This would primarily be to reduce unnecessary code complexity but also improve code execution speed.

5.4.2 Hollow Objects

The gripper tool currently cannot find the closest orthogonal point on an exit surface when analyzing hollow objects as explained above. This could possibly be solved by finding the closest point on several edges of the intersecting surface and discarding the points not located on an exit surface. Invalid points could reasonably be identified by checking if the shape's surface normal aligns with the vector from the origin point to that surface point.

5.4.3 Gripper Contact Surface Area Analysis

The gripper tool finds its grasping point by finding a point pair on the surface on the shape, however the surrounding area to these surface points is not analyzed. This means that the grasping point viability is unaffected by the size of the surface area that the gripper will touch. A further improvement step would therefore be to make a z-map surface analysis of the surfaces on which the point pair lies, with the footprint of the gripper arm shape.

5.4.4 Simulation

It would be advised to attempt running a simulation in ROS2 with Gazebo for a more robust simulation and more realistic results.

6

Conclusion

This project focused on the automated generation of grasping points on 3D CAD models for robotic manipulation, using a tool-specific analysis approach. The goal was to design and implement an algorithm capable of processing STEP files and producing accurate, usable grasping points for various end-effectors. This goal has been largely achieved. One known limitation exists for gripper tools when handling open-ended or highly concave objects, where grasp point generation is less reliable. However, suction and sponge tools perform well across a wide range of geometries.

The thesis set out to determine whether such an algorithm could be developed to reliably analyze object geometry, identify optimal grasping points across a range of objects and a set of tool types and finally rank them. The final algorithm is relatively robust and performs consistently in general use, demonstrating the practicality of the proposed approach.

The solution uses a combination of parameterized and mesh-based geometric analysis, leveraging Python libraries to extract and evaluate surface features from CAD models. For each supported tool type —gripper, suction, and sponge— a customized filtering process generates and refines points based on relevant criteria such as surface curvature, area, and orientation. These points are then ranked by expected torque during lifting to produce the most stable grasps. The system outputs this information as structured JSON files, supporting easy downstream integration.

Importantly, the algorithm is not limited to a fixed set of objects or tools. It can be generalized to new, previously unseen models, and allows for the addition of new tools within existing families, provided that the appropriate parameters are defined.

In summary, this project demonstrates that automated tool-aware grasp point generation from CAD data is both feasible and practical. The algorithm provides a strong foundation for future work in robotic manipulation and industrial automation, where adaptability to varying tools and objects is critical for real-world application.

Bibliography

[1] R. Tedrake, *Robotic Manipulation: Perception, Planning, and Control*. 2024. [Online]. Available: http://manipulation.mit.edu.

[2] A. L. S. Mota and R. G. Lins, "Production of customized commercial vehicles in assembly line based on modified-to-order demands: A novel method and study case," *International Journal of Production Economics*, vol. 282, Apr. 2025. DOI: 10.1016/j.ijpe.2025.109535.

[3] "International implementation of vehicle emissions standards," *Climate Change Authority*, 2025. [Online]. Available: https://www.climatechangeauthority.gov.au/reviews/light-vehicle-emissions-standards-australia/international-implementation-vehicle-emissions.

[4] E. Garcia, M. A. Jimenez, P. G. D. Santos, and M. Armada, "The evolution of robotics research," *IEEE Robotics & Automation Magazine*, vol. 14, no. 1, pp. 90–103, 2007. DOI: 10.1109/MRA.2007.339608.

[5] "Serial production starts for volvo trucks' new generation of heavy-duty trucks," *Volvo Trucks*, Mar. 2021. [Online]. Available: https://www.volvotrucks.com/en-en/news-stories/press-releases/2021/mar/serial-production-starts.html.

[6] D. Buchholz, *Bin-Picking: New Approaches for a Classical Problem*, 1st ed. Springer Cham, 2016, pp. xv+117. DOI: 10.1007/978-3-319-26500-1.

[7] E. Eros, Collaborative bin picking by an industrial robot, [Photograph].

[8] F. Spenrath and A. Pott, "Gripping point determination for bin picking using heuristic search," *Procedia CIRP*, vol. 62, pp. 606–611, 2017. DOI: 10.1016/j.procir.2016.06.015.

[9] A. Sintov, R. J. Menassa, and A. Shapiro, "Ocog: A common grasp computation algorithm for a set of planar objects," *Robotics and Computer-Integrated Manufacturing*, vol. 30, no. 2, pp. 124–141, 2014. DOI: 10.1016/j.rcim.2013.09.004.

[10] A. Lobbezoo and H. Kwon, "Simulated and real robotic reach, grasp, and pick-and-place using combined reinforcement learning and traditional controls," *Robotics*, vol. 12, 2023. DOI: 10.3390/robotics12010012.

[11] A. T. Miller and P. K. Allen, "Graspit! a versatile simulator for robotic grasping," *IEEE Robotics & Automation Magazine*, vol. 11, no. 4, pp. 110–122, 2004. DOI: 10.1109/MRA.2004.1371616.

