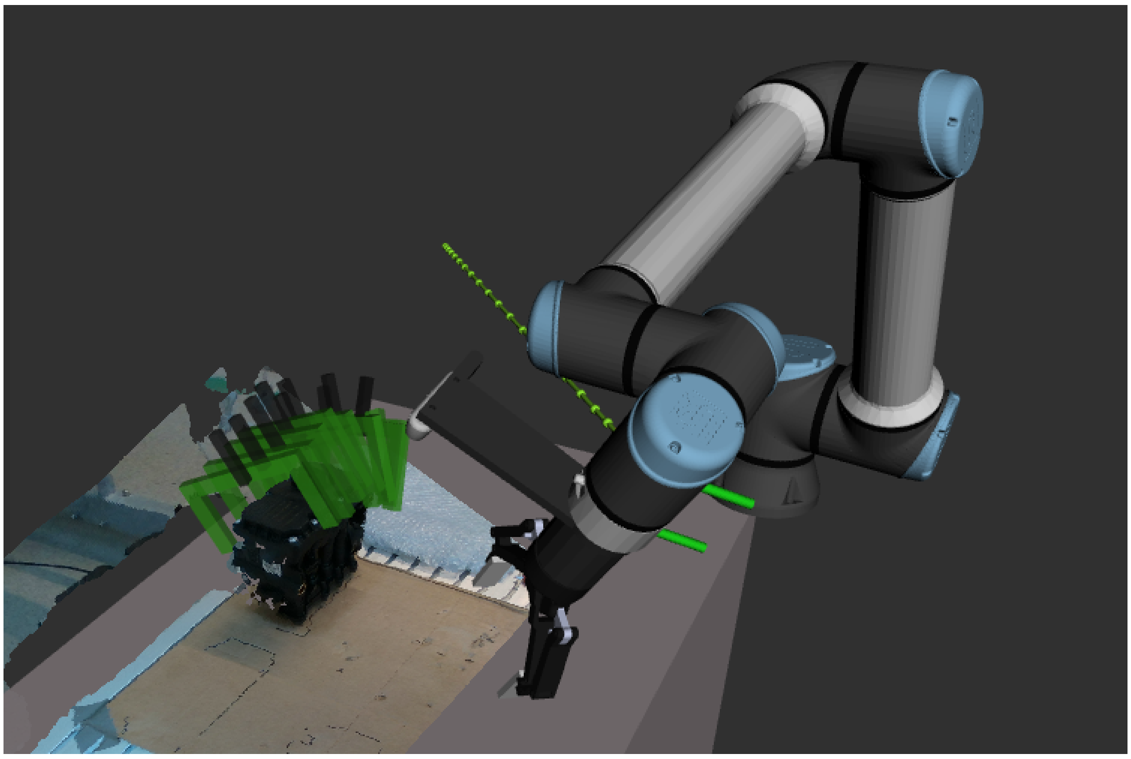
CHALMERS
UNIVERSITY OF TECHNOLOGY



Grasp Synthesis Methods on Known Objects for Bin-Picking

A comparison on deep-learning based and analytical-model based 6 DoF grasp pose synthesisers

Master's thesis in Systems, Control, and Mechatronics

SEVAG TAFNAKAJI
SIMON WIDERBERG

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Grasp Synthesis Methods on Known Objects for Bin-Picking

A comparison on deep-learning based and analytical-model based 6 DoF grasp pose synthesisers

SEVAG TAFNAKAJI
SIMON WIDERBERG

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

Grasp Synthesis Methods on Known Objects for Bin-Picking
A comparison on deep-learning based and analytical-model based 6 DoF grasp pose synthesisers
Sevag Tafnakaji
Simon Widerberg

Supervisor: Atieh Hanna, Volvo GTO
Examiner: Karinne Ramirez-Amaro, Electrical Engineering Department

Master's 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 in Rviz of multiple grasps generated using Contact-GraspNet in relation to a captured depth image of the scene.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

Grasp Synthesis Methods on Known Objects for Bin-Picking
A comparison on deep-learning based and analytical-model based 6 DoF grasp pose synthesisers
Sevag Tafnakaji
Simon Widerberg
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology

Abstract

Automated pick-and-place operations are foundational tasks in robotics, with wide-ranging applications in industrial automation, logistics, and service robotics. Central to these operations is the ability of a robotic system to reliably plan and execute grasps on a diverse set of objects. Grasp synthesis, which is the process of determining suitable contact points and hand configurations for successful object manipulation, remains a challenging problem due to the inherent uncertainties in perception, object variability, and physical interactions.

To address the challenges of grasp synthesis, this thesis explores and evaluates two distinct approaches to grasp pose generation. The first approach leverages a database-driven method, storing precomputed grasp poses for known, proprietary objects. The second employs an end-to-end deep learning model capable of generalizing grasp predictions across a wide variety of novel objects. A complete robotic pipeline was developed to integrate these grasp synthesis methods into practical pick-and-place and bin-picking tasks. Using this pipeline, we conducted experimental evaluations on a physical robotic platform to compare the grasp success rates of both approaches in real-world scenarios.

We conclude that the deep learning method, using Contact-GraspNet for generating grasps, appears more fitting for the applications Volvo desire in their production environment due to its flexible and scalable nature as well as achieving an overall 50% success rate compared to 39% for the database-driven method for single object pick-and-place. While the database-driven method could still work, it is not as scalable and is reliant on an object pose estimation system.

Keywords: Robotics, Automation, Grasp Synthesis, Bin-Picking, Pick-And-Place, Deep Learning, ROS

Acknowledgements

Firstly, I would like to thank my examiners and supervisors. Your direction and assistance on this project saved us much time and effort that would have otherwise been wasted.

I would like to also thank my family, whose eternal patience and support helped me see this project through, despite the setbacks and delays.

To my friends, Hannah, Karthik, Monika, and Abhishek, I can only say thank you for the amount of times you have raised my spirits enough to finish the project. Now, I will have to find new things to complain about.

Sevag Tafnakaji, Gothenburg, June 2025

I also, would like to thank my examiner Karinne and my supervisors Atieh and Maximillian. Your expertise and knowledge was instrumental to the project and your detailed feedback helped us stay on track. Thank you!

Finally, I would like to thank my friends and family - thank you for being so supportive of every decision I have made during my time at Chalmers. It's been a long journey, but you've helped me realise how much I truly love the time spent as a student.

Simon Widerberg, Gothenburg, June 2025

This work was supported by the Vinnova project AIHURO (Intelligent human-robot collaboration).

Contents

List of Figures

List of Tables

1 Introduction

Many companies introduce automation onto their production lines to safely increase productivity and output. Automation comes in many forms such as using conveyor belts to transport products, using camera systems for quality assurance or using robotic arms to lift heavy objects. The use of robotic arms have become synonymous with automation thanks to their ability to repetitively complete precise, heavy and hazardous tasks such as welding, painting, or material handling.

With improved and cheapened technologies, the latest research of the past couple of decades have been focused on making these automation solutions more generalizable and intelligent [1]. For example, in robotic manipulation, the utilization of machine learning methods for object recognition and grasp planning can eliminate the need for statically defining the grasp location and trajectory. These advancements benefit companies by streamlining production lines, reducing human intervention, and increasing system adaptability to minor variations.

One application of these advancements is bin picking, a popular research topic in industrial automation [2]. Bin picking refers to the process of using a robotic arm equipped with a vision system to identify, pick up, and place objects from a bin or container. This is a crucial application in industrial automation, particularly in manufacturing, logistics, and warehousing. However, bin picking presents several challenges [3], such as accurately distinguishing individual objects in a cluttered environment and ensuring the system operates quickly and reliably in a production setting.

Volvo Group wishes to implement such a system in an attempt to solve an inefficiency in supplying their kitting stations with the necessary kit components. Volvo envisions a solution where a robotic manipulator is attached to the component rack, which then receives an order of components that is then picked and placed aside for transport to the desired kitting station. For example, consider a break caliper. The Caliper is composed of multiple components such as, break pads, bushings, and seals that need to be assembled together. A worker currently needs to find these components from the correct rack and bucket and bring them back to their kitting station. Instead, Volvo wishes to have a robot system attached to the racks which gathers the desired components from the rack onto a automatically guided vehicle, which then transports the parts to the stations. An example of how kitting stations are used today compared to their vision can be seen in figure 1.1.

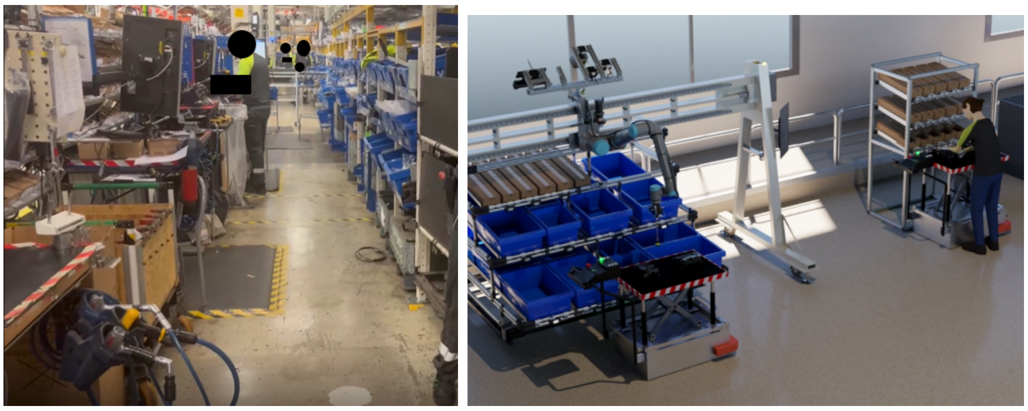


Figure 1.1: Current vs Volvo Group vision of the kitting process. rather than finding the correct object in one of the bins, it would be delivered to the worker through automated guided vehicles.

Volvo is currently researching one possible way to achieve their vision of having a robotic system detect and grasp objects for further transporting. Volvo identified that this can be achieved through grasp pose synthesis, which is the process of generating valid grasp poses to attempt to pick up an object [4] [5]. But their current implementation takes a long time to set up since the grasp pose needs to be manually configured for each object. This implementation also uses expensive proprietary hardware and software which would not be feasible to use on a large scale. These challenges are addressed in this thesis by relying on a cheaper camera and by developing a compatible method for synthesising these grasp points using only the visual information given from the camera and existing CAD (Computer Aided Design) files of the objects.

1.1 Objective

In order to determine a reliable method of synthesising grasps, two methods are implemented, tested and compared: One method where grasp poses are synthesised offline using analytical models and then queried during runtime, and another method utilizing a machine learning model to generate grasps from vision data.

The goal of the thesis is to answer the following research question:
Is a deep-learning model more viable than an analytical model for the purpose of grasp pose synthesis, given that the objects being picked are already known?

In order to answer this research question, we developed a pipeline which allows for easy and repeatable testing of grasp pose synthesis models, where we test their effectiveness in generating grasp poses that are good enough for a pick-and-place task. We measured this effectiveness through the use of two metrics: Grasp Success Rate (GSR), and Pick-and-Place Success Rate (PNPSR). GSR measures how often the model generates a grasp that is able to simply pick up the object. We measure it as a success if the robot is able to move the object from its current pose to an intermediary pose. PNPSR is similar to GSR, except the main difference is that it measures how often the robot is able to complete the full pick-and-place task using the grasp pose synthesis model. Both are measured by counting how many successful attempts were made out of a total number of attempts.

What differentiates our approach from the current research is that many of the cutting edge models assume no previous knowledge of the object when generating grasp poses. This means that the models do not require the full 3D CAD file of the object it attempts to grasp, and usually a depth camera and colour image of the scene is sufficient for it. We are more restrictive in our approach for two reasons:

- In manufacturing, very rarely is there a case that there are no CAD files for any of the objects to be picked from a bin.
- Access to the 3D CAD files allows the use of analytical models which allows for a stored database of possible grasp poses around an object.

1.2 Assumptions and Limitations

One of the main assumptions is that there will never be any "unknown" object, which are objects that 3D CAD files are not available for. This is what makes the comparison of methods feasible, as cases in which previously unknown objects can be encountered would make the analytical models useless, as they require the 3D CAD files to generate grasps.

Since our research question focuses on grasp pose synthesisers and will compare two different models, we must ensure that the data collected from our tests are not affected by other factors. Estimating the pose of the object is out of scope for this project. Therefore, we utilized known fixed object poses for testing. The reason for this choice is that we do not wish any vision based pose estimators to affect the performance of the grasp pose synthesising methods.

From the provided files and objects that Volvo can share, only 4 were available in both 3D CAD and physical form. Therefore, testing will only be performed on these 4 objects.

For training, and inferring grasp poses using the deep-learning model, we are limited to the hardware that is available. For both training and inference, a laptop with 16 GB RAM is connected to an external GPU, a Nvidia GTX 3090 with 24 GB VRAM. This limitation in RAM proved an issue for training and will be discussed later.

The thesis is limited to only testing cases where objects are positioned in a structured way on a table in front of the robotic arm as opposed the final configuration where objects will be placed in buckets on racks. The reason for this is that it would be very difficult, if not impossible to manually provide accurate poses for objects in a pile, so instead a structured setup was chosen to make repeatable tests possible.

Lastly, the thesis will only make use of a single gripper, which is a parallel fingered gripper from Robotiq [6]. This is a common enough type of gripper that is used in the industry, but there are other types, such as grippers with three or more fingers, human-like grippers, and vacuum/suction grippers. These other grippers will have grasp poses that are different than those of the Robotiq gripper, and therefore are not considered in this thesis.

1.3 Literature Review

1.3.1 End-to-End Deep Learning Models

Deep learning is a promising approach for grasp synthesis, driven by the rise of data-based solutions and advances in machine learning. [4] [5]. One simple version [7] would be to generate grasps from a top-down perspective which assumes the gripper will always be positioned vertically, such that only 2D position and the yaw orientation needs to be predicted. This provides compact output that is easier for models to infer, but lacks in generalized performance since the best point to grasp an object in different orientations is not always from the top. Therefore, in general cases, the full 6 degrees of freedom (DoF) is required to reliably find optimal grasps, but this comes at the cost of making the model and computations more complex.

Grasp synthesis in 6 DoF is a difficult problem to solve due to the amount of dimensions that need to be "searched" in order to find a viable solution. One possible solution to this is to add constraints on some dimensions. Sundermeyer et al. [8] uses what they call contact points, which are surface points from the pointcloud representing the object as a point where the parallel gripper must be in contact. This reduces the dimensionality of the problem by redefining the problem as a 3 DoF estimation around the estimated contact point. They showed improved results over the baseline model GraspNet [9] and showed that even unknown objects can still be grasped in a satisfactory manner. They also show promising convergence rates for training and integration of segmentation maps for filtering the input depth image. It is for these reason we decided to utilize Contact-GraspNet [8] for our implementation.

Other deep learning models are also developed other than the aforementioned end-to-end models. A combination of model-based and reinforcement learning (RL) solution was developed by Borja-Diaz et al. [10]. Rather than generating a grasp pose and then using a planner to move the robot to the grasp pose, they seek to remove that step and use their solution for both the grasp pose estimation, and navigation towards the goal pose. Regular deep learning models learn affordance regions (areas where the robot can interact with an object for a specific affordance/task) and estimate the objects' centre. The robot then uses an RL model to navigate the robot towards the goal pose. The use of RL allows for better navigation as it can more easily deal with errors in the prediction due to the stochastic nature of estimation based on visual information. This solution is able to deal with never before seen objects, but also tasks/affordances that it was previously unfamiliar, and able to guess where it would need to grasp in order to accomplish the task and was able to do it more effectively than the baseline it compared to. The reason that this is useful is that a good place to grasp an object is highly dependent on what it will be used for. For example, you can grasp scissors from the blades (as long as it is closed) in order to simply transport it, but you cannot do that if you plan to actually use the scissors. However, due to the nature of the implementation, there can only be one affordance region per object, and additional constraints on the region or grasp pose cannot be added without changing the critic in the reinforcement learning model or retraining the affordance region estimator model.

Other solutions were also developed, such as using transformers and/or attention modules to allow for more human-like understanding of knowing where to grasp. For example, a transformer-based model might infer that grasping one end of a long, thin object (like a pen) might be unstable and instead suggest a grip closer to the center [11] [12]. Additionally, some approaches leverage graph representations of the input point cloud, enabling the use of graph neural networks and attention mechanisms to capture spatial relationships more effectively [13] [14]. These networks, however, were shown to not work in real-time grasp synthesis applications due to the large size of the models, and therefore the high computational cost resulting in slow inference.

1.3.2 Datasets for Deep Learning Models

Developing deep learning models requires a vast amount of data for training and validation. However, collecting this data is a time-consuming process, as it often involves multiple millions of samples. Collecting this amount of data and labelling it is infeasible for this thesis, therefore existing datasets such as Graspnet-1Billion [9] and ACRONYM [15] can be used to train initial models. Both datasets are the basis for well performing models [9] [8] but they differ in their implementation where Graspnet-1Billion is based on real RGB-D images of 88 objects and over 1 billion annotated grasps while ACRONYM is a synthetic dataset comprised of 8872 objects and 17.7 million annotated grasps. Giving Graspnet-1Billion a much denser annotated dataset while ACRONYM boasts a much wider selection of objects.
The creators of graspnet-1billion also provide a baseline model to compare the results with, using common metrics in grasp synthesis research such as grasp success rate, that were shown to be popular in recent literature surveys [4] [5].

1.3.3 Analytical Grasp Pose Synthesisers

The previously mentioned approaches mostly focus on generating grasp positions online, meaning in real-time while the robot is actively operating. The alternative is offline methods that aim to create a large set of possible grasp positions that an online evaluation function can choose from. Such as Kleeberger et al. [16] which uses clustering algorithms to to pre-compute a diverse set of possible grasp positions of known objects. The grasp positions can also be generated using more intuitive heuristics such as difference in euclidean distance from the reference frame or based on constraints such as collision or infeasible joint angles [17].

Other alternatives include attempting to find grasps that will ensure the object would not slip from between the grippers fingers when moving. This is done through simulating the friction between contact points, and the forces and torques around these points. One of the most popular simulators for this purpose is GraspIt [18], where the friction, as well as wrenches (forces and torques) were modeled and tools are provided in the simulator to find suitable grasps for a multitude of different grippers.

The problem of finding grasps through suitable forces, however, is that the dimensions of the search are too high. This can lead to very long search times, and finding optimal solutions would be difficult. This search problem can be instead turned into an optimisation problem as shown in [19], where using neuroscience research, the degrees of freedom of modelled human grasps were reduced from about 20 down to 2 by using so called eigengrasps. Then by using Cost functions, these eigengrasps along with the grasp poses can be evaluated if they are good grasps or not, and solvers, like simulated annealing [20] can be used to find the local minima of these cost functions, and hence, the best grasps.

The issue with these simulators is that they require knowledge about the object's pose in order to then make use of these generated grasps, and since pose estimation is a non-trivial problem, how the user would solve that is a design choice that must be made. This is made trivial thanks to the deep learning models, as they do not require object pose information, but they are instead fixed to one perspective usually.

2
Theory

Robotics is inherently a multi-disciplinary field, combining mechanical, electrical, and software engineering. Therefore, some relevant information regarding vital parts of the thesis is provided in this chapter.

2.1 Robot Operating System

Establishing communication between different components of a robotic system is essential for ensuring coordinated and efficient operation. This is achieved using the Robot Operating System (ROS), specifically, ROS2 Humble [21]. ROS is a framework that allows easier and quicker implementations for robotics thanks to the open-source foundation of the framework. Reported by the developers of ROS, it is a popular tool in research and academics, thanks to its ease of use and quick implementations that allow the user to focus on the application rather than setting it up. This is due to the infrastructure that ROS provides, which is the communication through so called topics, services, and nodes (among many other features).

2.1.1 Nodes

A node in ROS can be considered as a block that is responsible for some behaviour. For example, a node can represent a sensor taking measurements of the environment, and doing any pre-/post-processing that is necessary, and then providing the result to other nodes through so called topics. These nodes can also set specific parameters that could be useful to know across different nodes as ROS parameters. Using a depth camera as an example, a node that represents it could set a maximum depth/distance value as a ROS parameter so that other nodes like the control system could take that information into account.

2.1.2 Topics and Communication

Communication between the nodes can be done in many ways, depending on the exact behaviour of communication desired. One of the most common ways is making use of topics. Topics are useful in continous stream type of communication, which are common for sensor readings such as sensors to represent the environment (cameras, LiDARs, etc) and sensors that represent robot states (encoders, IMUs, etc). Typically, topics have one node publish information to it through publishers, and one or more nodes listen to the published information through subscribers. If desired, multiple nodes can publish to the same topic, but if done simultaneously, it can lead to unexpected behaviour if not careful. The information that is published to the topics must be standardised through message types defined in message files. Within these publishers and subscribers, the user can define some script to process the message however they wish in order to make use of the information. The standard example to explain publishers and subscribers is by using a topic that takes string messages, with a publisher that sends "Hello" and an incrementing value, and a subscriber that reports what it received from the topic.

If the user instead wishes to trigger a process and receive a response, then one way to accomplish that is through services. Services are defined through servers and clients, where the server defines the process that the service is expected to do and generate the response, and the client is where the request is generated. Similar to topics, the services are defined through standardised service types. The services used in the thesis are defined in the appendix. The common example to explain services is a simple sum service, where the client sends a request with two values to be added, and the server returns the sum.

Figure 2.1 shows how communication through topics and services occur. The publisher will send its messages continously through the topic, and the topic will send duplicates of this message to any subscribers attached to it. Services are more one-to-one, where the client will get a response that is dependent on the request sent to the server.

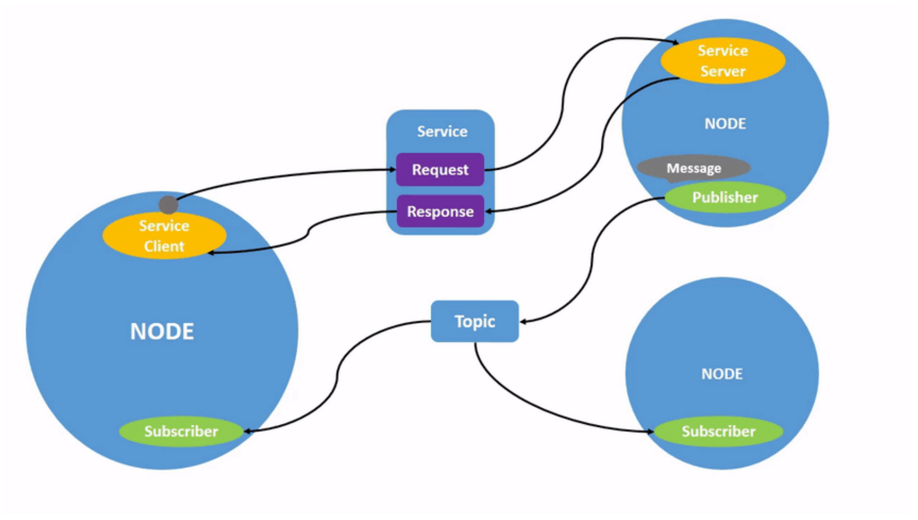


Figure 2.1: Image taken from the official ROS2 tutorials showing an example node setup with common ways of inter-node communication (described in section 2.1.2).

2.1.3 Coordinate Frames

In robotics, keeping track of where things are is essential for accurate planning and control. To accomplish this, coordinate frames and transformations are used. Positions must be defined to some relative fixed point, and in robotics, there are at least one fixed point that everything else will be relative to, commonly referred to as "world". However, it is sometimes simpler to change this frame of reference to another one that is more suitable, for example the lens of a camera or the base of a robot arm. Therefore, there are many frames of references called coordinate frames, and the change to another coordinate frame is called a transformation.

To aid with transformations, the ROS2 package TF2 [22] is used. The user will only need to define the coordinate frames relative to a parent, and the package will figure out the necessary transformations. The links and joints of robot can be abstracted as transformations, and using them, the end-effector of the robot can be tracked with any valid joint values. The user can define the robot through a xacro or URDF file, and ROS will interpret this file as a set of transformations. Once initialised, all of the latest transformations between the coordinate frames are stored in a buffer, and all interfacing with TF2 requires that the buffer is kept up to date. Any new transformations are added to the buffer, and therefore the current pose of any link can be extracted through the buffer itself, defined in terms of any of the coordinate frames, so long as there is a transformation between the two. Examples of some coordinate frames can be seen in figure 2.2.

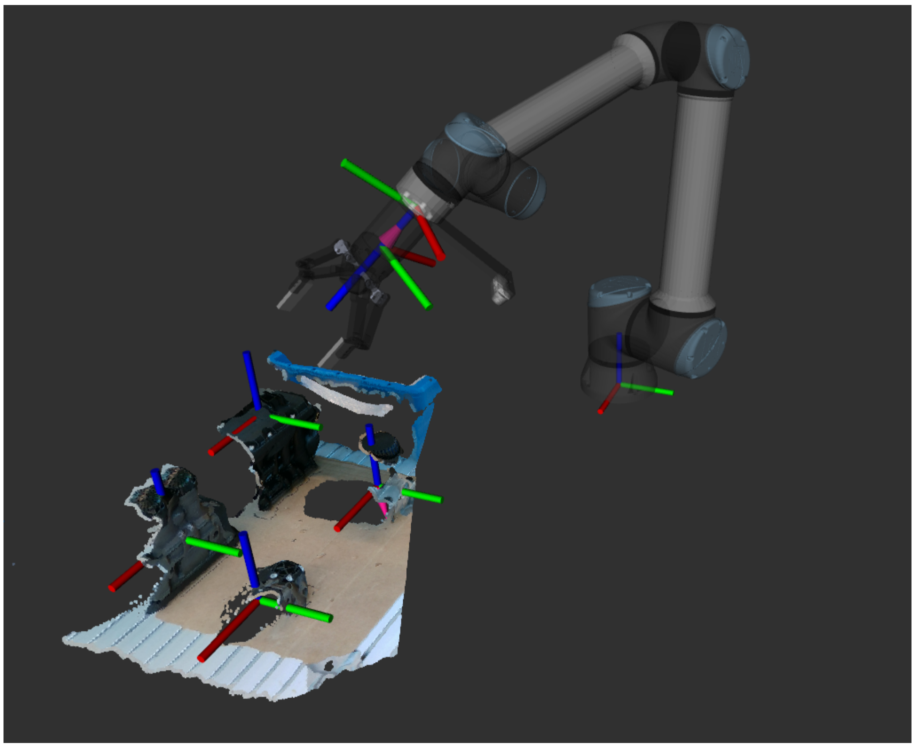


Figure 2.2: Example of Coordinate frames in TF2. The coordinate frames and its transformations on the robot were defined through the robot URDF file, whereas the coordinate frames of the objects were manually defined.

2.1.4 Controllers and MoveIt!

Moving a robot is commonly done through controllers, and the responsibility of the controllers is to take in the current joint values/angles, and the desired joint values, and decide what inputs to apply to each actuator to reach these desired joint values. Depending on the controller, the user can decide to control different aspects. Commonly, the user can control the joint position (joint values), but the user can also choose to control the forces and torques applied on each joint. The choice of controller, and its tuning is application dependent. ROS2 already provides many common control packages, and robot manufacturers also commonly provide packages to control their products/robots through ROS.

However, these controllers are basic and are not sufficient for robotics applications. In the case of robotic manipulation, the controllers packages do not consider collisions with the environment when reaching the desired joint values. This is left to a process that is called motion planning. There are many different algorithms for motion planning, and many collections of packages that already implement them. One of the most common packages for motion planning and its applications is called MoveIt! [23] as it combines implementations of popular stochastic motion planning algorithms. MoveIt allows for easy connection between controllers and motion planning, and allowing the robot to consider the environment when planning its path. Additionally, it also allows for planning a path in which the end-effector travels in as straight of a path as possible, by controlling the end-effectors path in cartesian space. This is beneficial as planning in joint space may require more movement than necessary to bring the end-effector from its current pose to its goal pose, as planning in joint space considers the joint values *at* the goal pose instead of the goal pose itself.

2.2 GraspIt

One of the grasp synthesis methods used for this thesis is a simulation environment called GraspIt! [18]. This simulator models friction between two objects using the Coulomb model. Simply put, the forces that can be applied at the contact point between two objects/materials is determined by the coefficient of friction between these two materials. The coefficient defines an angle of a cone (which is called the friction cone), and as long as the force is within this cone, then it is true that the magnitude of this force can be withstood and resisted by friction.

These friction cones are highly vital, as GraspIt attempt to find grasps that will have a property called force-closure, where the grasp will be able to resist any disturbance wrench (combination of force and torque), with the assumption that the contact forces are strong enough. This is done by determining if the wrench space origin is contained within the generated Grasp Wrench Space (GWS), which is "the space of wrenches that can be applied to an object by a grasp given limits on the contact normal forces" [18]. Colloquially, force-closure simply is a check of a "strong grip", and that the object will not slip from the grasp of the gripper.

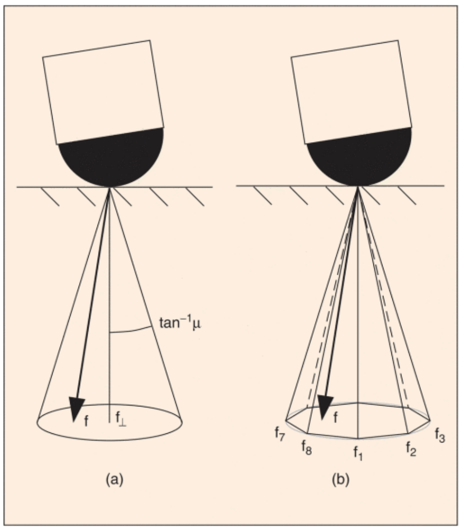
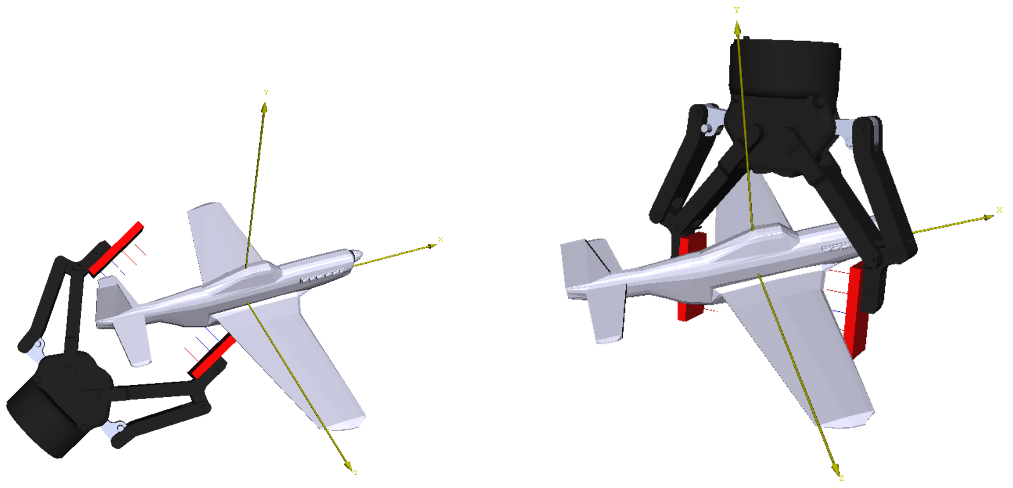


Figure 2.3: Cones used to if forces applied at contact is enough to counteract the friction. Taken from [18].

Force closure is not enough to determine if the grasp is good, however. To aid in this search, there are two steps taken to improve grasp quality and speed at which the good grasps are found: dimensionality reduction, and conversion from a search problem to an optimisation problem. Both of these steps are described in detail in [19]. The dimensionality of grasps is reduced from the number of degrees of freedom that a gripper may have into a set of so called "eigengrasps", which are common postures taken by the gripper. For example, with a human hand, one eigengrasp is described as "thumb rotation, thumb flexion, MCP flexion, and Index abduction" [19]. This can effectively be converted to a single dimension with a minimum and maximum "joint" value. Then one or two of these eigengrasps are enough to significantly reduce the dimension of the search.

By using so-called "Energy Functions", that take in the eigengrasp amplitudes (what values between the min and max it is for each eigengrasp), and the pose of the grasp and provide a single value to represent how good it is, the problem of searching for a good grasp pose gets converted into an optimisation problem. The specific energy function that is recommended in GraspIt is called Guided Potential Quality Energy, where it first checks if the grasp has force-closure, and if it does, evaluates how close the grasp contact points between gripper and object is to the user-defined desired contact points. The planner/simulator uses simulated annealing for the optimisation, and since that method minimises the search function, the energy value is made negative. This means that any non-zero grasp has force closure, and the lower the value is, the closer each contact points gets to the user defined points. With these user defined points, the user can determine if only the tip of the grippers should be used, or if the object should be covered by as much of the gripper as possible.



(a) Grasp with highest energy score. Thanks to force-closure criteria, it is highly likely that the toy airplane will not slip from its grasp

(b) Grasp with the second highest energy score. Thanks to force-closure criteria, it is highly likely that the toy airplane will not slip from its grasp

Figure 2.4: Robotiq Adaptive 2F-140 gripper grasping a toy airplane. Result gathered after 100,000 Iterations using the energy function guided potential quality energy.

2.3 Contact-GraspNet

Contact-GraspNet [8] is a grasp synthesis model published in ICRA in 2021 by Nvidia and TUM researchers Sundermeyer et.al. They proposed an end-to-end network that efficiently generates a distribution of 6-DoF parallel-jaw grasps directly from a depth recording of a scene. This depth recording can be a RGB-D image coupled with the camera intrinsics and optionally a segmentation map of the objects in the scene, or, it can be a point cloud composed of 3 dimensional points.

Including a segmentation map will allow the model to filter out grasps that were generated outside of the intended object. The segmentation map also allows the use of local regions of interest for preprocessing the depth scene by cropping the scene to only focus on a the segmented area. This maximizes the amount of contact points the model can find as well as minimizing the inference time.

Figure 2.5 shows the architecture as the paper defines it. First, the dense point cloud is converted to a feature list, which the model can learn from. To do this, a highly popular architecture called PointNet++ is used [24], Pointnet++ can be interpreted as a more effective alternative to volumetric CNNs (Convolutional Neural Networks) that similarly captures features in 3D space but without rigidly scanning the space with a set stride. These features are then fed to 4 different heads are used for the 4 different estimation tasks.

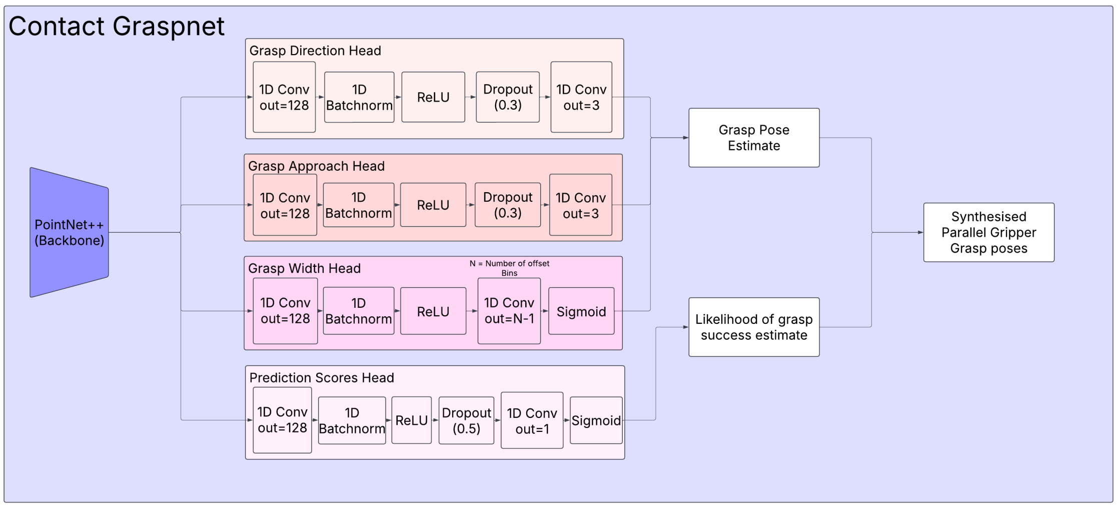


Figure 2.5: Figure showing the architecture of Contact-GraspNet (from left to right). The input would be the RGB-D image and segmentation map from the camera, and the output would be the synthesised grasp poses with estimated probabilities of success

The heads for the approach and baseline (which we will refer to as grasp direction) vectors have very similar structures, as they are both estimating vectors. The outputs of these heads are vital in determining the orientation of the grasp pose. The grasp width head is a simpler variant, but estimated through equidistant bins ranging from zero to the maximal width of the gripper. Each bin contains a range of values, and the final grasp width would be the centre value of the grasp bin with the highest confidence.

The final head (Prediction Scores Head) is the basis of the grasp pose representation, as this head is what is used to predict the contact point, c, which is the point from the point cloud that the gripper will be in contact with and therefore apply its approach and grasp direction vector in relation to. This head converts the input point cloud into likelihoods of grasp success, such that the point with the highest likelihood of success is subsequently chosen as c. The combination of all four heads can then represent a parallel grasp, as seen in figure 2.6.

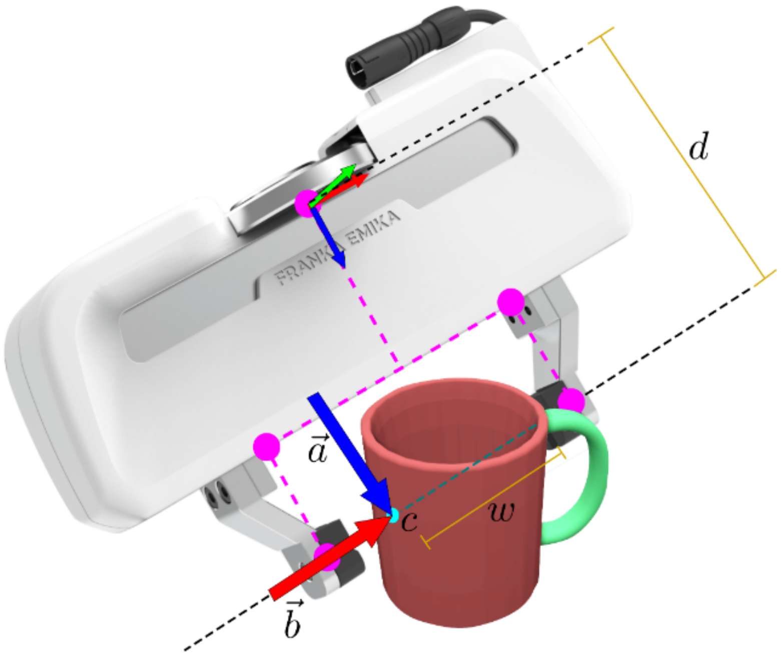


Figure 2.6: Image taken from the original Contact-GraspNet paper [8]. Figure showing the all estimated variable from each head in relation to each other. $\vec{a}$ and $\vec{b}$ are the approach and grasp direction vector. w is the grasp width and c is the contact point.

The network is trained on the ACRONYM dataset [15] which consists of 8872 meshes from the Shapenet dataset [25] and 17.7 million simulated grasps under varying friction. These meshes are then placed in random stable poses in scenes. Point clouds of the scenes are then rendered and used as inputs that can be compared to the ACRONYM ground truths.

3
Methodology & Design

In this chapter, we will present how and why we implemented the two different methods for synthesising grasps. We also go over the implementation for using the robotic manipulator with ROS2 and how we designed the experiments to compare the two synthesis methods.

3.1 Using a Deep Learning Model to Generate Grasps

We decided to use Contact-GraspNet for our deep learning based solution because it stands out among grasp synthesis models for several practical and technical reasons, particularity because they claim stellar performance and sufficiently fast runtime for complex scenes. The model also accepts segmentation maps as an input, allowing the model to operate in cluttered environments.

The Anaconda environment provided on the official GitHub [1] is not compatible with 3000-series Nvidia cards therefore different versions of libraries needed to be used. The model also utilize Tensorflow 2.5 which has limited compatibility with newer libraries. The environment used in our implementation is based on a forked repository [2] that also implements Contact-GraspNet for Docker and a 3000-series Nvidia card.

The original configuration for training the model is built around the Franka Emika Panda gripper which has a maximum grasp width of 80 mm. For our implementation the model needs to facilitate 140 mm grasp width. The model makers recommends to either retrain the model with the necessary configuration or to scale the point cloud dimensions and the resulting grasp position by the fraction $\frac{W_d}{W_p}$ where W_p is the Panda gripper width and W_d is the desired gripper width. Scaling the input and output works very well for many objects and has the benefit of using their model weights that are advertised to have great performance. When using the model with RGBD images captured using the realsense camera, there were issues generating grasps for objects with heights lower than approximately 5 cm. Scaling down the point cloud also has the negative effect of making the scene tighter and harder to produce collision free grasps. Therefore, we also attempted to retrain the model to

[1]https://github.com/NVlabs/contact_graspnet
[2]https://github.com/tlpss/contact_graspnet

validate if a reconfigured model could remedy these problems.

3.1.1 Retraining the Model

Contact-GraspNet is trained on the ACRONYM dataset which is built using mesh data from ShapeNetSem [26], a smaller and more densely annotated subset of the large 3D CAD dataset ShapeNet. Compared to ShapeNet, ShapeNetSem enriches the dataset with useful semantics such as weights, material composition and physical sizes of common household items. These meshes then need to be processed further to make them watertight, meaning they do not have holes, gaps or overlaps in its surface. And finally the meshes complexity needs to be simplified in order to load as many objects as possible into memory at once. These meshes are then combined with the ACRONYM grasp annotations and Contact-GraspNet's provided scene configurations and contact point annotations.

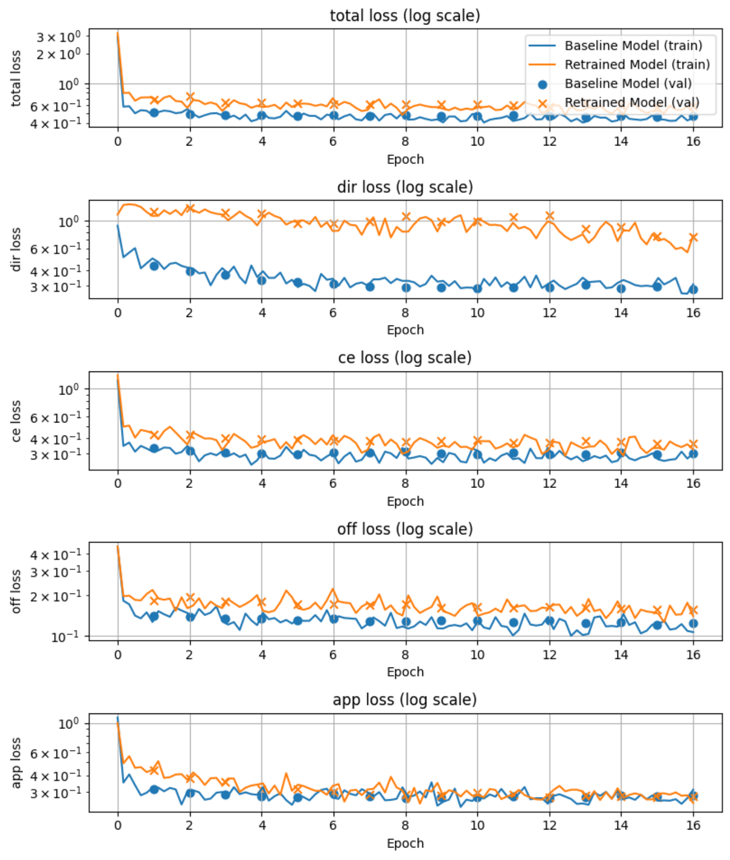


Figure 3.1: Plots showing the loss and validation loss for both the Baseline models training vs the retrained model. "dir loss" represent the grasp direction estimation loss, "ce loss" short for cross entropy loss representing the contact point classification loss, "off loss" representing grasp width estimation loss, and "app loss" representing the approach vector estimation loss. Baseline (blue) include 14 000 iterations over 16 epochs while retrained (orange) include only 1300 iterations over 16 epochs.

Contact-GraspNet recommends to use a system with a Cuda capable GPU with more than 24 GB VRAM and 64 GB of normal RAM to be able to load the entire dataset into memory at once. The host computer used for this tasks was limited to 16 GB of RAM and can therefore only load a small subset of the dataset. Less training data leads to decreased model performance, especially in terms of generalisation as overfitting to certain objects becomes more likely. In our case, the model did not run enough iterations to start improve the grasp direction estimation, as can be seen in figure 3.1 where "dir loss" only improved slightly over all 16 epochs compared to the training logs from the baseline model. The grasp direction vector is the vector from the finger of the gripper to the contact point, which is orthonormal to the approach vector.

3.1.2 Segmentation Map

One of the inputs for the grasp synthesis model is a segmentation map of the object(s) in the scene that grasps should be generated for. Meta's Segment Anything Model (SAM) [27] combined with manual region-of-interest (ROI) selection was used to generate a segmentation map of the object of interest. The ROI selection is made with an interactive `Opencv` [28] window.

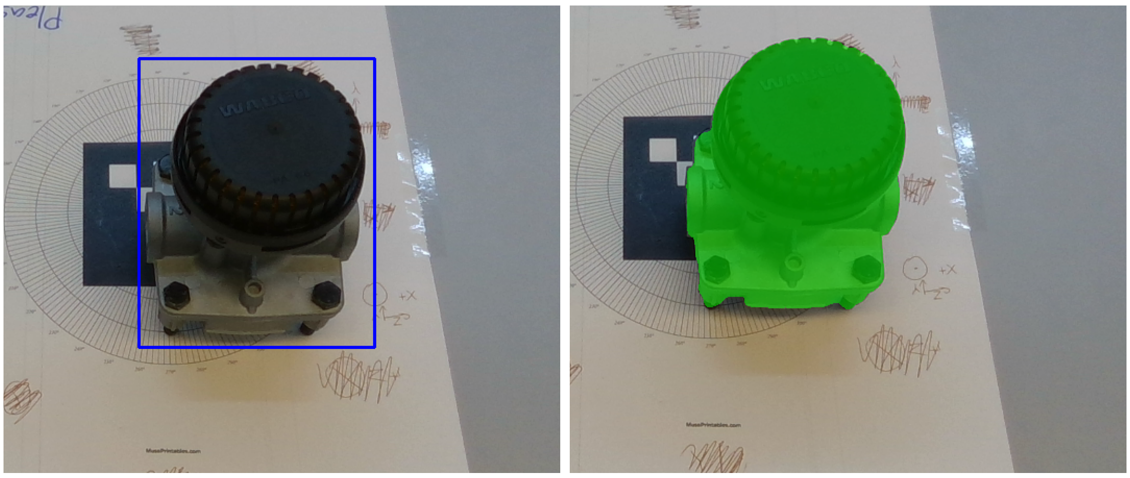


Figure 3.2: ROI selection applied to image (1) and resulting SAM segmentation map overlayed on top of original image (2).

3.1.3 Inference for Generating Grasp Poses

During runtime, the inference is performed in a ROS2 node, figure 3.3 shows the pipeline of the node. The inference script allows for several runtime arguments to increase the probability of finding a grasp as well as to filter what grasps are returned. These are the runtime arguments used:

- `forward_passes`: Determines how many batched forward passses the model will perform, increases potential grasps and increases inference time.
- `z_range`: Limit the search for grasp to within the thresholds $[z_{min}, z_{max}]$.
- `local_regions`: A boolean flag whether to reduce the search area to around the provided segmentation map.
- `filter_grasps`: Only return the grasps in the scene that contain a contact point in the segmentation map.
- `skip_border_objects`: Will ignore grasps generated on the boundaries of the final depth image.

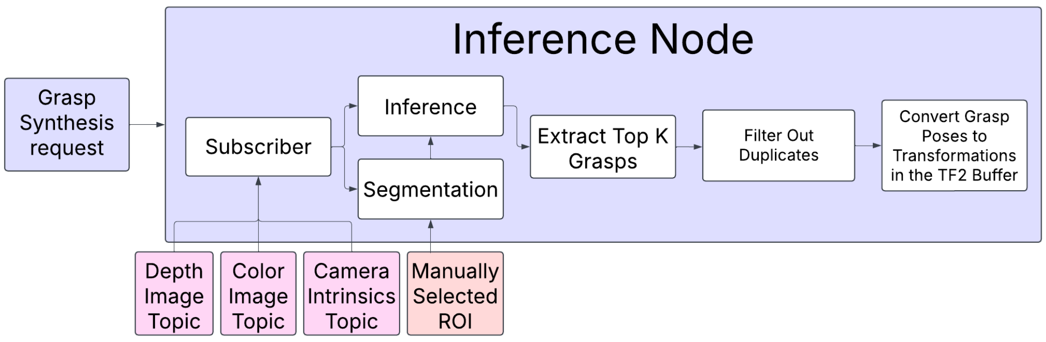


Figure 3.3: The inference node receives camera data from dedicated topics. The depth and color image as well as the camera intrinsics are passed into the inference script along with a segmentation map generated from the color image and a manually selectd ROI. The inference script outputs up to 200 grasps, many of which can be duplicates, which are filtered out. Finally the poses are converted to ROS interpretable TF2 frames.

Resulting grasp poses will be a homogeneous matrix describing the rotation and translation of the grasp. This grasp need the then be converted into quaternions and finally placed in a transformation frame with the camera color lens as the reference frame.

3.2 Using an Analytical Model to Generate Grasps

For the analytical model, we chose to use GraspIt. The reason for this choice is due to the fact that the simulator in GraspIt models the forces applied at contact and ensures that it would not slip from its grasps, and would allow for custom evaluation functions if need be, so it is highly adaptable to Volvo's use case.

3.2.1 Environment Setup

In order to use the GraspIt in ROS2, the user must first generate a database of possible grasps. This is done through packages provided by this repository [29]. These packages allow for direct interfacing with GraspIt, without having to use the graphical window. It requires that a setup is defined in so called "worlds", and an example setup is shown in 3.4. The table in this environment is loaded in as an obstacle, so that GraspIt knows to not generate grasps that would lead to collisions with the table.

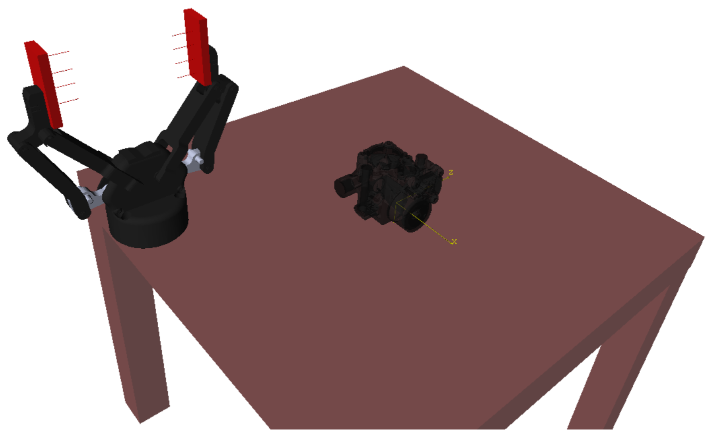


Figure 3.4: An example of the worlds used to generate grasps for one of the tested objects. Note the red lines on the gripper fingers, which are the contact points and its normals. see 2.2 for definitions of contact points.

As defined in section 2.2, the user must add specific contact points along the gripper. Then, the ROS1 interface to the GraspIt planner generates a list of the best grasp poses it can find, and stores them all in separate files, defined in a similar fashion to the input world file. A separate script, outside of ROS1, then combines all of them into a single database that can be read when needed. This database can then be used in ROS2 inside of the grasp synthesis node.

3.2.2 Using the Grasps in ROS2

Inside of the grasp synthesis node, when a request for grasp poses is received, the name of the object is expected. The reason for this is to extract the coordinate frame from the TF2 buffer. It is assumed that the name of the coordinate frame defined in the buffer is the same as the object name itself. This means that how the pose of the object is given/estimated is not relevant for this node.

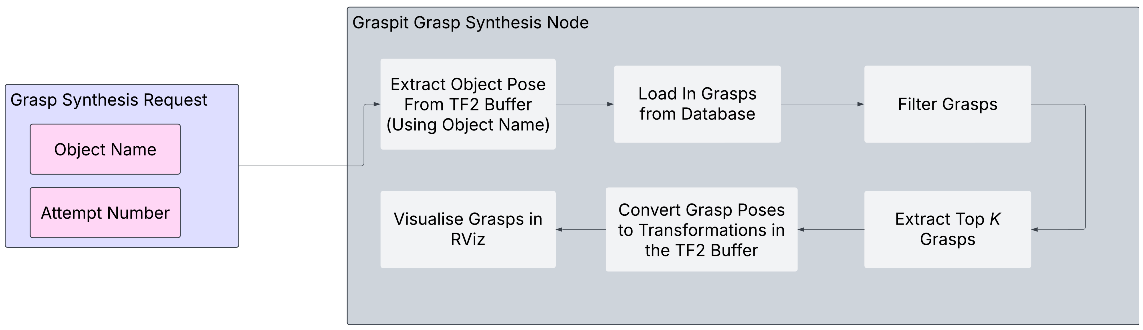

Grasp Synthesis Request
Object Name
Attempt Number
Graspit Grasp Synthesis Node
Extract Object Pose From TF2 Buffer (Using Object Name)
Load In Grasps from Database
Filter Grasps
Extract Top K Grasps
Convert Grasp Poses to Transformations in the TF2 Buffer
Visualise Grasps in RViz

Figure 3.5: Figure describing a single pass of grasp synthesis using the grasp database generated by GraspIt. Filtering is dependent on specific the object that is going to be grasped, and how many attempts were made so far to generate a grasp

For this thesis, the pose of each graspable object was manually fed into the buffer through static transformations. After the pose is loaded from the buffer, the grasp pose database is loaded, and then the grasp poses are filtered out. The filtering step is the most vital, as it would re-order the grasp database to fit the current environment best. Through iteration, the following filtering conditions were determined:

1. If the angle between the grasp position vector and the horizontal plane, represented by Θ in figure 3.6, is less than a minimum given angle, θ_1, remove the grasp pose from the database.
2. if Θ is between θ_1 and a second provided angle, θ_2, remove the grasp pose from the database if it has grasp rotation value, represented by Ψ in figure 3.6, less than a given value, ψ_{max}.
3. If the angle between the position vector projected to the horizontal xy-plane and the x-axis, represented by the angle Φ in figure 3.6, is less than a given ϕ_{max}, remove the grasp pose from the database.
4. If the distance from the robot end effector to the object pose is smaller than the distance from the robot end effector to the grasp pose, remove that grasp pose from the database
5. In case of Object 2, if we reached attempt number 3, add a slight offset of 5cm to the grasp position in the z-axis. This is done to offset the top part of the object, which is not present in the CAD file. Since it is not present in the CAD file, it cannot be taken into consideration by GraspIt.

If none of the conditions above are met for a grasp pose, then it means that the specific grasp pose stays in the database.

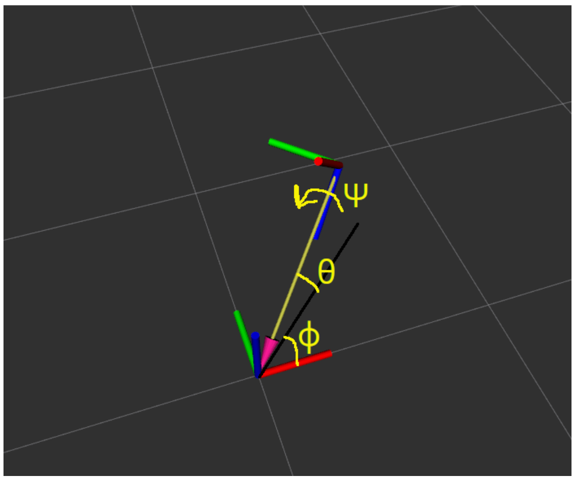


Figure 3.6: Filter angle representation. Figure shows an example object coordinate frame with an example grasp pose coordinate frame above it. Θ shows the angle between the position vector of the grasp pose and its projection to the horizontal plane, Ψ shows the yaw euler angle of the grasp pose (rotation around the z-axis), and Φ is the angle between the projected position vector of the grasp pose and the x-axis.

The exact values for θ_1, θ_2, $\psi_{\max}$ and $\phi_{\max}$ is dependent on both the object itself, and the attempt number and is shown in table 3.1.

Attempt Nr.	Object 1	Object 2	Object 3	Object 4
1	(0, 30, 60, 90)	(0, 30, 80, 90)	(0, 30, 60, 10)	(0, 30, 60, 10)
2	(20, 60, 75, 90)	(20, 60, 85, 90)	(0, 60, 75, 20)	(0, 60, 75, 20)
3	(40, 90, 90, 90)	(40, 90, 90, 90)	(0, 90, 90, 25)	(0, 90, 90, 25)

Table 3.1: Filter angles (θ_1, θ_2, $\psi_{\max}$, $\phi_{\max}$) (in degrees) based on object and attempt number. These angles are part of what determine which synthesised grasp pose is the "best" and most likely to succeed.

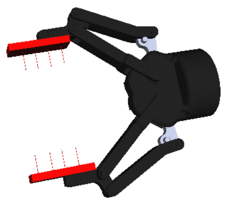
The values of these angles were determined through trial and error, and could still be further improved through more testing or better filtering methods.

After the filtering step is done, the top five (or however many were left after filtering) grasps were extracted, then these grasps are visualised in Rviz and their poses is added to the TF2 buffer of transformations. Once the TF2 buffer is updated, a response sent back to the client containing information about the grasps generated (pose, index from database, etc.).

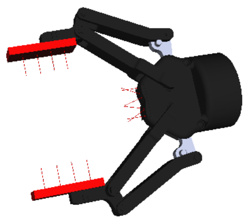
3.2.3 Additional Contact Points

The definition of contact points on the gripper is very vital, as it will affect the output of the energy function used during GraspIt's grasp synthesis. It is suggested that the contact points should be along the fingers of the gripper, but to encourage grasps with the gripper closer to the object, contact points can be added along other parts of the gripper.

To test if grasps with more contact points were better, we created two different sets of contact points, one with the standard suggestion of contact points only along the finger (figure 3.7a), and another with more contact points (figure 3.7b). The idea is that heavier objects, such as objects 3 and 4 (figures 3.15c and 3.15d), would perform better when the grasps are encouraged to be closer to the actual object itself, since the forces applied would be closer to the center of gravity.



(a) Contact points just on the finger pads, encouraging that the point of contact between the object and gripper occur mostly on the finger pads. This configuration will be known as "G, few CP" (GraspIt, few Contact Points).



(b) Additional contact points added to force the gripper to be closer to the object. This configuration will be known as "G, more CP" (GraspIt, more Contact Points).

Figure 3.7: Two different contact points defined, which lead to different grasp poses stored in the database

For an explanation as to how the defined contact points can affect the grasp pose synthesis in GraspIt, see section 2.2.

3.3 Implementation of the Pick & Place Pipeline

In this section, we will cover the main nodes that facilitate the pick and place task and allow the user to keep track of the performance of the grasp synthesis models as well as the overall system performance. Figure 3.8 shows how these nodes communicate between each other, where services are marked in purple, and topics are marked in red.

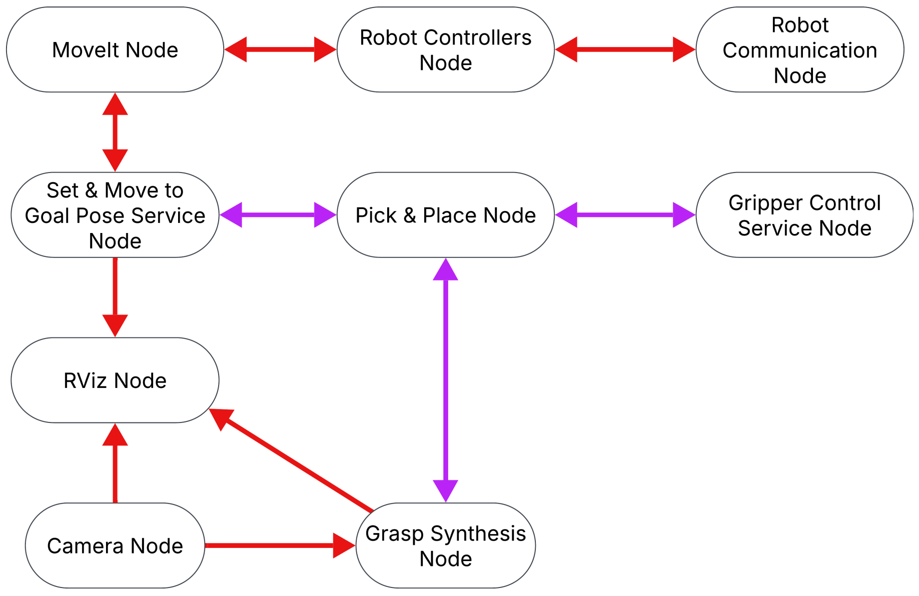


Figure 3.8: System architecture shown through the ROS2 nodes. The red arrows are connections through topics (i.e. publishers and subscribers), whereas the purple arrows are connections through services. Here the Pick & Place node contains the client of each service called.

Since most communication between nodes uses topics and services, the internal processing doesn't matter as long as they send and receive data in the same format. This makes the implementation highly modular, and allows the user to change any of the nodes shown in the figure and use any other implementation. A new user can choose to control the robot with a different controller, have a different planner instead of MoveIt, or even implement new grasp pose synthesising methods. This is one of the main benefits of using ROS2, and it is what allows for the easy comparison of the two different grasp pose synthesis methods.

3.4 Controllers and Motion Planner

The controllers chosen for this thesis is a scaled joint trajectory controller, which is an extension of the popular joint trajectory controller. The joint trajectory controller finds a viable trajectory that the robot can take such that, at the end, the robot will end up at the given joint values, and will attempt to fulfil that trajectory within a given execution time. The extension is based on the fact that the UR arms can have set velocity scales such that the maximum velocity can be some reduced value. If the scale is set to some value that is less than one, then the controller is not aware that it cannot use the full potential of the robot, and will hence fall behind in the trajectory. The scaled joint trajectory controller however mitigates this problem and will not lead to any of these issues. This controller is provided by Universal Robots [30]. These controllers also work in tandem with the real robot, so any control outputs are sent to the real robot for execution.

The MoveIt node starts all the necessary configurations for the MoveIt package, which is necessary for our custom interface to MoveIt planners and collision checks to work.

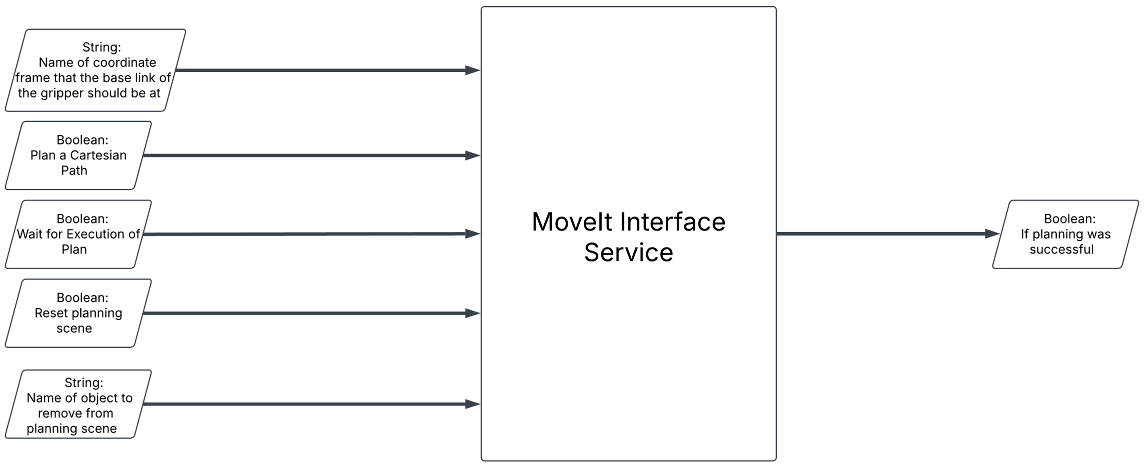

String:
Name of coordinate frame that the base link of the gripper should be at
Boolean:
Plan a Cartesian Path
Boolean:
Wait for Execution of Plan
Boolean:
Reset planning scene
String:
Name of object to remove from planning scene
MoveIt Interface Service
Boolean:
If planning was successful

Figure 3.9: The service that uses the interface to the MoveIt interface. Shows the expected inputs and outputs of the service.

The MoveIt interface node contains the custom services defined for this thesis, which allows the user to request that the gripper base link coordinate frame would lie on top of another coordinate frame. The coordinate frame passed into the service server through the request should be already defined in the TF2 transformation buffer, such that it can be used to navigate there. The idea is that the grasp poses generated by the grasp synthesis nodes will be added to the TF2 transformation buffer, and the grasp task node will use this interface to check if it is feasible to navigate there, and to plan a path there if it is. Figure 3.9 shows the expected inputs and outputs of this service. In order to let the planning algorithm find a direct, straight line between the end-effectors current pose, and the desired goal pose, we added the flag for cartesian path planning, which uses the MoveIt interface to find a viable plan in cartesian space rather than the joint space of the robot.

The last two inputs deal with the planning scene, which is the representation of the environment (scene) that MoveIt uses for planning. It always includes the robot itself, but can also include other objects as obstacles, in our case, we add a rectangular box to represent a table.

3.5 Gripper Control Node

The gripper control node is an additional custom service that allows the user to easily set the gripper to open and close. Thanks to the aforementioned force sensors on the gripper, the user does not need to consider the grasp width that is estimated through the grasp pose synthesis models, and instead simply command the gripper to close as much as it can. The firmware on the gripper stops the gripper before it damages the object.

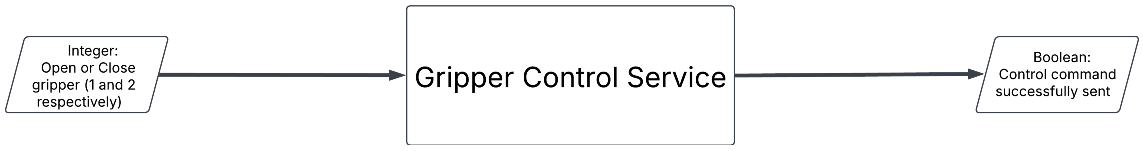


Figure 3.10: The service that controllers if the gripper is open or closed.

3.6 Grasp Synthesis Node

The grasp synthesis node is the core of the thesis, where the two different methods of grasp pose generation are interchangeable here. In its current form, the two different methods of grasp synthesis use the same service name, and are hence never run simultaneously, but it can very easily be adapted so that each solution is a different service to call. Figure 3.11 shows the expected inputs and outputs of these services.

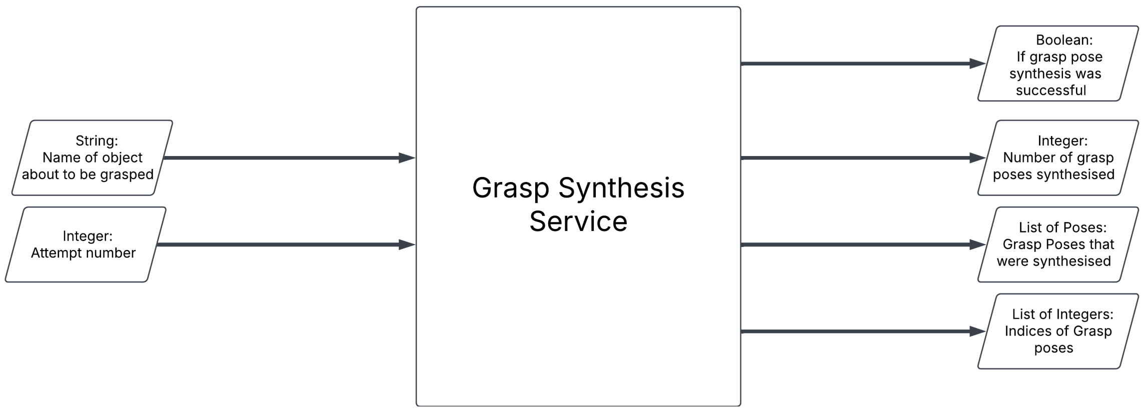


Figure 3.11: The service that generates grasps when called.

For our implementation, only the boolean value which shows if grasps were synthesised is used for logic, and the rest were useful for debugging. The user can directly make use of the provided grasp poses by looping through the response, or they can also use the TF2 buffer as the grasp poses are also added there.

3.7 Pick And Place Node

To test the grasp synthesis model, a simple pick-and-place task will be used, in order to determine if the grasp pose generated is robust enough to be able to transport an object. The grasp task node is what coordinates this. This is the central node that is expected to be the main interaction point between the system and the user. The pick and place task consists of the following steps:

1. Plan and navigate to the start pose.
2. Synthesise grasp poses
3. Use the MoveIt interface to determine if any valid grasp pose was given (i.e. a plan for navigation was found and there are no collisions). If a valid plan is found, execute it and wait until the robot is at the goal pose.
4. If none of the provided grasp poses were valid, repeat steps 2 & 3 for a maximum of three attempts.
5. If no valid grasp pose was synthesised after a maximum of three attempts, the pick and place task has failed. Otherwise, continue.
6. At this stage, the robot should be at the goal pose. Close the gripper.
7. Move to the start pose
8. Move to the drop off pose (passing through an intermediary pose that is some vertical offset away from the drop-off pose).
9. Open the gripper, dropping off the object in the bin.
10. move to the start position

If more than one string is provided in the list of string input field shown in figure 3.12, repeat steps 2-10 for each object name/string. The planning and navigation, grasp pose synthesis, and gripper control is done through the aforementioned services.

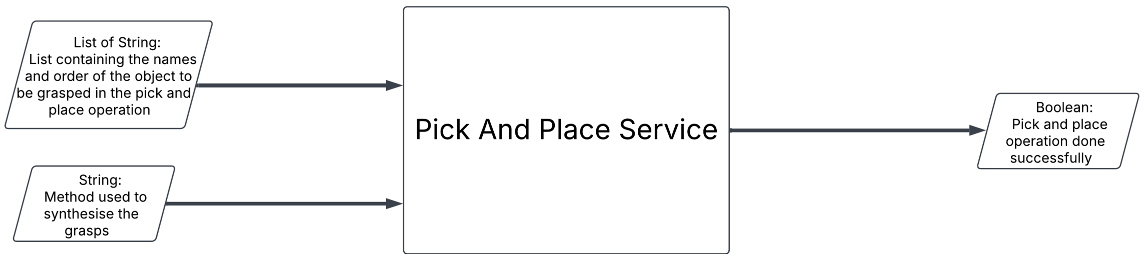

List of String:
List containing the names and order of the object to be grasped in the pick and place operation
String:
Method used to synthesise the grasps
Pick And Place Service
Boolean:
Pick and place operation done successfully

Figure 3.12: The service that performs the pick and place task by iterating through sub-tasks using the other services.

3.8 Visualisation Node

To visualise how well each system is working and the planned path for movements, Rviz will be used. Almost every node will make data available for visualisation through topics, so that the user can more easily track how each sub-system is behaving. The motion planner will visualise the final pose it will plan to, as well as the planned trajectory it expects to execute (if it is valid. In case it is not, it simply visualises the trajectory). The grasp synthesis node will also visualise the grasps through placing a simplified model of a parallel fingered gripper at the grasp poses, along with its index, see figure 3.13.

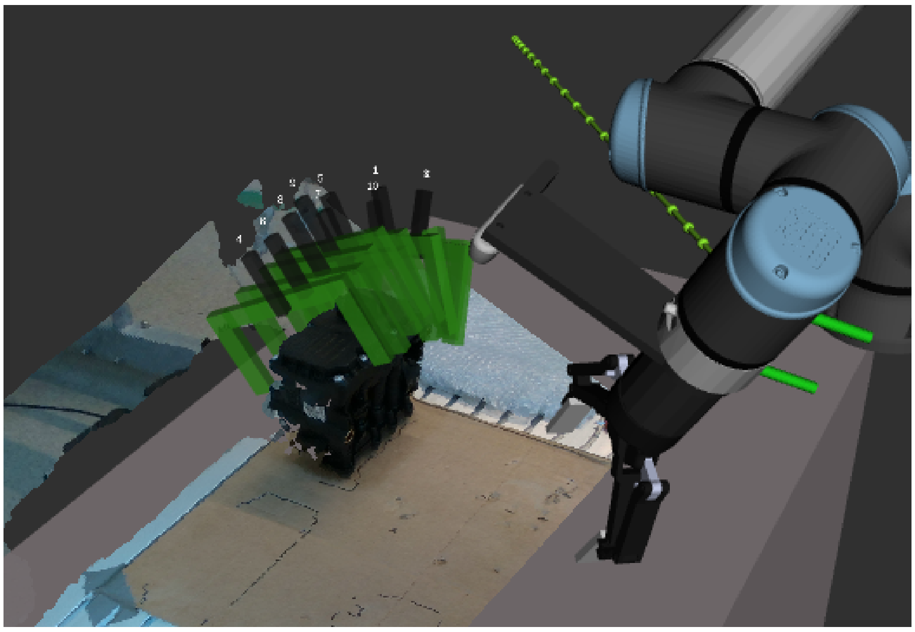


Figure 3.13: 10 Generated grasps on an object displayed in Rviz as markerArrays along with their index.

3.9 Camera Node

To capture the environment the realsense ROS2 wrapper is used to spawn a camera node. This node Publishes the color image, depth image, an aligned depth-to-color image and the camera intrinsics to its own topics. The aligned depth-to-color image contains a transformed depth image that has the same pixel mapping as the color image. This allows any classification or segmentation that is computed on the color image to be mapped directly onto the depth image. This alignment is also necessary for the grasp synthesis model. The ROS wrapper provides Several post-processing filters for the depth imaging that are used to reduce noise as well as decrease processing time. The filters applied are:

- **Disparity filter**: Performs transformation between depth and disparity domains which enhances following filters.
- **spatial filter**: Applies edge-preserving smoothing of depth data.
- **temporal filter**: Filters depth data by looking into previous frames

There are more filters available such as decimation filter and hole-filling filter these were not used. This is because decimation filters downsamples the depth image which loses information and because the processing time is fast enough without it. While the hole-filling filter created a more dense depth image it also created unwanted artifacts around the edges of objects that were not spatially distinct enough from its background.

3.10 Experiment Setup

The system consists of a UR10e robot [31] with a Robotiq Adaptive 2F-140 gripper [32] attached to it. Also attached to the robot is a Realsense D435i depth camera.

In order to have a full view of the workspace, we chose to attach the Realsense camera using a custom designed holder. This holder fits on the end effector of the UR10e, and adds a vertical offset such that the camera is not occluded by the gripper in front of it. The part was designed to be just long enough so that the finger of the gripper is out of view. The camera can then be screwed on the holder as shown in figure 3.14

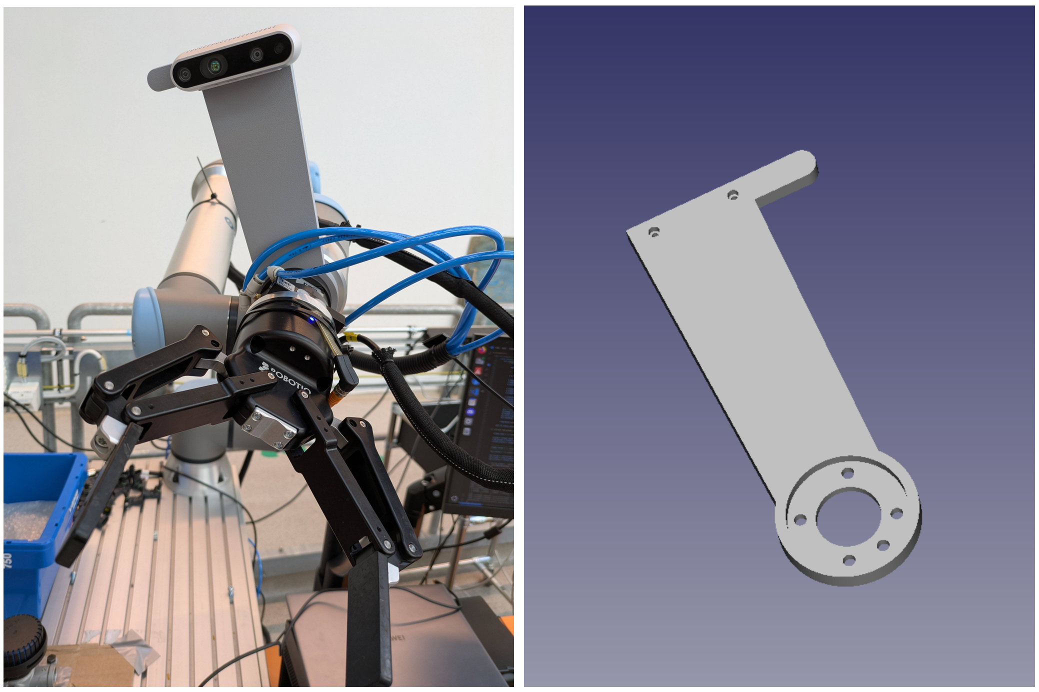
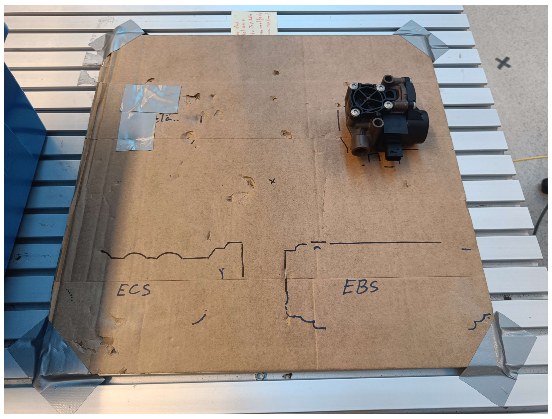


Figure 3.14: Figure showing the robot used to test the implemented pick and place pipeline and grasp pose synthesis models.

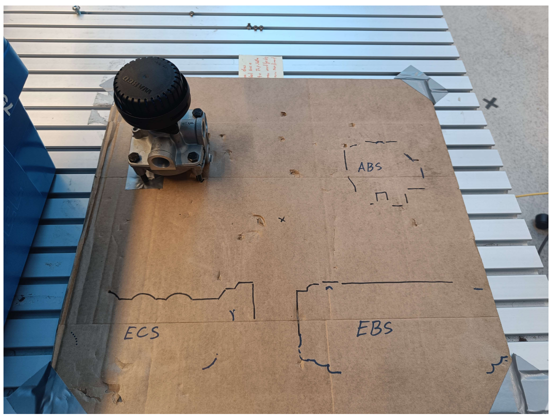
A laptop hosting a docker container of an Ubuntu 22.04 environment with ROS2 Humble is connected to the robot with an Ethernet cable. Communication with the gripper is done through the same Ethernet cable.

To answer the main research question of the thesis, and to facilitate a fair comparison of the grasp synthesis models, the aim was to define a highly repeatable test setup.

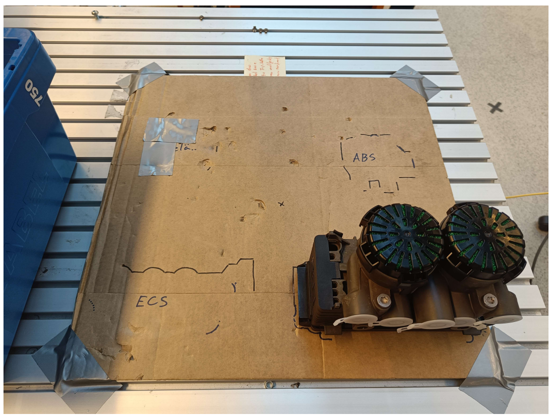
Available to test with were 4 objects, shown in figure 3.15. These are objects that require "kitting" on the Volvo Group production line, and hence primary candidates for automated retrieval.



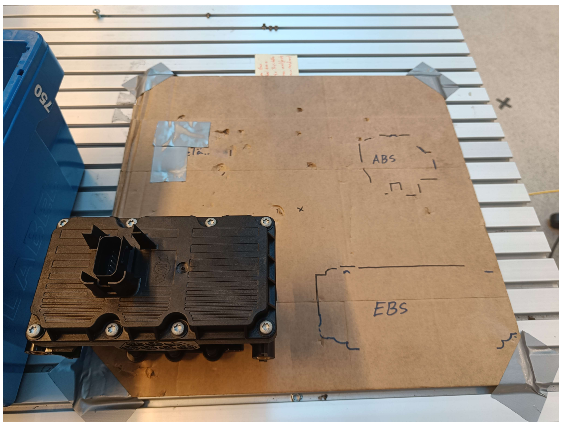
(a) Object 1



(b) Object 2



(c) Object 3



(d) Object 4

Figure 3.15: The four different objects that grasps can be generated for using *both* of the methods. While Contact-GraspNet can work on many different objects effectively, GraspIt required more information about the objects provided through CAD files.

Due to the geometry and weight of objects 3 and 4, it was expected that both methods will have the most difficulty with these objects, as slight deviations from an optimal grasp pose would result in a failed grasp or the object slipping from the grasp during movement.

It is worth noting that there is some discrepency between the CAD file of object 2 and the object that we received to test with. Inside of the CAD file, there is no top black section, and since we did not have access to more objects, we had no choice than to continue with the discrepancy. This would only affect the results of the GraspIt solution of object 2.

In order to test the efficacy of the methods, two categories of tests will be setup: single objects, and grouped objects. Each object has a fixed defined pose in the environment, as can be seen in figure 3.16, where during the first testing phase, only one object is present in its pose, whereas in the second phase, all of them are present at the same time.

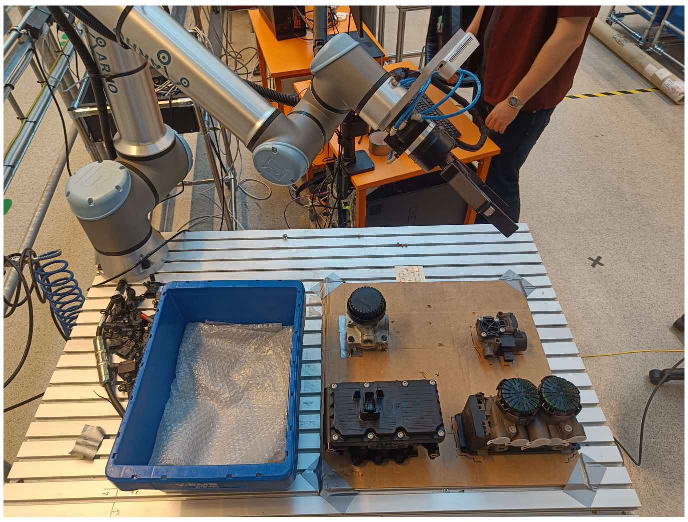


Figure 3.16: Objects all placed in their fixed poses for testing, in relation to the robot and the blue bucket.

The reason for these two different configurations is to see if the grasp synthesis models are able to generate grasps that would not lead the robot to collide with the other obstacles. This would be a common issue in bin picking if not careful. To mitigate this, we modelled the table, as well as all of the objects to be considered as obstacles when testing with grouped objects, whereas only the table would be considered when testing with single objects. This difference is shown in figure 3.17. For each test setup, the robot will attempt to perform a simple pick and place task, where the objects in the scene will be transported to the nearby blue bucket. The robot will begin from 5 different configurations, such that different perspectives and views of the scene can be achieved. For each configuration, the test was repeated 10 times, and the amount of successful grasps, as well as successful pick-and-place operations that were successful. For a robust grasp synthesis model, it was expected

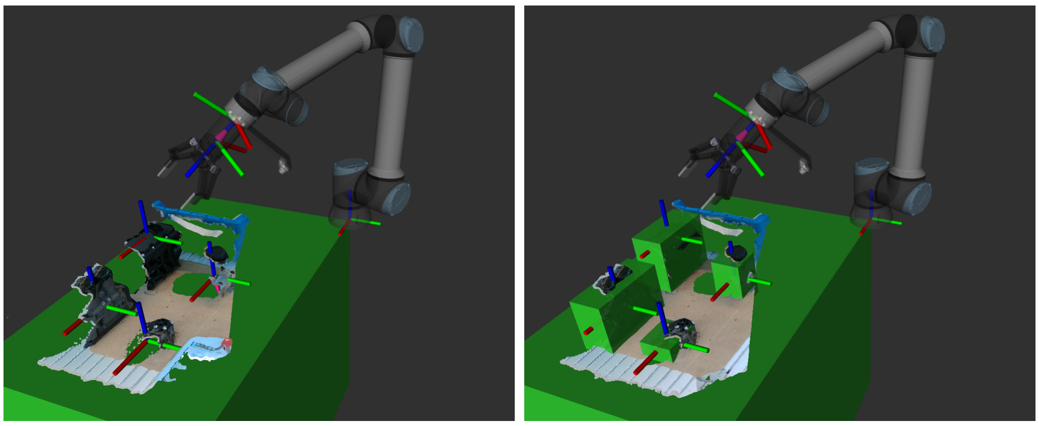


Figure 3.17: Pictures showing the difference between configurations as seen by MoveIt's planning scene. Right shows the grouped object setup, where the green boxes are modelled obstacles, whereas left shows tests using single objects (note the missing green boxes around the objects).

that these two values should be as close as possible, since a robust grasp entails that disturbances such as moving the arm during grasps should not let the object slip from the grip.

The metric most commonly used to measure the efficacy of grasp synthesis models is called Grasp Success Rate (GSR). In the research, Grasp Success Rate does not have a standard definition, and the authors choose what condition would count as a success. Since bin picking is the main application of these methods, it was deemed necessary to have the additional metric of determining the success of the pick-and-place task as well, even though it has more points-of-failure due to motion planners and implementation. However, the risk of additional failures due to implementation were deemed low enough to test and measure the success rate of the whole pick and place task. To avoid confusion, the pick and place success rate was labelled as Pick And Place Success Rate (PNPSR).

4

Results

4.1 Results from Pick and Place Operations

The following tables show the Grasp Success Rate (GSR) and Pick-And-Place Success Rate (PNPSR) for each category for both the offline synthesised method using GraspIt and the end-top-end deep learning method using Contact-GraspNet. Each cell represents the success rate after 50 attempts, 10 tries from each of the 5 configuration totalling to 1600 separately tested grasps, See section 3.16.

4.1.1 Graspit

As can be seen in table 4.1, grasp poses generated through GraspIt were not able to grasp or transport object 3 entirely, and had many difficulties with object 4.

	Single Object		Grouped Objects	
	G, few CP	G, More CP	G, few CP	G, More CP
Object 1	88 / 88	100 / 100	90 / 90	76 / 76
Object 2	60 / 60	100 / 100	82 / 82	78 / 78
Object 3	0 / 0	0 / 0	0 / 0	0 / 0
Object 4	8 / 8	0 / 0	16 / 6	0 / 0

Table 4.1: GSR (%) / PNPSR (%) results for the GraspIt based grasp-pose synthesis method. Contact points shown in figure. 3.7

For objects 1 and 2, it can be seen that GraspIt with less contact points on the gripper performs better than the Graspit with more contact points on the gripper only when there are multiple objects in the scene. The behaviour is reversed if only just one object is being grasped.

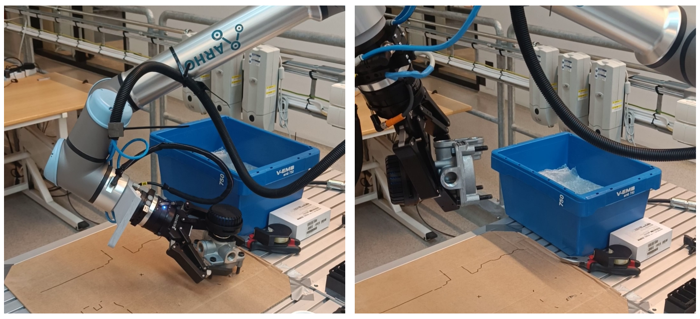


Figure 4.1: Example grasp of object 2 using GraspIt. Left shows point at which gripper was closed, right shows object at starting pose before being dropped off.

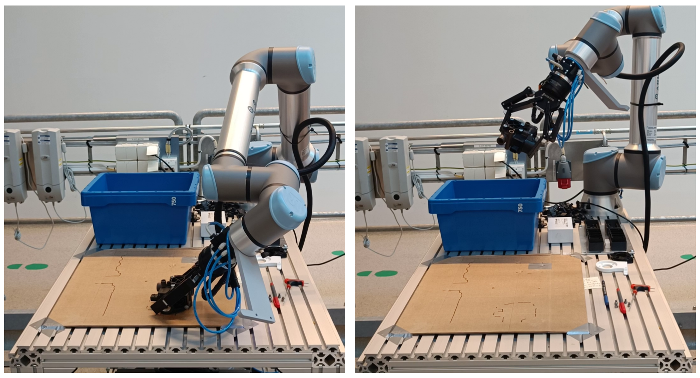


Figure 4.2: Example grasp of object 1 using GraspIt. Left shows point at which gripper was closed, right shows object at starting pose before being dropped off.

4.1.2 Contact-GraspNet

Table 4.2 shows that the Baseline model of Contact-GraspNet performs very well on both object 1 and 2. It also manages to pick object 3 and 4, despite the objects being very heavy and unwieldy. The retrained model has very poor performance, but the fact that it manages to generate suitable grasps for object 2 is still a good indication that retraining the model to accommodate a larger gripper width has potential. For grouped objects, both models are uniformly worse for all the objects. For our experiments, synthesising grasps took around 6-8 seconds on average.

	Single Object		Grouped Objects	
	CGN, Baseline	CGN, Retrained	CGN, Baseline	CGN, Retrained
Object 1	60 / 60	0 / 0	50 / 50	6 / 6
Object 2	92 / 90	46 / 42	74 / 72	18 / 18
Object 3	32 / 26	0 / 0	12 / 6	0 / 0
Object 4	18 / 8	0 / 0	0 / 0	0 / 0

Table 4.2: GSR (%) / PNPSR (%) results for the Contact-GraspNet based grasp-pose synthesis method, where "Baseline" is the model using the weights from the public Contact-GraspNet repository, and "Retrained" is the model using weights trained for the robotiq gripper

4.1.3 Special Cases

	CGN, Baseline	CGN, Retrained	G, Less-CP	G, More-CP
Successes Due to Object Slipping Into Stable Pose	6.41%	24.24%	10.78%	0%
Failures Due To No Valid Grasp Poses Were Synthesised	18.44%	26.70%	60.09%	94.17%

Table 4.3: Table showing the fraction of the successes that were due to the object slipping into a more stable configuration such that it would not fall, and the fraction of the cases where failures were due to no valid grasps being generated at all.

Given that tables 4.1 and table 4.2 are averages over all different iterations of all different starting poses, we amassed in table 4.3 some interesting cases that occurred during the testing. With some of the successes, where both the grasp and pick and place task was successful, we found that the grasp was not fully closed and the object was then able to slip into a more stable position such that it was transported to the drop off point. While we still marked that as a success, we kept track of these grasps. We can see how the retrained Contact-GraspNet model has the highest ratio of this scenario occuring, and how GraspIt with more contact points does not have a single case of this occurring.

Similarly, in the cases of failures, there were some cases where there were simply no valid grasp poses that were synthesised. These could show limitations of the synthesis model itself, and could possibly be remedied by better tuning of the model or some additional recovery strategies to perform in these scenarios. It is interesting to note here that GraspIt has the highest rates of this occurring.

5
Discussion & Conclusion

In this chapter, a thorough analysis of the results gathered during testing will be discussed. Eventual improvements to the system and future research topics will also be mentioned as well as a final conclusion and answer to the research question based on the analysis provided.

5.1 Discussion

5.1.1 Contact-GraspNet: Baseline vs Retrained

The baseline model for Contact-GraspNet performs surprisingly well considering the input scene is scaled down significantly. There were some clear issues with generating grasps for object 1, resulting in severely lower scored grasps, requiring us to lower the score threshold from the recommended 0.23 down to 0.13. Generally grasps that are generated are very consistent between tries. A high-scoring grasp from one perspective is usually considered a good grasp from a different perspective given that the contact point is visible from both perspectives. The retrained model performs far worse, which is to be expected given it is only trained on about 1/10 the data as the Baseline model. But given the small amount of training data, we are still impressed by the performance. The grasps generated are more sparsely spread around the object, while the baseline usually outputs more clustered grasps, the difference is illustrated in figure 5.1. Both models share the problem of having very high computational cost and slow inference time which can be partially reduced by a sparser depth image.

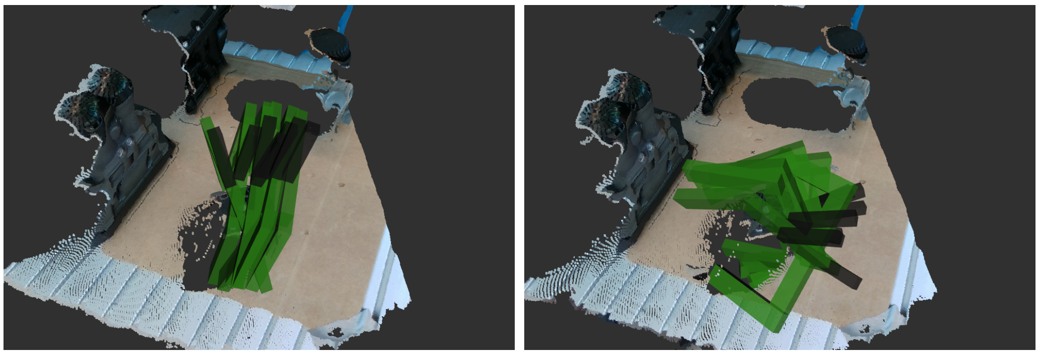


Figure 5.1: Figure showing the difference in clustering for generated grasps between the baseline model (left) and the retrained model (right) on the same object.

Baseline Contact-GraspNet actively generated grasps that avoid collision with other objects in the scene. This could be observed by the grasps generated on object 3 when object 4 and taken out of the scene, producing distinctly different grasps in both scenarios, this can be seen in figure 5.2.

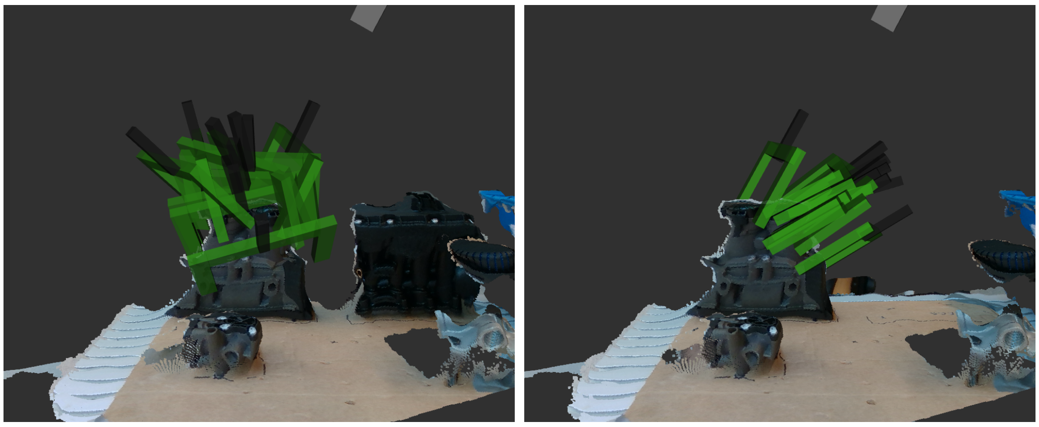


Figure 5.2: Baseline model generating grasps on object 3 with object 4 present in the scene (left) and without object 4 (right).

The retrained version of Contact-GraspNet proposed a lot of colliding grasps, especially on object 3 and 4 where grasps were generated from inside the object, suggesting that extending the training to more data significantly increases collision avoidance and the models intuition of object shape.

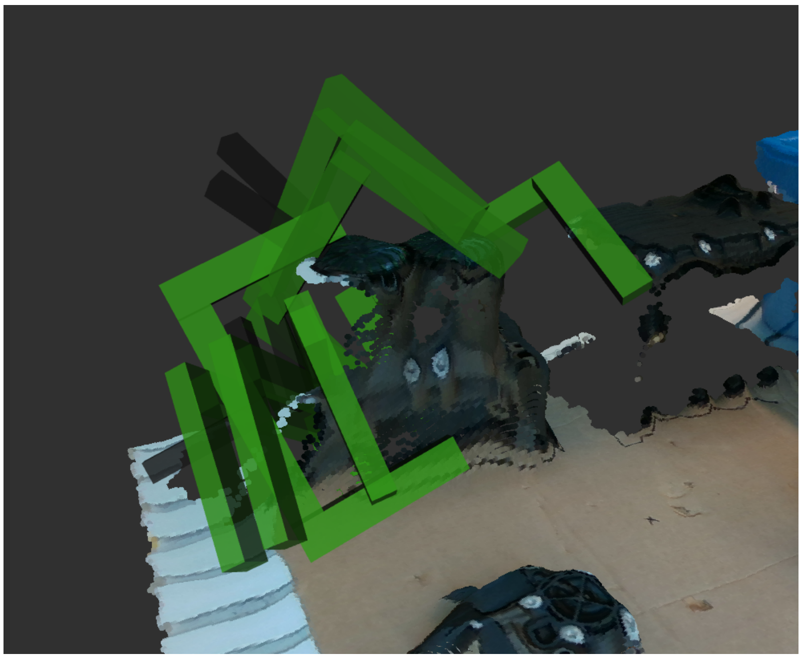


Figure 5.3: Retrained model generating grasps whose pose are located within the object.

5.1.2 Effect of Number of Contact Points for GraspIt

When more contact points are added onto the gripper in GraspIt, the performance on objects 1 and 2 shows clear improvement if there are no obstacles nearby. This is reasonable as it is able to get close to the object more often, whereas in the case of grouped objects, many of the previously valid grasps were discounted due to collisions with other objects. This highlights the fact of how important the filtering of grasps is, and how it needs to take into consideration both the grasps stored in the database, as well as the environment which the object is in.

Another point of interest is comparing how performance improved when using less contact points when there are obstacles. Having the grasp poses use only the finger pads as possible contact points lead to the generated grasp poses being further away, and hence more likely to generate some grasp pose that is both stable, and not hindered by any of the obstacles that we had placed. Given that there will be many obstacles in bin picking tasks (other objects in the bin, the bin itself, etc), this suggests that having contact points only on the finger pads would lead to better performance for bin picking and other pick and place tasks where objects are close to each other.

For both objects 3 and 4, its clear that neither configurations of contact points lead to stable performance in generating valid grasp poses. This can be attributed to the geometry and weight distribution of the object, making it highly sensitive to any disturbances in grasp poses, but also in the fact that GraspIt did not take into consideration any information about the actual weight of the object, and only the friction at the surface. This leads to grasps that *might* have worked for lighter objects, but is highly unstable for the ones that were available to test with.

During testing, some of the test configurations (start position, object being grasped, etc) showed high sensitivity to the object pose. Due to the fact that the objects were placed back into its original position manually, it may not have aligned perfectly with the estimated object pose. This *could* be translated as possible noise from a pose estimator model. What this showed was that some configurations and objects are more sensitive to this noise than others, and is highly dependent on how "stable" the final choice of grasp was. This goes to show how the filtering of grasps is important, such that only stable grasps should be chosen and not just any valid grasp.

5.1.3 Comparison between Models

We can see that both synthesis models had varying performance, and some are more fit than others in certain situations. While both models can be further optimised and tuned such that their performances are improved, it is clear that they already have their benefits and downsides.

Using GraspIt as the model for grasp synthesis means that the automation that Volvo wishes to introduce would be cheaper, given that it can run on weaker hardware at the kitting station, on the condition that more effort is placed on the filtering of grasps, and that engineer-hours are spent importing the necessary CAD files correctly into the GraspIt environment. Any new part that is added to the kitting station would necessitate an update to the grasp database and ensuring that the filtering of grasps is working and viable for that new object, which is a lot of time investment for something that happens constantly in manufacturing (which is the addition of new parts). This model would also require a precise 6 DoF object pose estimator, which is not trivial and would require more research into determining the best approach that would work for Volvo Group.

On the other hand, using Contact-GraspNet is more flexible and adaptive, given that it can estimate grasp poses for some never before seen objects. Given that the baseline model was never trained or tested on the objects we used in the tests, it had very good performance, and we hoped that retraining the model with the right gripper configurations would improve it, but due to the limitation of the hardware were not able to confirm. However, retraining with the full dataset is vital to achieve good performance on proprietary parts with complex shapes, since scaling down the point cloud with the baseline model will worsen the performance of the model with some small objects, and is only a temporary solution to show if the network architecture is a functional one. Given the performance of the baseline model, simply training it with the exact same dataset but with the correct gripper configuration is sure to improve the model performance, and therefore it is logical to conclude that adding the proprietary objects into the dataset would also improve the performance even more than that. This solution is also dependent on powerful hardware which would require powerful computers at the racks or the infrastructure for decentralised computing to be in place, which are both expensive options.

5.1.4 Effect of available setup

While we tried to implement the pipeline in such a fashion that it would allow us to measure the effectiveness of the grasp pose synthesis models, we were also limited with some of our hardware and software. The addition of more objects to GraspIt is time-consuming, and difficult to get correct, and since we had access to so few CAD files and even fewer actual objects, this lead to very few objects that we could test our grasp synthesis models with.

The limited variety in the data we collected also means that while we can show if there is potential in either solution for use in automating the kitting process for Volvo Group, we believe that our conclusions should be validated through more tests with more types of objects commonly found in the kitting process.

Due to a limited amount of time, the amount of parameters we could tune, test, and vary were limited. While we attempted to test the models using some varied configurations, other factors could affect their performance. Since we determine that a generated grasp is valid only if a plan could be found without colliding with obstacles, investigating different parameters and configurations in the navigation planners of MoveIt could improve performance. Lastly, since the deep learning model is highly dependent on the quality of its input, investigating the filters applied on the depth camera would also be beneficial.

5.2 Future Work

5.2.1 Improvements to the Grasp Synthesis Model

We have pinpointed some possible starting points for any future work that aims to improve the grasp synthesising models, and their use to automate the kitting process at Volvo Group. Retraining Contact-GraspNet with the full ACRONYM dataset should be the first step such that the right gripper configuration can be used when making inferences, after this point, transfer learning can be investigated such that a smaller dataset containing the proprietary objects are used to extend the newly trained Contact-GraspNet model, with possibly better performance as well.

One can also revisit newer grasp synthesis models that utilize deep learning in case models that are faster, easier to train, or are generally better than Contact-GraspNet can be found.

To improve GraspIt, an investigation on how to filter the grasps that are generated is worth the time. Exactly how that is done, either through Bayesian learning models, deep learning models, or some simple conditional statements, is left to the future investigator, and Volvo Group to determine if that is viable for their purposes.

Lastly, in order to fully be able to use these synthesis models to automate, a full object pose estimation and segmentation model must be implemented. The expectation is that these systems should be fully automated, and should be able to provide the grasp synthesis models with both object pose estimates and segmentation maps when needed, rather than being manually defined or manually select a region of interest.

5.2.2 Extensions and Improvements to the Pick and Place Pipeline

In order to ensure that the bin-picking tasks are performed well, we must ensure that the robot is able to find good plans to navigate towards the proposed grasp poses. One way to improve this is to extend the MoveIt implementation, such that it continuously can update its planning scene, and account for obstacles in a better fashion using the same depth camera as is mounted on the robot arm.

Another improvement could be on the planning algorithms themselves. A thorough investigation on a good planning algorithm for cartesian path planning of the end effector, as well as good detection of a valid path through the MoveIt interface would allow for safer automation, less time spent recovering from failed paths, and the ability to have people work closer to the robot.

Another possible improvement is to introduce collaborative behaviour to the robot. While the goal is that the robot would take the objects from bins and drop them off at a table or automated guided vehicles, there would still be humans nearby. Investigating how collaborative behaviour between the human and robot could best be implemented would allow the human to also take any required object from a nearby bin without risking any harm to the human.

5.3 Conclusion

Below is the research question we wished to answer through this thesis:

- Is a deep-learning model more viable than an analytical model for the purpose of grasp pose synthesis, given that the objects being picked are already known?

We find that while both solutions show high potential in succeeding, and that each have their benefits and downsides, the deep learning model would be more adaptive and future proof, given its quick and easy expandability and that it showed the highest success rate in our trials. However, given the already available infrastructure and previous attempts at automation tested at Volvo Group R&D, the analytical model would be the easiest and quickest to get implemented at the cost of more time spent each time a new object is to be added.

At the same time, we emphasize that before any development is made, more investigation needs to be done on improving both systems to ensure that these conclusions can be generalised to the entire Volvo Group kitting automation.

Bibliography

[1] R. X. Gao, J. Krüger, M. Merklein, H.-C. Möhring, and J. Váncza, "Artificial intelligence in manufacturing: State of the art, perspectives, and future directions," *CIRP Annals*, vol. 73, no. 2, pp. 723–749, Jan. 1, 2024, ISSN: 0007-8506. DOI: 10.1016/j.cirp.2024.04.101. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S000785062400115X (visited on 02/21/2025).

[2] A. Cordeiro, L. F. Rocha, C. Costa, P. Costa, and M. F. Silva, "Bin picking approaches based on deep learning techniques: A state-of-the-art survey," in *2022 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, Apr. 2022, pp. 110–117. DOI: 10.1109/ICARSC55462.2022.9784795. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9784795 (visited on 02/25/2025).

[3] M. Alonso, A. Izaguirre, and M. Graña, "Current research trends in robot grasping and bin picking," in *International Joint Conference SOCO'18-CISIS'18-ICEUTE'18*, M. Graña, J. M. López-Guede, O. Etxaniz, *et al.*, Eds., Cham: Springer International Publishing, 2019, pp. 367–376, ISBN: 978-3-319-94120-2. DOI: 10.1007/978-3-319-94120-2_35.

[4] R. Newbury, M. Gu, L. Chumbley, *et al.*, "Deep learning approaches to grasp synthesis: A review," *IEEE Transactions on Robotics*, vol. 39, no. 5, pp. 3994–4015, Oct. 2023, Conference Name: IEEE Transactions on Robotics, ISSN: 1941-0468. DOI: 10.1109/TRO.2023.3280597. [Online]. Available: https://ieeexplore.ieee.org/document/10149823 (visited on 01/21/2025).

[5] R. Platt, "Grasp learning: Models, methods, and performance," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 6, pp. 363–389, Volume 6, 2023 May 3, 2023, Publisher: Annual Reviews, ISSN: 2573-5144. DOI: 10.1146/annurev-control-062122-025215. [Online]. Available: https://www.annualreviews.org/content/journals/10.1146/annurev-control-062122-025215 (visited on 01/17/2025).

[6] "Robotiq gripper product sheet." (), [Online]. Available: https://blog.robotiq.com/hubfs/Product-sheets/Adaptive%20Grippers/Product-sheet-Adaptive-Grippers-EN.pdf (visited on 02/26/2025).

[7] D. Morrison, P. Corke, and J. Leitner, *Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach*, May 15, 2018. DOI: 10.48550/arXiv.1804.05172. arXiv: 1804.05172[cs]. [Online]. Available: http://arxiv.org/abs/1804.05172 (visited on 05/13/2025).

[8] M. Sundermeyer, A. Mousavian, R. Triebel, and D. Fox, "Contact-GraspNet: Efficient 6-DoF grasp generation in cluttered scenes," in *2021 IEEE Inter-*

national Conference on Robotics and Automation (ICRA), ISSN: 2577-087X, May 2021, pp. 13 438–13 444. DOI: 10.1109/ICRA48506.2021.9561877. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9561877 (visited on 01/22/2025).

[9] H.-S. Fang, C. Wang, M. Gou, and C. Lu, "GraspNet-1billion: A large-scale benchmark for general object grasping," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, ISSN: 2575-7075, Jun. 2020, pp. 11 441–11 450. DOI: 10.1109/CVPR42600.2020.01146. [Online]. Available: https://ieeexplore.ieee.org/document/9156992 (visited on 01/20/2025).

[10] J. Borja-Diaz, O. Mees, G. Kalweit, L. Hermann, J. Boedecker, and W. Burgard, *Affordance learning from play for sample-efficient policy learning*, Mar. 1, 2022. DOI: 10.48550/arXiv.2203.00352. arXiv: 2203.00352[cs]. [Online]. Available: http://arxiv.org/abs/2203.00352 (visited on 01/17/2025).

[11] Z. Zhao, H. Yu, H. Wu, and X. Zhang, "Bio-inspired affordance learning for 6-DoF robotic grasping: A transformer-based global feature encoding approach," *Neural Networks*, vol. 171, pp. 332–342, 2024. DOI: 10.1016/j.neunet.2023.12.005.

[12] Z. Chen, Z. Liu, S. Xie, and W.-S. Zheng, "Grasp region exploration for 7-DoF robotic grasping in cluttered scenes," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, ISSN: 2153-0866, Oct. 2023, pp. 3169–3175. DOI: 10.1109/IROS55552.2023.10341757. [Online]. Available: https://ieeexplore.ieee.org/document/10341757 (visited on 01/17/2025).

[13] C. Zhuang, H. Wang, W. Niu, and H. Ding, "A parallel graph network for generating 7-DoF model-free grasps in unstructured scenes using point cloud," *Robotics and Computer-Integrated Manufacturing*, vol. 92, p. 102 879, Apr. 1, 2025, ISSN: 0736-5845. DOI: 10.1016/j.rcim.2024.102879. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0736584524001662 (visited on 01/17/2025).

[14] H. Wang, W. Niu, and C. Zhuang, "GraNet: A multi-level graph network for 6-DoF grasp pose generation in cluttered scenes," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, ISSN: 2153-0866, Oct. 2023, pp. 937–943. DOI: 10.1109/IROS55552.2023.10341549. [Online]. Available: https://ieeexplore.ieee.org/document/10341549 (visited on 01/17/2025).

[15] C. Eppner, A. Mousavian, and D. Fox, *ACRONYM: A large-scale grasp dataset based on simulation*, Nov. 18, 2020. DOI: 10.48550/arXiv.2011.09584. arXiv: 2011.09584[cs]. [Online]. Available: http://arxiv.org/abs/2011.09584 (visited on 02/04/2025).

[16] K. Kleeberger, F. Roth, R. Bormann, and M. F. Huber, "Automatic grasp pose generation for parallel jaw grippers," in *Intelligent Autonomous Systems 16*, M. H. Ang Jr, H. Asama, W. Lin, and S. Foong, Eds., Cham: Springer International Publishing, 2022, pp. 594–607, ISBN: 978-3-030-95892-3. DOI: 10.1007/978-3-030-95892-3_45.

[17] J. P. Carvalho de Souza, C. M. Costa, L. F. Rocha, *et al.*, “Reconfigurable grasp planning pipeline with grasp synthesis and selection applied to picking operations in aerospace factories,” *Robotics and Computer-Integrated Manufacturing*, vol. 67, p. 102 032, Feb. 1, 2021, ISSN: 0736-5845. DOI: 10.1016/j.rcim.2020.102032. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S073658452030243X (visited on 01/23/2025).

[18] A. Miller and P. Allen, “Graspit! a versatile simulator for robotic grasping,” *IEEE Robotics & Automation Magazine*, vol. 11, no. 4, pp. 110–122, Dec. 2004, Conference Name: IEEE Robotics & Automation Magazine, ISSN: 1558-223X. DOI: 10.1109/MRA.2004.1371616. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1371616 (visited on 02/03/2025).

[19] M. Ciocarlie, C. Goldfeder, and P. Allen, “Dimensionality reduction for hand-independent dexterous robotic grasping,” in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, ISSN: 2153-0866, Oct. 2007, pp. 3270–3275. DOI: 10.1109/IROS.2007.4399227. [Online]. Available: https://ieeexplore.ieee.org/document/4399227 (visited on 05/04/2025).

[20] L. Ingber, “Very fast simulated re-annealing,” *Mathematical and Computer Modelling*, vol. 12, no. 8, pp. 967–973, Jan. 1, 1989, ISSN: 0895-7177. DOI: 10.1016/0895-7177(89)90202-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0895717789902021 (visited on 05/04/2025).

[21] “ROS 2 documentation — ROS 2 documentation: Humble documentation.” (), [Online]. Available: https://docs.ros.org/en/humble/index.html (visited on 02/05/2025).

[22] “Tf2 — ROS 2 documentation: Humble documentation.” (), [Online]. Available: https://docs.ros.org/en/humble/Concepts/Intermediate/About-Tf2.html#paper (visited on 05/08/2025).

[23] “MoveIt motion planning framework.” (), [Online]. Available: https://moveit.ai/ (visited on 02/05/2025).

[24] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, *PointNet++: Deep hierarchical feature learning on point sets in a metric space*, Jun. 7, 2017. DOI: 10.48550/arXiv.1706.02413. arXiv: 1706.02413[cs]. [Online]. Available: http://arxiv.org/abs/1706.02413 (visited on 02/03/2025).

[25] A. X. Chang, T. Funkhouser, L. Guibas, *et al.*, *ShapeNet: An Information-Rich 3D Model Repository*, arXiv:1512.03012 [cs], Dec. 2015. DOI: 10.48550/arXiv.1512.03012. [Online]. Available: http://arxiv.org/abs/1512.03012 (visited on 05/13/2025).

[26] M. Savva, A. X. Chang, and P. Hanrahan, “Semantically-enriched 3D models for common-sense knowledge,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, ISSN: 2160-7516, Jun. 2015, pp. 24–31. DOI: 10.1109/CVPRW.2015.7301289. [Online]. Available: https://ieeexplore.ieee.org/document/7301289 (visited on 05/13/2025).

[27] A. Kirillov, E. Mintun, N. Ravi, *et al.*, *Segment Anything*, arXiv:2304.02643 [cs], Apr. 2023. DOI: 10.48550/arXiv.2304.02643. [Online]. Available: http://arxiv.org/abs/2304.02643 (visited on 05/13/2025).

[28] G. Bradski, “The OpenCV library.,” *Dr. Dobb’s Journal: Software Tools for the Professional Programmer*, vol. 25, no. 11, pp. 120–123, 2000, ISSN: 1044-789X. [Online]. Available: `https://www.elibrary.ru/item.asp?id=4934581` (visited on 02/26/2025).

[29] J. Buehler, *JenniferBuehler/graspit-pkgs*, original-date: 2016-02-22T15:18:43Z, Mar. 13, 2025. [Online]. Available: `https://github.com/JenniferBuehler/graspit-pkgs` (visited on 05/08/2025).

[30] “Universal robots a/s · GitHub.” (), [Online]. Available: `https://github.com/UniversalRobots` (visited on 02/26/2025).

[31] “UR10e collaborative industrial robot - payload up to 10 kg.” (), [Online]. Available: `https://www.universal-robots.com/se/produkter/ur10-robot/` (visited on 05/08/2025).

[32] “Adaptive grippers | robotiq.” (), [Online]. Available: `https://robotiq.com/products/adaptive-grippers` (visited on 05/08/2025).

A

Appendix 1

A.1 Custom Service Files

In this section is the full service file that describes the input and output of the custom services.

A.1.1 GenerateGrasp.srv

```
# Request
string object_name
uint8 attempt_number
---
# Result
bool grasp_synthesis_successful
uint8 num_poses
geometry_msgs/Pose[] grasp_poses
uint16[] grasp_pose_idxs
```

A.1.2 PickAndPlace.srv

```
# Request
string[] object_names
string synthesis_method  # database, or deep_learning
---
# Result
bool operation_successful
```

A.1.3 SetDesiredPoseFrame.srv

```
# Request
string link_to_move_ee_to
bool plan_cartesian_path
bool wait_for_execution
bool reset_planning_scene
string object_name
---
# Result
bool planning_successful
```

A.1.4 SetGripperState.srv

```
uint8 OPEN=1
uint8 CLOSE=2

# Request
uint8 gripper_state

---
# Result
bool service_call_successful
```