[12] S. A. Wright and A. E. Schultz, "The rising tide of artificial intelligence and business automation: Developing an ethical framework," *Business Horizons*, vol. 61, no. 6, pp. 823–832, 2018. DOI: 10.1016/j.bushor.2018.07.001.

[13] I. Godin, F. Kittel, Y. Coppieters, and J. Siegrist, "A prospective study of cumulative job stress in relation to mental health," *BMC Public Health*, vol. 5, p. 67, 2005. DOI: 10.1186/1471-2458-5-67.

[14] T. Paviot, *Pythonocc*, version 7.8.1.1, Dec. 2022. DOI: 10.5281/zenodo.3605364.

[15] Dawson-Haggerty et al., *Trimesh*, version 4.6.6, Dec. 8, 2019. [Online]. Available: https://trimesh.org/.

[16] ChatGPT (version Apr. 16), *OpenAI*, [Large Language Model], 2025. [Online]. Available: https://openai.com/.

[17] DeepSeek (version Mar. 24), *DeepSeek*, [Large Language Model], 2024. [Online]. Available: https://www.deepseek.com/en.

A

Appendix 1

Example JSON output file


```
{
    "cad_file_path": "cads/silver_box.stp",
    "export_time": "Mon May 12 09:23:25 2025",
    "meta": {
        "mass": 2248.8669433722102,
        "centre_of_mass": [
            0.2004983329373491,
            -0.06882585075691472,
            -44.99988736868933
        ],
        "matrix_of_inertia": [
            [
                461871942.6680981,
                2.5011104298755527e-11,
                2524.837834184058
            ],
            [
                2.5011104298755527e-11,
                1147387894.7232032,
                3.4924596548080444e-09
            ],
            [
                2524.837834184058,
                3.4924596548080444e-09,
                919474508.796606
            ]
```

```
        ],
        "descriptions": {
            "units": {
                "length": "mm",
                "mass": "g",
                "torque": "g*mm"
            },
            "coordinate_frame": "step_file_origin",
            "tool_origin": "step_file_origin",
            "position": "Global coordinate using step file
            ↪  coordinate space.",
            "normal": "Pointing towards the approach direction."
        }
    },
    "point_sets": {
        "tool_0": {
            "specifications": {
                "name": "Piab piGRIP S-50",
                "type": "suction",
                "max_width": 50.0,
                "max_penetration": 0.5,
                "max_torque": 182793
            },
            "points": {}
        },
        "tool_1": {
            "specifications": {
                "name": "Piab piGRIP S-35",
                "type": "suction",
                "max_width": 35.0,
                "max_penetration": 0.5,
                "max_torque": 61486
            },
            "points": {
                "point_0": {
                    "position": {
                        "x": 0.20123690728881627,
                        "y": -0.06882585075691594,
                        "z": -90.00000000000009
```

```
                    },
                    "quaternion": {
                        "x": 1.0,
                        "y": 0.0,
                        "z": 0.0,
                        "w": 6.123233995736766e-17
                    },
                    "max_torque": 1.6609554442371108
                },
                "point_1": {
                    "position": {
                        "x": 0.20123690728881627,
                        "y": -0.06882585075691594,
                        "z": -7.815970093361102e-14
                    },
                    "quaternion": {
                        "x": 0.0,
                        "y": 0.0,
                        "z": 0.0,
                        "w": 1.0
                    },
                    "max_torque": 1.6609554442371108
                }
            }
        },
        "tool_2": {
            "specifications": {
                "name": "Piab piGRIP S-15",
                "type": "suction",
                "max_width": 15.0,
                "max_penetration": 0.5,
                "max_torque": 6496
            },
            "points": {
                "point_0": {
                    "position": {
                        "x": 79.20123690728882,
                        "y": -0.06882585075691594,
                        "z": -45.00000000000008
```

```
                },
                "quaternion": {
                    "x": 0.0,
                    "y": 0.7071067811865475,
                    "z": -0.0,
                    "w": 0.7071067811865476
                },
                "max_torque": 0.25329283153238535
            },
            "point_1": {
                "position": {
                    "x": 0.20123690728881627,
                    "y": -0.06882585075691594,
                    "z": -90.00000000000009
                },
                "quaternion": {
                    "x": 1.0,
                    "y": 0.0,
                    "z": 0.0,
                    "w": 6.123233995736766e-17
                },
                "max_torque": 1.6609554442371108
            },
            "point_2": {
                "position": {
                    "x": 0.20123690728881627,
                    "y": -0.06882585075691594,
                    "z": -7.815970093361102e-14
                },
                "quaternion": {
                    "x": 0.0,
                    "y": 0.0,
                    "z": 0.0,
                    "w": 1.0
                },
                "max_torque": 1.6609554442371108
            },
            "point_3": {
                "position": {
```

```
                        "x": -78.79876309271118,
                        "y": -0.06882585075691594,
                        "z": -44.85000000000008
                    },
                    "quaternion": {
                        "x": 0.0,
                        "y": -0.7071067811865475,
                        "z": 0.0,
                        "w": 0.7071067811865476
                    },
                    "max_torque": 337.07674867429597
                }
            }
        }
    }
}
```

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS