Tampere University

Akber Ali Khan

DEEP LEARNING FOR OBJECT DETECTION

Training Data Generation using Parametric CAD Modelling and Gazebo Simulation

Master of Science Thesis
Faculty of Engineering & Natural Sciences
Associate Prof. Roel Pieters
Kulunu Samarawickrama
November 2021

ABSTRACT

Akber Ali Khan: Deep Learning for Object Detection: Training Data Generation using Parametric CAD Modeling and Gazebo Simulation
Master of Science Thesis
Tampere University
Master's Programme in Automation Engineering
November 2021

Deep learning-based object detection and pose estimation methods need a large number of synthetic data for application in robotic assembly tasks. The acquisition of such data from real objects tends to be arduous, erroneous, and time-consuming. Alternatively, synthetic data can be generated autonomously from 3D models efficiently and relatively quickly in a simulated environment. These 3D models can be generated by utilizing either conventional or parametric approaches. Conventional approaches generate free-form mesh models that are generally unalterable when repetitive changes are required in the models, which is an important aspect in parts customization in an industrial context. This challenge is addressed by implementing a script-based parametric modelling approach to automate the generation of 3D models of an industrial part via parameters. Then, the 3D models of the dataset are loaded in the simulation environment for synthetic data generation to train and evaluate a state-of-the-art model-based pose estimation network for 6DoF object pose estimation. This thesis comprehensively illustrates the implementation of automated parametric modelling of an industrial part to create a dataset of CAD models, generate synthetic data for deep learning-based object detection methods, and compute the 6DoF poses of the dataset objects in a cluttered scene using a state-of-the-art pose estimation method. The results of the computation speed for generating and rendering the models are analysed. Finally, the study analyses the results of the benchmark 6DoF pose estimation network evaluated for 6DoF poses of the custom dataset objects.

Keywords: synthetic data, deep learning, parametric modelling, object detection, pose estimation

PREFACE

This has been an exciting and enriched experience while pursuing my master's degree at Tampere university.

I am sincerely thankful to my supervisor, Associate Prof. Roel Pieters, for his consistent support and guidance throughout this thesis work. I feel grateful to Kulunu Samarawickrama, PhD researcher at Tampere university, for assisting me with the troubleshooting during the entire research work.

It is worth mentioning here that it would not have been possible to complete my masters without the support of my parents.

Tampere, 26 November 2021

Akber Ali Khan

CONTENTS

1. INTRODUCTION 1
1.1 Overview 1
1.2 Thesis Structure 3
2. BACKGROUND 4
2.1 Parametric Modeling 4
2.2 Analysis of Script-based Parametric CAD Modelers 5
2.3 3D Modeling Paradigm with FreeCAD 8
2.3.1 Using Python Console 10
2.3.2 Creating Macro Script 10
2.3.3 Integrating External Workbenches 11
2.4 Vision-based Pose-Estimation 11
2.5 Model-based Learning 12
2.5.1 Correspondence-based Learning 13
2.5.2 Template-based Learning 14
2.5.3 Voting-based Learning 16
2.6 Model-free Learning 17
2.7 Point Cloud-based Approaches 17
2.7.1 Point Cloud-based Feature Extraction 19
2.7.2 Point Cloud-based Pose Estimation 23
2.7.3 Point Cloud-based Grasp Detection 24
3. METHODOLOGY 25
3.1 Overview 25
3.2 Automation of Parametric Gear Modeling 26
3.3 Integrating FreeCAD with Data Generation Pipeline 28
3.4 Synthetic Data Generation from CAD Models 29
3.5 6-DoF Pose Estimation for Multi-Class Objects 31
3.5.1 6-DoF Pose Estimation 31
3.5.2 PVN3D Network Architecture 31
3.5.3 Network Optimization 32
3.5.4 Network Training 34
3.6 Least-Squares-Fitting for Pose Estimation 35
4. RESULTS 36
4.1 Gear Dataset Generation 36
4.2 Training Dataset Generation Results 37
4.3 Parametric Modeling Computation Time 39
4.4 Pose Estimation Evaluation 40
5. DISCUSSION 43
5.1 Parametric Modeling Automation 43
5.2 Synthetic Data Generation 43

5.3 Network Training in CSC 44
6. CONCLUSION 46
6.1 Achieving Research Objectives 46
6.2 Delimits and Future Works 47
REFERENCES 48

LIST OF FIGURES

Figure 1. The CAD Design process comparison: Parametric versus Conventional modeling [48] 5
Figure 2. FreeCAD 3D Modeling example 9
Figure 3. Macro script generation in FreeCAD. 10
Figure 4. Vision based robot grasping System [13] 11
Figure 5. Correspondence-based learning methods [12] 13
Figure 6. Template-based learning methods [12] 15
Figure 7. Voting-based learning methods 16
Figure 8. Workflow diagram of point cloud-based grasp estimation [13] 18
Figure 9. Point cloud-based feature-extraction techniques [34] 20
Figure 10. Extraction of features using point-based approaches [37] 21
Figure 11. PointNet and PointNet++ architectures workflow 22
Figure 12. The GraspNet network [42] 24
Figure 13. Workflow of methodology 25
Figure 14. Synthetic data generation using hemisphere sampling in gazebo simulation 30
Figure 15. PVN3D functional diagram [30]. 31
Figure 16. Automation of parametric involute gear CAD models dataset 36
Figure 17. RGB, mask and depth image samples from 3 different viewpoints. 38

LIST OF TABLES

Table 1. Characteristics of some popular FOSS parametric CAD modeling tools [6].....7
Table 2. Parametric CAD Models computation time.....40
Table 3. 6-DoF pose estimation accuracies for custom gear test dataset.....42

LIST OF SYMBOLS AND ABBREVIATIONS

API	Application Programming Interface
CAD	Computer Aided Design
DL	Deep learning
DNN	Deep neural network
DoF	Degrees of Freedom
FOSS	Free Open-Source Software
GUI	Graphic User Interface
RGB	Red green blue
RGB-D	Red green blue- depth (combination of RGB and depth image)
STEP	Standard for Exchange of Product
2D	Two dimensional
3D	Three dimensional

1. INTRODUCTION

1.1 Overview

Some of the most sophisticated and complex robotic tasks such as object detection, pose estimation, and robot grasping require robots to learn from data for the application of machine learning. These robotic tasks are presented in the context of Agile Manufacturing or Production where the robots are required to be agile and adaptable with new tasks and target objects [1]. Fortunately, implementation of these tasks has become possible due to the evolving and latest machine learning approaches, which enable robots to learn from real or simulated data. Usually, for training a DL-based model, a large amount of data is needed that can be obtained from real or simulated objects. Conventionally, the data obtained from real objects with real sensors, such as RGB-D cameras, is quite tedious, time-consuming, and impractical, at least in the context of industrial applications. Alternatively, to solve this issue, simulation techniques can be utilized to automate the generation of training data from a CAD model of a part, which is the objective of this thesis.

One way of generating such an automated dataset is by utilizing parametric CAD models. By altering the parameters of the models, a variety of models of the same design can be generated. Later, each of the CAD models of a part can be loaded in the Gazebo simulator, simulated with a camera at a certain pose, take images, change the camera pose, and repeat the process to generate training data. Additionally, other variables can be changed iteratively such as lighting, color, or texture of objects. Eventually, this data set is used to train the object detection model.

Traditional CAD tools capture subsequent operations on CAD design as a construction sequence, whereas parametric modeling aims to enable changes in a design on selected features or constraints. Therefore, parametric models enable the automation of repeated changes which is important for the customization of parts in industrial applications.

Furthermore, among the parametric modelers, many share the source file for the design. The key is to share the design with other software without losing important information [4]. Additionally, sharing the source file of a design makes it possible to render the script in other CAD modelers for modifying the design. For that purpose, analysis of different parametric modeling tools is required before selecting the one which best fits the purpose.

There are two approaches to generate the training dataset.

- **Training Data Generation from Real Objects**

  In this approach, an RGB-D camera is aimed at the real object to capture images from multiple angles. However, this is a conventional approach and does not automate data generation. Generating training data through this approach is quite arduous, time-consuming, and computationally expensive. For this reason, this approach has not been considered for this research as the purpose was to generate the data automatically.

- **Automatic Training Data Generation**

  In contrast to the approach discussed above, this approach utilizes parametric CAD tools to create a CAD model of a part and render it into a simulation environment to automate the training dataset generation. Consequently, this approach is easier and more efficient. In addition, different parameters can be varied during simulation time by using programming scripts.

Keeping in view the automatic training data generation approach above, this thesis aims to achieve the following objectives:

- To analyze script-based parametric modelers and explore their characteristics.
- To generate script-based parametric CAD models of a gear part, simple involute gear in this case, by looping through the parameters.
- To integrate the parametric modeler with the data generation pipeline.
- To evaluate the custom parametric gearset for pose estimation accuracies.

1.2 Thesis Structure

This thesis comprises of six chapters.

Chapter 1, Introduction, provides a general overview of the thesis topic, research objectives, and thesis organization.

Chapter 2, Background, provides an analysis and comparison of different script-based parametric CAD tools. In addition, state-of-the-art robotic pose-estimation and grasp detection methods are discussed.

Chapter 3, Methodologies, discuss the methods to automate the parametric involute gear CAD modeling and integration of parametric CAD modeler with the data generation pipeline. It also illustrates the generation of training dataset for custom gearset rendered in gazebo simulation environment and training a deep learning network for pose-estimation.

Chapter 4, Results, tabulates the computation time for generating and exporting parametric gear models. It also evaluates the pose-estimation accuracies of the custom gear models.

Chapter 5, **Discussion**, discusses the parametric modeling, data generation, and pose-estimation procedures in detail.

Chapter 6, **Conclusion**, concludes the thesis with conclusions and remarks. It discusses delimits of the thesis and future works.

2. BACKGROUND

2.1 Parametric Modeling

In manufacturing industries, modification of design models is often required during design exploration where regeneration of parts design is carried out according to need [2]. One such example is gear, where certain features should remain the same when the overall design is altered, for instance, the profile and dimension of gear teeth. Some of the parameters that can be altered in a gear design are the number of teeth, module size, gear height, beta (helix angle). By varying these parameters, a variety of gear models can be generated.

Therefore, solid CAD modeling tools can be used to generate such alterable models and these modelers are of two types: Conventional or Free-form mesh and parametric modelers. Conventional modeler uses a direct approach, without utilizing parameters in their designs. Moreover, two-dimensional (2D) sketches are not fundamentally required to generate three-dimensional (3D) models. Therefore, pre-set constraints are neglected in the design. However, the later modeler is parameter-based and pre-defines constraints during the 2D sketching phase. There are at least three advantages of using parametric modeling:

i. Geometry reusability for later stages
ii. Propagate alteration in a design or model automatically
iii. Knowledge of manufacturing with geometry [3]

Such limitations force the free form mesh modelers to use parameters and constraints in the design and the 3D models cannot be modified by others. For that reason, the free form mesh modeler is not related to the research purpose of this thesis, so it has not been discussed in the future sections. The comparison

between the design process of free form mesh models and parametric models is shown in Figure 1.

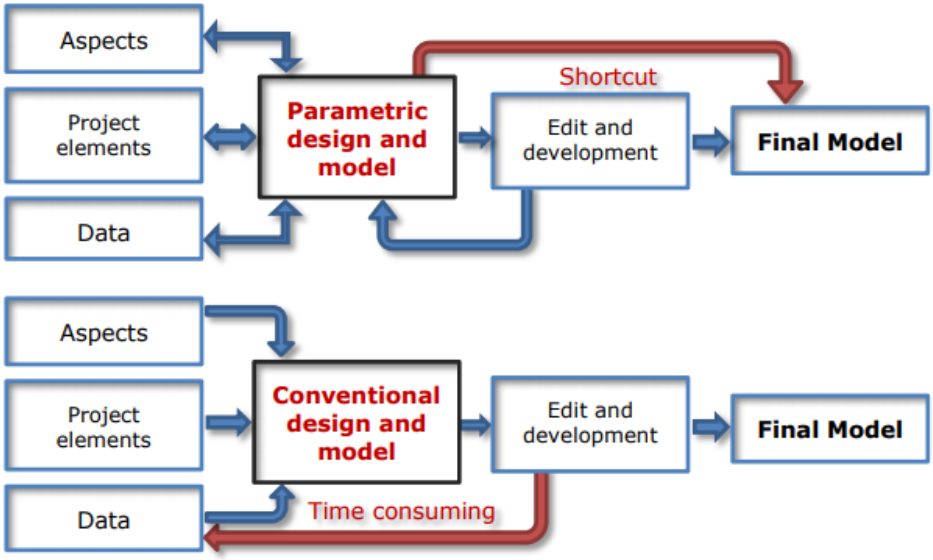


Figure 1. *The CAD Design process comparison: Parametric versus Conventional modeling [50]*

Figure 1 shows that parametric CAD modelers generate dynamic and flexible models as compared to conventional design tools and minimize the effort for modification. This enables the designer to make quick changes whenever necessary. Along with direct manipulation and custom featuring, they also provide scripting which can ease the alteration using transaction sequences [2] [4]. Moreover, some parametric modelers can export standard parametric CAD files, such as STEP [5] formats and it is sometimes required in other modelers for modification.

Thereby, it is appropriate to only consider parametric modelers with the option to use scripts [6]. Some parametric CAD modelers with scripting capabilities are OpenSCAD [7], FreeCAD [8], Cadquery [9], PythonOCC [10], ImplicitCAD, and OpenJSCAD [11]. A detailed analysis of these tools is presented in the next section.

2.2 Analysis of Script-based Parametric CAD Modelers

According to Machado et al. [6], many modern CAD tools can render or export the standard parametric CAD file, thereby allowing the model to be opened in any other modeler for further modification without losing important features of the

design. Next, we discuss some of the very common free-open-source script-based parametric models.

OpenJSCAD [11] can be used via command line, browser to generate 3D parametric designs; utilizes JavaScript programming language, and it is commonly used for 3D printing applications. Similarly, Implicit CAD also generates 3D models using JavaScript. However, neither of these two modelers can export STEP files [6].

FreeCAD generates 3D models in boundary representation (B-rep), and it is completely python-based with a variety of Application Programming interfaces (APIs) available for 3D modeling. Apart from GUI, Solid modeling in FreeCAD, using python can be done in three ways: Typing commands in the FreeCAD python console, creating macro files, using external workbenches or scripts through FreeCAD API. This provides the user with flexibility and ease of usage. Moreover, FreeCAD can export STEP files. The official documentation provided by FreeCAD for python scripting is not well organized, thus it is not easy to design complex 3D models. However, python is easier than other programming languages and it provides leverage to non-expert programmers to understand it better as compared to the other 3D modeling languages.

Another popular parametric 3D modeler is OpenSCAD which performs its 3D computation by using Constructive Solid Geometry (CSG). Geometric primitives such as a box, sphere, cylinder, are used by OpenSCAD script to perform Boolean operations to construct a 3D model. OpenSCAD programming language has functional language, and its syntax looks like C-language. However, like many other CAD tools, it is unable to export STEP. Another significant drawback of this tool is the lack of a GUI model editor for design modification, so the only way to edit models is through the script. Since OpenSCAD has inadequate functions and primitive objects, it is simple to learn for novices. In addition, OpenSCAD also provides easy-to-follow tutorials and documentations for beginners to learn the software with minimal effort.

Python Open Cascade (PythonOCC) is similar to FreeCAD, and it offers advanced topological and geometric operations. Although, it can export STEP

files, but it has no GUI interface available for the user [6] [10]. Nevertheless, this is a disadvantage for users with limited programming experience [6].

Ballistic Research Laboratory-CAD modeler is also based on constructive solid geometry (CSG) and supports numerous primitive shapes which are used through Boolean operations to create complex and complicated models [12]. Due to its complicated tools, it is quite expert-oriented software, mostly used by experienced CAD designers.

The main purpose of the Cadquery library is to reduce the number of codes as compared to conventional FreeCAD programming. There are two versions of Cadquery to date: Cadquery v1.2 and Cadquery v2.0. The former version can be either used as a workbench through FreeCAD API. In addition, it can be integrated with FreeCAD's graphical interface like normal, whereas the latter version is a stand-alone external tool that can be installed for project usage in three different ways as described in FreeCAD's official GitHub repository. Both versions have the STEP export capability [9] [6].

Table 1 below summarizes the different characteristics, such as the ability to export standard parametric files, 3D modeling interface type, programming language, and learning curve of seven different parametric CAD modelers.

Table 1. *Characteristics of some popular FOSS parametric CAD modeling tools [6]*

Parametric CAD Tool	STEP Export	3D Modeling Interface	Programming Language	Learning Curve
OpenJSCAD	No	Script-based	Javascript	High
Implicit CAD	No	Script-based	OpenSCAD language interpreter	High
FreeCAD	Yes	GUI + Script-based	Python	Medium
PythonOCC	Yes	Script-based	Python	High
OpenSCAD	No	Script-based	Functional language	Easy
BRL-CAD	Yes	Script-based	Embedded	Very high
CADQuerry v1.2	Yes	Script-based	Python	High
CADQuerry v2.0	Yes	Script-based	Python	High

In Table 1, it can be observed that only FreeCAD provides both graphical and script-based modeling interfaces for programmers. OpenSCAD is relatively easy to learn, but it does not export STEP nor provides a graphical modeling interface. Most of the other modelers have numerous limitations except FreeCAD. Learning OpenSCAD is easier as compared to other script-based modeling tools, but it does not provide a graphical interface for modeling, nor it exports standard parametric files for rendering. Although learning FreeCAD is arduous as compared to OpenSCAD, it provides more advantages for designers [6]. Consequently, FreeCAD seems a more reasonable parametric modeler to fulfill the objective of this thesis.

2.3 3D Modeling Paradigm with FreeCAD

In FreeCAD 3D parametric models can be generated either using graphical interface or python scripts, even in parallel. Many 2D and 3D tools are already available in the FreeCAD in the form of workbenches. By default, these tools are integrated into every FreeCAD installation. Some common workbenches are sketcher, part, part design, and some GUI-based workbenches.

Sketcher workbench is used as a starting point for generating any 3D model from scratch. Geometric constraints are set in the sketching phase. It is responsible for generating 2D geometries used for part and part design workbenches in the later stages. First, sketches are extruded to generate 3D shapes. Later, the 3D shape can be further modified by using part and part design features such as an extrusions, holes, pockets, fillets, and chamfers. These features can be used both in the graphical interface as well as python scripts.

An empty or named document needs to be created before writing python codes for a new 3D model. This can be done by simply typing the following commands in the python console or macro python scripts:

```
>>  DOC = FreeCAD.newDocument("DocumentName")
```

>> $DOC.recompute()$

This command creates a new FreeCAD document and all 2D or 3D objects are attached to this document for further operations. To render the model in the graphical interface for visualization it is important to recompute the document.

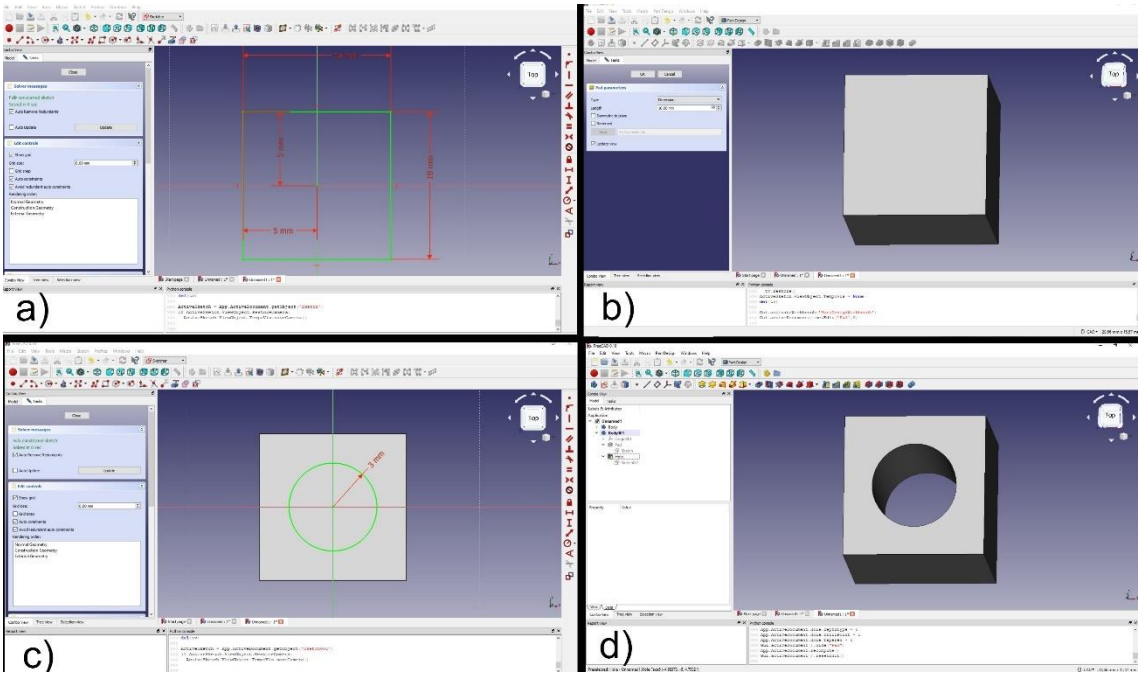


Figure 2. *FreeCAD 3D Modeling example*

Figure 2 shows a simple 3D model for a cube with a cylindrical hole. As one can notice that the first step is to generate a 2D geometry for a cube that is square with constraints such as length and distance from the origin. This shape is then extruded to form a cube. In the next step, a circle with constraints, radius, is created on the top face of the cube. Using a hole feature from the part design workbench, a cylindrical hole is created with the depth of the hole equal to the height of the cube. Thus, a cube with a cylindrical hole is created using sketches, part design, and part workbenches.

2.3.1 Using Python Console

Python codes can be directly typed in the FreeCAD's python console in an interactive way to generate immediate output on the graphical interface. This is not an efficient way to write codes for a 3D model but helps in debugging and troubleshooting.

2.3.2 Creating Macro Script

Apart from the python console, python scripts can be generated in FreeCAD by using Macros. Generally, the macro is used to record the graphical interface actions into python codes. This is an efficient method to generate python codes for complicated models through the graphical interface as well as by typing python codes in the macro editor to generate 3D models. All the constraints and parameters can be set in the graphical interface or python script to automate the modeling process.

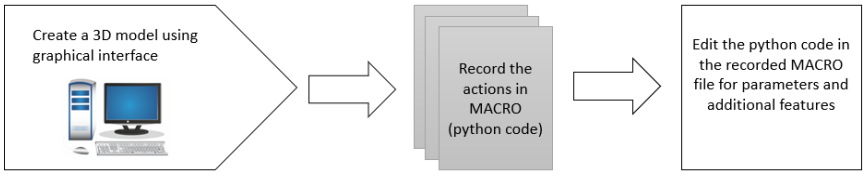


Figure 3. *Macro script generation in FreeCAD.*

Macro scripts are generated by recording the 3D modeling process that is performed through the graphical user interface as shown in Figure 3. While modeling, every GUI command is stored in this script as python code. Each graphical interface command is a python code and can be visualized on the python console as well. After finishing the model, the macro needs to be stopped to avoid storing redundant codes.

The recorded macro codes are hardcoded since constants have been used to set the 2D geometries and constraints. Variables can be introduced to substitute constant values. By this approach the number of codes in the script is reduced, as well as parameters are included which can be altered to modify the model quickly.

2.3.3 Integrating External Workbenches

External workbenches can also be used in FreeCAD. For instance, Cadquery v1.2 can be used in FreeCAD API as a workbench with its own code editor and the graphical interface of FreeCAD thus becomes available for visualizing the models.

However, the latest version Cadquery v2.0 is stand-alone software with a graphical interface for displaying 3D objects. Since it is based on pythonOCC, it does not work in FreeCAD API.

2.4 Vision-based Pose-Estimation

The purpose of vision-based pose estimation is to estimate a viable object pose for the robot to execute human-like object grasping. In this regard, Du et al. [13] have summarized the key tasks for robotic grasping as, localization, pose-estimation, and grasp-estimation of the target objects. The taxonomy of vision-based robotic grasping is shown in the figure below.

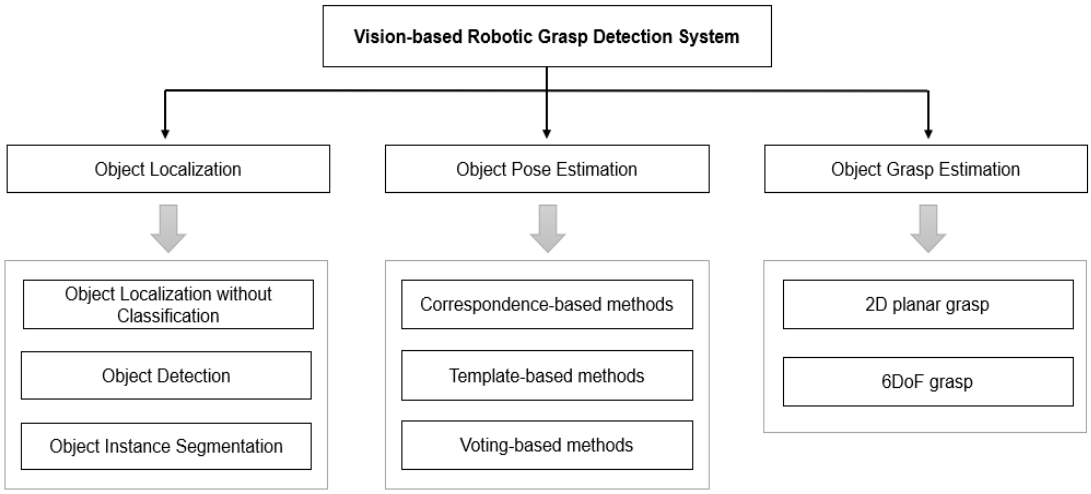


Figure 4. *Vision based robot grasping System [13]*

Localization generally provides the target object regions within the visual input data [13]. Further, there are three types, each with different purposes and applications as shown in Figure 4. Classification-based object localization is category agnostic and only provides the regions with potential target objects. On the other hand, object detection detects all the target objects categorically and

draws a bound box around them. Contrarily, object instance segmentation detects the points or pixel level areas of the object with the respective categories.

The main goal of object pose estimation is to find the 6D pose to assists the robot to compute the target object's 3D position and 3D orientation. The 6D object poses can be retrieved by three methods which are correspondence, template, and voting based methods. Each method has been discussed in detail in the subsequent sections. [13]

In the last few years, the issue of pose estimation is dealt with as a machine learning-based task. All the state-of-the-art machine learning algorithms, such as probabilistic, reinforcement, or deep learning methods, are data-driven approaches. Hence, these methods learn from data, either real data or synthesized data, and the basic idea is to train a machine learning model with the data acquired from the object. The earlier approaches require object-specific parameter tuning for novel objects, which is a complicated and exhausting task. However, learning-based methods do not require object-specific parameter tuning, rather the learning models are trained on a huge number of synthetic data generated in simulations to get the optimum 6DoF pose estimation further extended to grasp manipulation for robotic tasks. [14].

Based on the previous knowledge about the object, learning-based methods fall into two categories explained in the next sections.

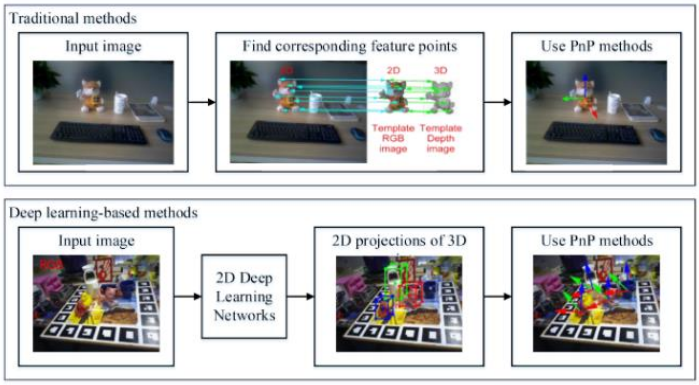
2.5 Model-based Learning

Model-based learning for grasp-estimation requires an appropriate CAD model of the target object to learn object features. The grasp detection is computed from the pose estimation of the CAD models in the reference camera coordinate [14].

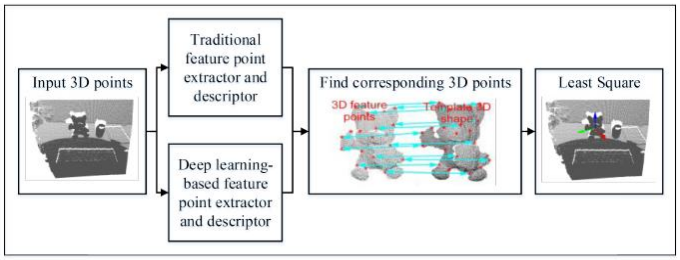
These methods have proven to be robust to occlusions, lighting, and occasionally scale invariant as discussed in various studies [15] [16] [17]. Based on various techniques, the model-based learning method can be further extended to the following.

2.5.1 Correspondence-based Learning

Correspondence-based learning aims to find out the correspondences between the input images and the CAD model of the known target object. For RGB images, taken from various angles, the correspondence is determined between the two-dimensional pixels of the images and the three-dimensional points on the CAD model of a known object [13]. In contrast, for input depth images, the correspondence is between 3D points on the point cloud and a partial or complete 3D model. Such correspondences are called descriptors. The correspondence-based learning is described in Figure 5 below.



(a) 2D-3D correspondence



(b) 3D-3D correspondence

Figure 5. *Correspondence-based learning methods [12]*

Some typical 2D descriptors, such as SIFT [18], SURF [19], FAST [20], and ORB [21], have been extensively used in various literature to compute 2D feature matching. Later, perspective-n-point techniques are used to compute the pose of the object. Since this learning approach is applicable for objects with rich texture

and geometrical details to identify local features, it becomes susceptible to lighting conditions, cluttered arrangements, and occlusions [13].

To provide robustness against textures, 3D descriptors such as CVFH [22] and SHOT [23], used 3D correspondences between the partial and full point cloud of the object to recover the object pose. Such methods used least-square instead of perspective-n-point to retrieve the object pose. Nevertheless, sensitivity to detailed object geometry was still an issue with these techniques. [13]

Recently, several other studies have been conducted based on deep learning methods. Some of the methods [24] [25] are based on finding discriminative feature points and comparing them with representative convolutional neural network features. These methods can address occlusions and texture-less objects.

2.5.2 Template-based Learning

Template-based learning methods are used to estimate the object pose-estimation by recovering an identical template from the templates with predefined ground truth poses. For 2D templates circumstances, 2D images are retrieved from the seen 3D models and this problem is more like an image retrieval task. These methods are appropriate for texture-less objects in an occluded and lightly cluttered environment, which is not dealt with by correspondence-based methods [13].

Several methods suggest utilizing point cloud from a 3D model, without projecting 2D images from the 3D models. This is done by comparing the partial point cloud from a target object with the complete point clouds of the known models and retrieve the best matching template for determining the object pose. Nonetheless, this method tends to be tedious.

There has been a lot of work done in the case of 2D template-based learning by using the machine learning techniques. Hinterstoisser et al. [26] proposed the idea of automatically generating templates, using hemisphere sampling, from 3D models of multiple objects. Their method used image gradients on the 2D images for object pose estimation. This technique was tested on the LIMOD dataset which contained fifteen household objects of different sizes, colors, and shapes.

Another study that was conducted by Hodaň et al. worked on the pose estimation using RGB-D images regressed from numerous texture-less objects in the scene. However, the number of templates was inadequate for deep learning. The functional workflows of template-based learning are shown in the figure below.

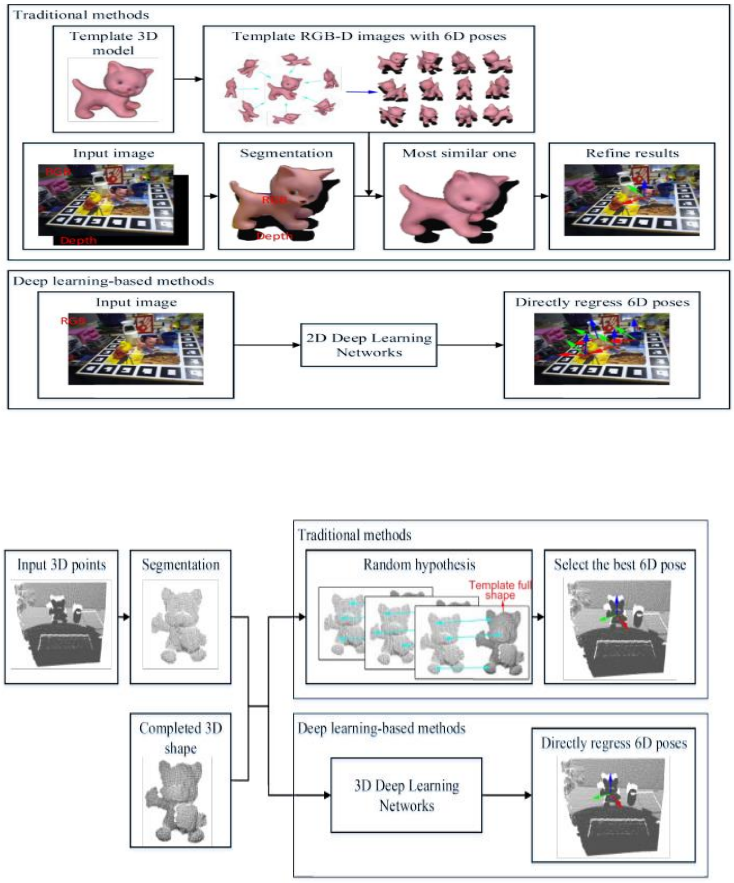


Figure 6. *Template-based learning methods [12]*

PoseCNN [27] computes the 6D pose of an object by predicting its 3D translation and rotation. The 3D translation refers to the distance of the localized object from the camera, and object rotation corresponds to the regressed quaternion representation. This method has proven results on symmetric objects again clutters and occlusion. ConvPoseCNN [28] improves the results of earlier approach by considering region-of-interest (RoI). This method applies pooling feature-extraction in a fully connected convolutional network to extract interesting regions. It also combines translation and rotation into a single regression task with improved accuracy, reduced inference time, and complexity.

2.5.3 Voting-based Learning

Contrary to the previous methods, voting-based learning determines the object pose using the votes from every pixel value or 3D point on the target object. In this regard the voting-based learning assumes two approaches. Indirect voting approaches consider the individual pixel votes for a certain feature point via correspondences such as 2D-3D, whereas direct voting-based techniques

contemplate the votes for a certain ground truth pose. The general layout of both indirect and direct voting-based methods is shown in the figure below.

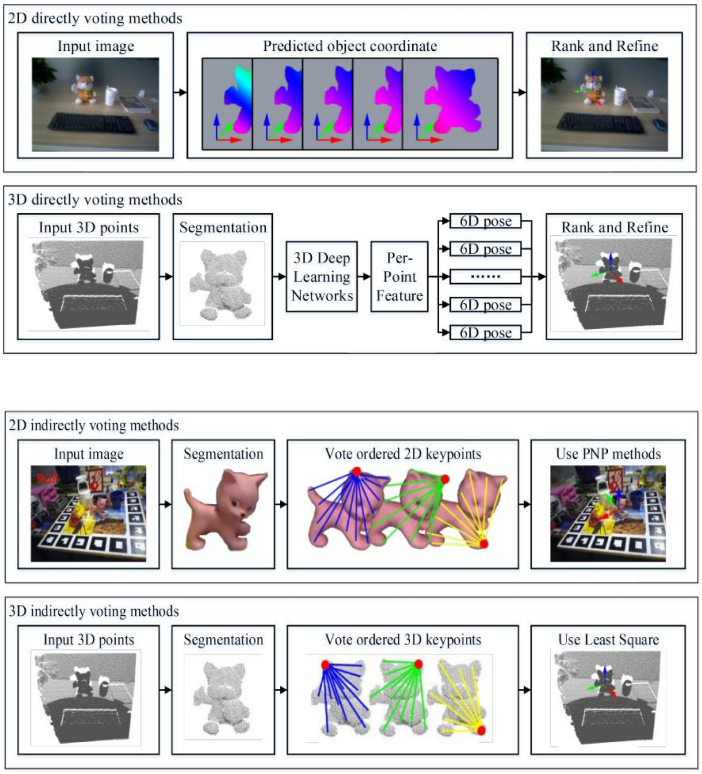


Figure 7. *Voting-based learning methods*

PVNet [29] is an example of an indirect voting-based technique and outperforms some of the earlier methods. This method utilizes pixel-wise voting for detecting 2D keypoint features in the images. Moreover, the network identifies uncertain keypoint locations addressed by correspondence-based methods to enhance robustness against occlusions. A similar network, PVN3D [30], was developed later to deal with 3D key points which has been discussed in the next chapter.

2.6 Model-free Learning

Model-free methods differ from the above-mentioned methods as these methods are normally suitable for novel objects, without having any previous information about the object model instances. Consequently, the pose estimation step is not needed in this case. Also, object placement is ignored, and the object grasped is unfamiliar.

Most of these methods utilize object geometry, retrieved from visual sensors, to perform grasp manipulation. The model is trained with perceptual sensory data of the object in an end-to-end manner and evaluation of grasps is carried out using grasp metrics. Based on the differing approaches, modeling-based learning is further categorized into discriminative and generative approaches [14].

Discriminative approaches involve extensive grasp sampling around the target object. In addition, the sampled grasp candidates are evaluated and ranked using a neural network. [14] Despite high runtime, these methods are advantageous due to multiple grasping capabilities. Levine et al. utilized this approach by implementing hand-eye correspondence for grasping with input RGB images. They carried out this experiment with fourteen robots and gathered around 0.8 million sampled grasps over two months. However, for any changing environmental setup, the data collection and training need to be done again.

Robotic grasp candidates can be retrieved directly when using a generative approach, analogous to an object detection task. In this method, oriented rectangles [Section 2.5] are detected in the RGB images, which explicitly computes the grasp candidates for the robot gripper. Redmon et al. proposed the concept of a single grasp that can estimate an oriented rectangle and classification in an input 2D image. Moreover, they also proposed the MultiGrasp approach for the detection of multiple grasps for the same object from different angles. [14]

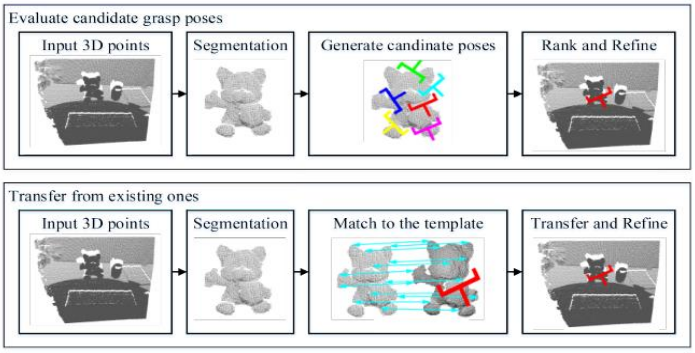
2.7 Point Cloud-based Approaches

Since point clouds store detailed and rich object geometrical representations of 3D models, their application in object detection with deep learning methods has become inevitable during the last few years. Widely available depth sensors, such

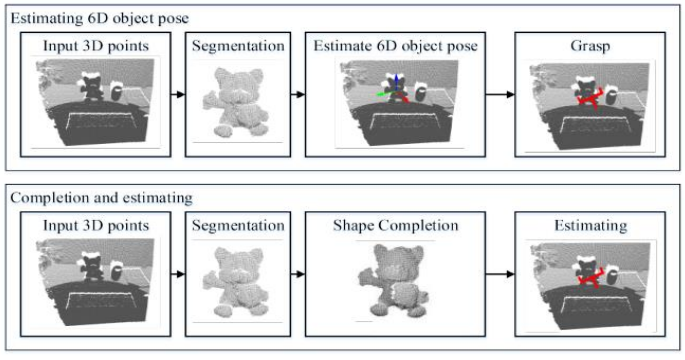
as Kinect, Apple 3D, and RealSense, can easily capture RGB-D images from objects of interest [13]. RGB-D images are RGB images with corresponding depth information. The robotic grasping system deploys depth sensors to project point clouds from depth images for 6DoF pose estimation, grasp detection, and grasp manipulation.

As discussed earlier, 2D image-based techniques tend to be lossy in terms of feature learning. With 3D key-point learning ability, point clouds eliminate losses, as well as handle texture-less objects. However, some of the challenges faced by point-cloud-based methods are the lack of sufficient datasets and the need for high computational requirements.

Point cloud-based 6DoF grasp manipulation can be extended to approaches considering a partial point cloud or complete shape as shown in Figure 8.



(a) Complete shape-based grasp estimation



(b) Partial point-cloud based grasp estimation

Figure 8. *Workflow diagram of point cloud-based grasp estimation [13]*

Further, the partial point cloud is based on two approaches: One approach is evaluating grasps qualities from the candidate grasps database and another is retrieving grasp from current grasps. In the case of complete shape, grasps are predefined for known objects and the problem is analogous to object pose estimation. [13] The major advantage of using point cloud in 6D pose estimation is their improved performance in adapting unseen objects, due to the rich object geometrical features in point clouds.

Point cloud-based approaches can be classified into point cloud-based feature extraction, pose estimation, and grasp detection steps. Each step has been elaborately discussed in the next sections.

2.7.1 Point Cloud-based Feature Extraction

The 2D image-based techniques can be expanded to 3D space with the additional depth information available in RGB-D images to enhance grasp estimation accuracy [30]. Such methods allow utilizing 3D keypoint features from point clouds. Many methods such as Pointfusion [31], Votenet [32], and Pointnet [33] have achieved better results using 3D keypoints instead of the traditional 2D keypoints. In their paper, Guo et.al presented three different point-cloud-based feature extraction methods for classification tasks applied in several grasp detection methods. Initially, these methods get individual points on the point cloud and subsequently collectively retrieve 3D feature points in the form of 3D shape. In the last stage of the process, these points are given as input in deep learning algorithms for classification tasks. [34]

Based on the type of 3D feature points extracted, these classification methods can be differentiated into four other techniques. Multi-view classification methods take various views of the point cloud, retrieve the multi-view 3D features from them, and combine those features to perform classification. Another technique, called Volumetric-based techniques, extracts 3D features from the point cloud in the form of voxelized 3D grids. This point cloud-based classification techniques, along with two other techniques, are described in Figure 9.

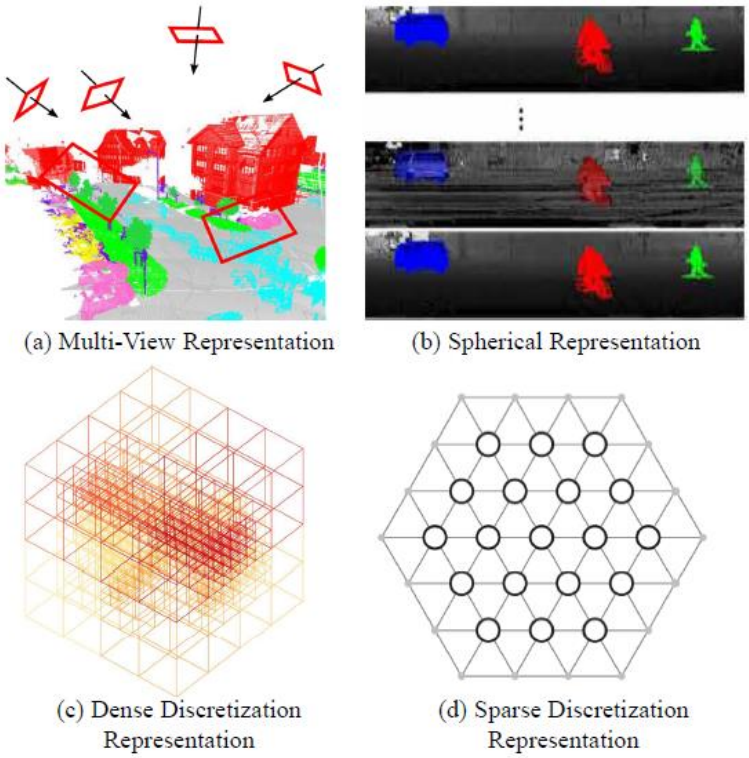


Figure 9. *Point cloud-based feature-extraction techniques [34]*

Using multi-view as the basic approach, Su et al. [35] proposed 3D shape recognition from a set of images taken from various views and fed into a neural network. However, the process of max pooling in the neural network causes loss of information. Similarly, Yang et al. used the relationships between images that were based either on view or region matching and combined them to retrieve the 3D representation. But again, such methods tend to cause loss of information.

Point-based feature extraction approaches have gained significant importance due to better efficiency and these approaches are preferred by researchers. Guo et al. [34] have introduced a few sub-methods under point-based approaches. These methods are graph, convolutional, and point-wise multi-layered perceptron-based approaches as shown in Figure 10.

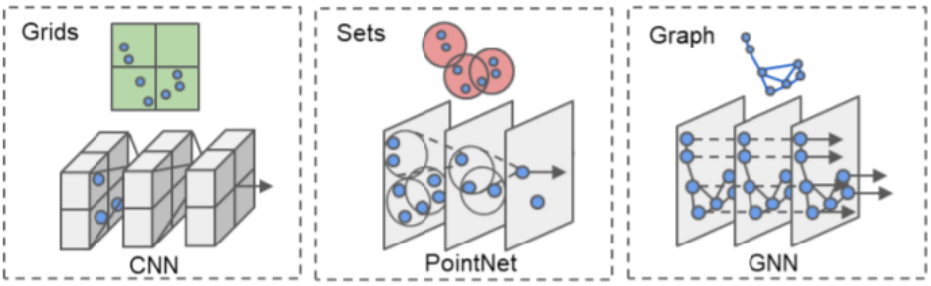


Figure 10. *Extraction of features using point-based approaches [37]*

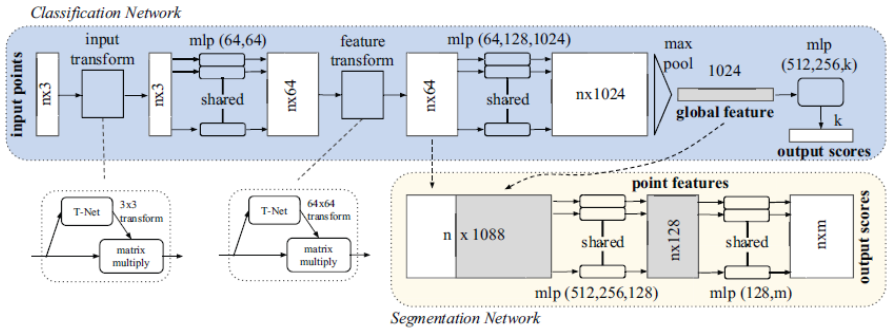
Features in graph-based methods are learned over multi-layered perceptron (MLPs) either in the spatial or spectral domain. In the case of the spatial domain, the graph network is generated first. Each vertex represents a coordinate point or intensities (laser or color). Vertices are connected to their neighboring vertices through edges, and the edges store the object's geometric elements. Convolutional layers operate on spatial neighbors using multi-layer perceptron, while pooling coarsens the graph by gathering data from neighboring points [34]. Two renowned studies [36] [37] have used the above-mentioned graph neural network and achieved encouraging results for object detection tasks using unstructured point clouds.

Unlike 2D image-based feature extraction that uses 2D kernels, 3D kernels are difficult to implement because of the unstructured nature of the point clouds. However, this problem can be resolved by utilizing two different techniques. The first technique is to apply continuous 3D kernels on continuous space and the corresponding nearby vertices are spatially distributed from the center. On the other hand, the second technique considers weights of the nearby vertices at an offset from the center [34].

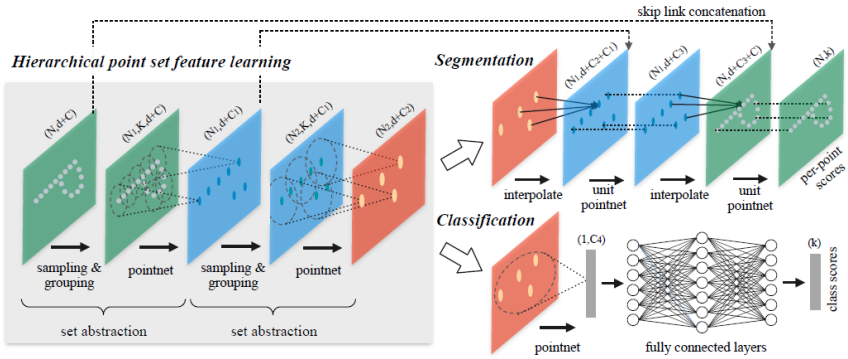
Lastly, pointwise MLP is a point-based technique that feeds individual points as an input with multiple shared MLPs to summate global features for classification and segmentation tasks. Two prominent methodologies used pointwise MLP. PointNet's [38] approach claims to be the first method using unordered point sets from a point cloud, as the earlier methods were based on multi-view and volumetric techniques. Pointnet is a network consisting of shared MLP and max-pooling layers computing global feature extraction for classification tasks. A

significant feature of PointNet is invariance to permutation which means the unordered point sets do not alter the geometric features of the object, thereby allowing direct input of point clouds in deep learning networks.

Since, the point-wise features are learned individually in PointNet, the local Euclidean metrics do not exist between the setpoints. For this reason, the network is unable to generalize to local features. PointNet++ [39] addresses this issue by implementing a hierarchical network. The overall architecture constitutes sampling, grouping, and point net layers. Sampling and grouping layers filter the setpoints and group the overlapping input setpoints based on Euclidean metrics. These grouped points are fed into PointNet layer to extract feature vectors from the localized regions. Set abstraction refers to the process of sampling, grouping, and PointNet feature extraction in an end-to-end manner. The set abstraction process can be repeated until the whole point set is processed for features retrieval. The general workflows of PointNet and PointNet++ network architectures are illustrated in Figure 11.



(a) PointNet Architecture [38]



(b) PointNet++ Architecture [39]

Figure 11. *PointNet and PointNet++ architectures workflow*

The most recent 6DoF pose estimation method, PVN3D [30], employs PointNet++ for retrieving object geometry information from the point cloud with normalized maps, so it serves as an integral part of the state-of-the-art pose estimation network.

2.7.2 Point Cloud-based Pose Estimation

Point clouds, having richer object geometry representation, can perform efficiently for object pose estimation tasks while implemented in the deep learning environment. In addition, the 6DoF object pose can be retrieved directly from the point cloud without the requirement of any additional procedures which were required in the case of 2D image-based methods, such as depth estimation in RGBD images, etc. Like 2D-based methods, point cloud-based pose estimation comprises three sub-categories: correspondence, template, and voting-based methods.

Correspondences are 3D to 3D in point clouds where the pose of the object is computed by matching the partial point cloud with a complete shape of a previously seen object. 3D descriptors, discussed in section 2.5.1, are generally applied to find 3D-3D correspondences between the target object's partial point cloud and the observed complete point clouds. Then the least square algorithm estimates the 6DoF object pose. There exist similar 3D descriptors such as 3DMatch [40], 3DFeatNet based on deep learning methods which estimate robust pose estimation. 3DMatch detects 6DoF object pose by using a 3D voxelated deep learning framework [13].

In template-based methods, the objective estimates the 6DoF object pose for which the partial point cloud matches up with the full point cloud template, discussed in section 2.5.2. Yang et al. suggested a deep learning global registration method robust to pose and noise variations. However, this method consumes a lot of time. Other notable works using this method are PCR-net, DGR, and G2LNet. [13]

Voting-based methods constitute direct and indirect voting approaches, and these approaches have already been discussed in section 2.5.3. From a deep learning perspective, only a few methods are available that estimate 6DoF pose estimation using voting-based approaches. Some notable works are YOLOff, 6-

PACK, and PVN3D that use indirect voting-based methods, whereas DenseFusion and MoreFusion are direct voting-based methods. [13]

2.7.3 Point Cloud-based Grasp Detection

The point cloud-based grasping methods compute grasp directly on the point cloud without the requirement of object pose estimation step. A partial point cloud is taken as input and viable grasps are estimated. Technically, tons of random candidate grasps are produced, and then the viability of each candidate grasp is assessed. Ultimately, the learning networks detect the graspable parts of the point cloud. Since the graspable parts are detected irrespective of the object knowledge, these methods perform efficiently for novel objects [41].

GraspNet [42] used an efficient point-cloud-based methodology with sub-networks to detect stable grasps. This network estimates successful 6DoF grasps via encoder and decoder sub-networks operated end-to-end. First, an encoder network generates numerous sets of 6DoF grasps (gripper poses) from the target object point cloud in a latent space. An encoder network simply samples the grasps by extracting geometrical features from the point cloud to produce a variety of grasps. The subsequent grasp evaluator network predicts proposed grasps to filter out the successful ones only and back-propagates them into the network. The elimination of unsuccessful grasps helps in the generation of viable grasps. The GraspNet network is illustrated in Figure 12.

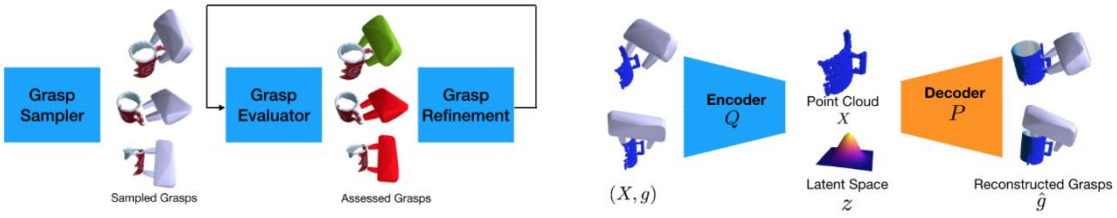


Figure 12. *The GraspNet network [42]*

3. METHODOLOGY

3.1 Overview

The two approaches of grasp detection techniques discussed in sections 2.7 and 2.8 are model-based learning and model-free learning methods. Since the model-based approaches require synthetic data to estimate 6DoF pose estimation of target objects, thereby model-based approaches are considered for the use-case of this thesis. This chapter reflects on the approaches taken to automate the parametric gear modeling, synthetic data generation from custom gear models, and 6-DoF pose estimation of the dataset using a state-of-the-art method, PVN3D [30]. The methodology to generate automated gear models and utilize the models for synthetic dataset generation to train and evaluate a pose-estimation network is illustrated in the Figure 13.

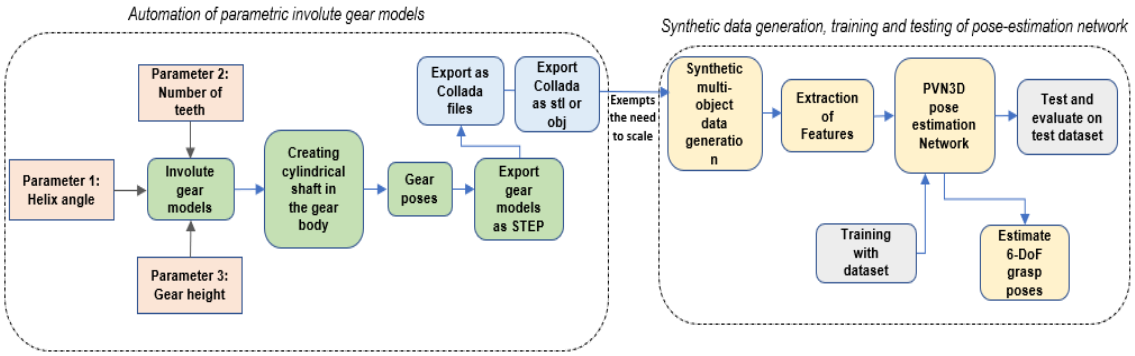


Figure 13. *Workflow of methodology*

As described in Figure 13, the workflow has been divided into two major parts. The first part provides a comprehensive illustration of the method employed to automate the generation of involute gear parts and integrate it with the data generation pipeline. The second part describes the synthetic dataset generation, feature extraction, training the pose estimation network with the training dataset, and testing on the test data set.

3.2 Automation of Parametric Gear Modeling

The core objective of this thesis is to automate the generation of CAD models and training data for robotic tasks such as object detection, object pose estimation, and robot grasping. Generally, unalterable free-form CAD models are acquired for the data generation. However, parametric models can alternatively be used because of their ability to instantaneously generate iterative designs of a part with minimal effort.

Among the FOSS parametric modelers, FreeCAD was chosen mainly for the following reasons:

- Based on python with tons of libraries and workbenches available for 3D modeling
- Provides both scripting and graphical user interface for modeling

The basic gear module was imported from an external workbench, FGGear Workbench [43], that provides numerous gear types such as involute gears, involute rack, cycloid gear, bevel gear, worm gear, and timing/lantern gears. A module can be chosen for customization and a variety of the designs by altering the parameters in an iterative manner. Such an approach automates the design process. In addition to the utilization of intrinsic gear parameters, the gear bodies can be modified and customized parametrically by using python commands or through the graphical user interface in FreeCAD. The methodology for automating involute gear generation is illustrated in Figure 13.

The CAD models were generated using python scripts in the macro editor. Following the instructions in the GitHub repository of FGGear Workbench *https://github.com/looooo/freecad.gears*, the workbench was installed and imported into the FreeCAD python script. From the intrinsic gear parameters, the number of gear teeth, gear height, helix angle, and module size were utilized for customization. When looping through the gear parameters, several involute gears were generated. To induce a sufficient complexity to the design, a cylindrical gear shaft was added to the gear body whose size depends on the gear module parameter. To keep the shaft size proportional and smaller than the gear size, the shaft radius was calculated by dividing the gear radius with a factor of 1.2.

Ultimately, the shaft size remains proportional to the changing gear sizes. The automated gears have either flat or upright poses which were defined in the script. Each gear is then exported either as STEP.

The pseudocode of the script for automation of parametric gear modeling has been described in Algorithm 1. This algorithm was generated for involute gears, but it can generalize to other gears with similar structure: such as cycloid gears, bevel gears, and timing gears.

ALGORITHM 1: AUTOMATION OF PARAMETRIC GEAR MODELING

Parameter:

teeth: number of gear teeth (list)
height: height or thickness of gear (list)
helix_angle: helix angle of gear teeth (list)
m: gear module size (float)

Input : Parametric gear module from FGGear Workbench

Output : Parametric involute gear models with flat or upright poses

Function : involuteGear () (Creates parametric gear with cylindrical shaft through the gear body)

```
1  for h in height do:
2      for t in teeth do:
3          for h in helix_angle do:
4              Function call → involuteGear () → Generate
               parametric gear with a cylindrical shaft.
5              Rotation of the parametric gear about an axis.
6              Placement of the parametric gear from the origin/axis.
7              Export the gear as STEP.
8          end
9      end
10 end
```

As it can be observed in Algorithm 1, each of the parameters is provided in the form of a python list. A parametric gear model would have either 12 or 30 teeth, and the gear height would be either 20 or 60mm, and so on for each iteration. Similarly, the helix angle defines the gear type such as a spur or helical type. For

spur gear, the teeth angle is 0°. On the other hand, the angle has been set to 20° to make it a helical gear.

Nevertheless, the gear models can be automated by plugging in different parameters as well. This can be merely done by updating the parameters with different values for teeth, gear heights, and helix angles. Eventually, parametric models are dynamic and automate the modeling process. These parametric models are required in the next step for synthetic data generation. For that reason, the models are exported with a suitable format and loaded in the simulation environment for generating synthetic data.

Another approach can be generating varying models for one involute gear type only. As an example, the helix angle can be set 20 degrees and different models of helical gears can be generated for different height and teeth parameters. Similarly, only spur type models can be generated for different height and teeth parameters by keeping the helix angle as 0 degree. Therefore, there are many other combinations possible. These variations can be inducted by making the modification to Algorithm 1 accordingly.

3.3 Integrating FreeCAD with Data Generation Pipeline

The integration of FreeCAD with the data generation pipeline is rather indirect, which means that the CAD models generated in FreeCAD need to be imported to the gazebo environment for a synthetic dataset generation by utilizing the available Kinect_ros depth camera. Script-based modeling in FreeCAD allows exporting the model in only three formats: STEP, Standard-Tessellation-Language (STL), and Boundary-Representation (BRep) formats. On the other hand, there are plenty of options to export models through the graphical user interface.

STEP and BRep files are not supported in the gazebo which forces the usage of STL representation of the model or conversion to another suitable format. To date, only four types of CAD models can be imported into gazebo: Standard-Tessellation-Language (stl), wavefront files (obj), and Collada files (dae). To overcome this barrier, the step files generated from the script are exported as Collada files through the graphical interface. Collada file is a richer representation

of a model with the texture and physics information of the model. Also, to generate point clouds from the CAD models, the STEP files are exported as wavefront files (obj) or STL files. Scaling is an issue when models are imported to the gazebo because of the mismatch between the gazebo and model units. Since the gear dimensions have been taken care of in the modeling phase, scaling is not required when exporting the STEP file to Collada format. This way the CAD model dimensioning helps in deciding a reasonable scale factor for importing them into the gazebo.

3.4 Synthetic Data Generation from CAD Models

To generate synthetic data, the CAD models of the gears are placed in a gazebo simulation environment around the origin to create a gazebo world. The gazebo world contains only the dataset models in a lightly cluttered scene. A Kinect sensor, integrated with a robot operating system (ROS), is also added to the gazebo world. Importantly, the shadow and light variations are turned off for the scene. Since the objects do not contain any color at this point, unique color is assigned to each using the model editor in the gazebo.

The synthetic data generation utilizes a unique data collection technique described as hemisphere sampling [44]. This method utilizes a Kinect sensor, an RGB-D camera, to collect images by moving around the multi-object cluttered dataset in the upper hemisphere. The sensor moves around the dataset in incremental values of the yaw angle, pitch angle, and radius of the hemisphere. During the whole process, the X-axis of the camera continuously points towards the origin of the gazebo world coordinate, keeping the camera pointed to the dataset [45].

A concise description of the process is described below:

- The camera is initially at rest, at 0° yaw angle, and starts moving around the dataset at increments of 10° until it reaches 360° yaw angle.
- For each increment in yaw angle, the pitch angle is incremented by 10°. The pitch angle ranges from 0° to 90°.
- For each 15° increment in the yaw angle and 10° increment in pitch angle, the camera generates samples from different scales while moving around

objects at the gazebo origin. The number of scales depend on the arrangement of the dataset objects around the origin and the desired number of synthetic data samples to be generated.

Figure 14 shows the hemisphere sampling technique used to generate synthetic data from the gear's dataset in the gazebo simulator.

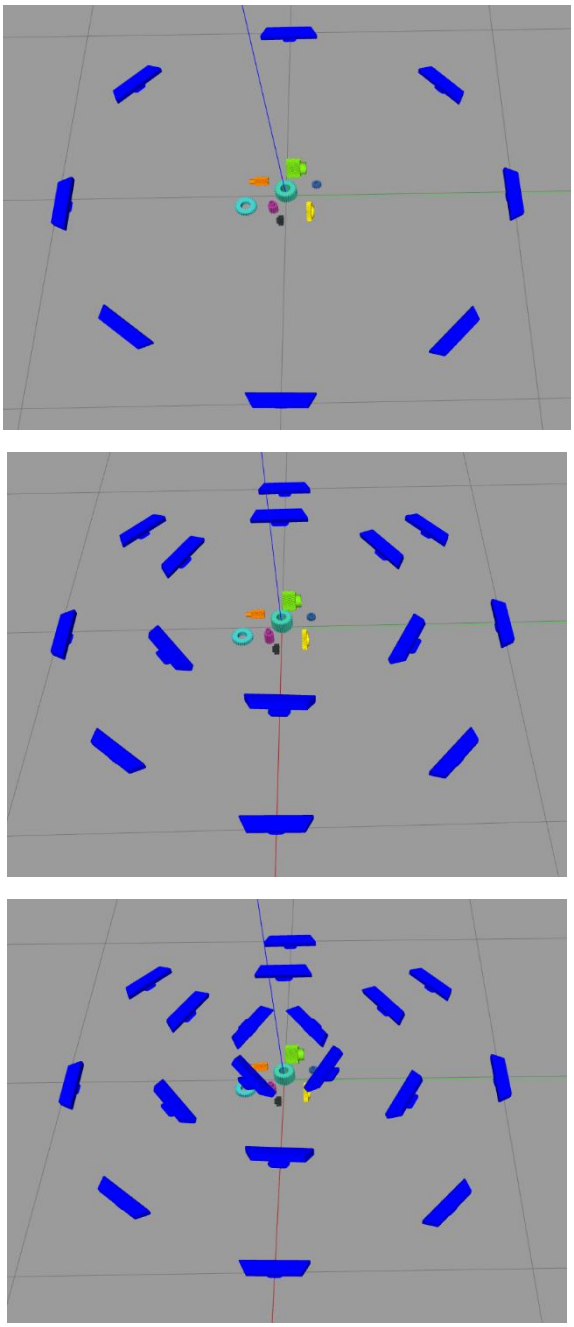


Figure 14. *Synthetic data generation using hemisphere sampling in gazebo simulation*

3.5 6-DoF Pose Estimation for Multi-Class Objects

3.5.1 6-DoF Pose Estimation

In this thesis study, the 6-DoF object pose for the custom gear dataset utilizes a recent model-based method known as PVN3D [30]. The network operates on object point cloud and uses Hough voting for keypoints detection to estimate the object pose. The 6-DoF object pose is characterized by its 3D translation and 3D rotation in the world coordinate frame and the purpose of the 6DoF pose estimation network is to transform the 6DoF object pose from world to camera coordinate.

3.5.2 PVN3D Network Architecture

For the computation of multi-class 6-DoF object pose estimation, an open free source deep learning network, PVN3D [30], has been implemented by following the network's official GitHub repository: https://github.com/ethnhe/PVN3D. This deep learning pose estimation network is based on dense correspondence methods that use depth information to obtain 3D keypoints from target objects and ultimately estimate 6-DoF poses. Figure 15 illustrates the different sub-blocks of the PVN3D network.

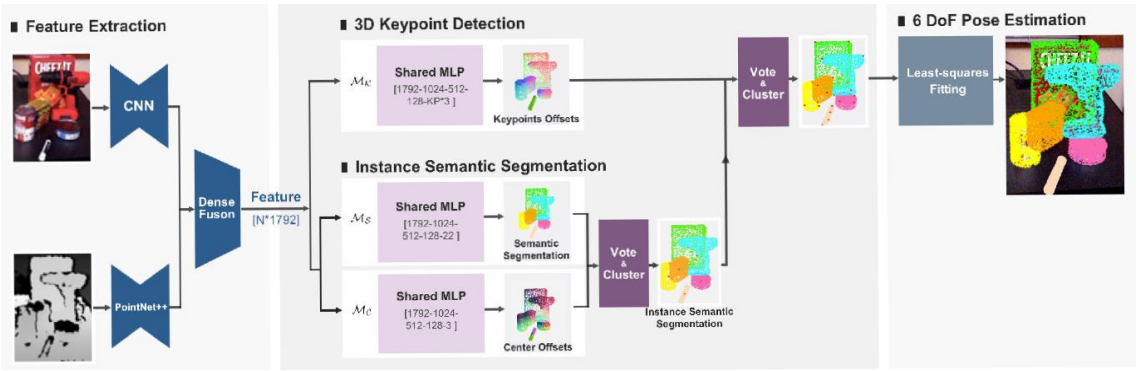


Figure 15. *PVN3D functional diagram [30].*

As can be seen in Figure 15, the cascaded PVN3D network has four functional modules which have been discussed briefly next.

i- Feature Extraction Module

Given an RGB image, this module applies the CNN-based feature extraction method, PSPNet [46], to extract object features. This method performs scene parsing which is based on semantic segmentation. In parallel, PointNet++ [39] operates on the point cloud generated from an RGBD image to retrieve geometric features of the object. The individual points are fused together by DenseFusion [47] to retrieve combined features for all individual points.

ii- 3D-keypoints Detection Module

The task of this module is to utilize the features extracted in the previous module for the detection of 3D keypoints on each target object. The module first estimates the visible per-point offset to the keypoints on the target object within the Euclidean space. Then the keypoints along with the estimated offsets vote for the candidate keypoints.

iii- Instance Semantic-Segmentation Module

This module constitutes two shared multi-layered perceptron (MLPs) layers to perform semantic segmentation on the multi-objects dataset. The first layer performs semantic segmentation by predicting object class labels whereas the second MLP layer utilizes a center voting network to identify object instances in the dataset.

iv- 6-DoF Pose Estimation using Least-Squares Fitting

The least-squares method is implemented to estimate the correspondence and fitting between the network predicted keypoints and the keypoints on the object in world coordinate.

3.5.3 Network Optimization

The goal of the training network is to train the MLPs in the cascaded network modules while optimizing losses incurred at each stage. The network training initiates with the feature extraction module generating combined features from appearance and object geometry fed into the three parallel modules, each having a shared MLP layer. Eventually, the last module estimates the 6-DoF pose of the target objects using a least-squares algorithm. Therefore, this is a multi-tasking learning network trying to optimize loss function at each stage. In addition, to train DNN models, computational requirements are required such as a GPU-enabled processors to run CUDA applications.

The 3D keypoints detection module $\mathcal{M}_{\mathcal{K}}$ takes an input of seed points $\{p_i\}_{i=1}^{N}$ and keypoints $\{kp_j\}_{j=1}^{M}$ from the same object instance I and estimates the ground truth translation offset $\{of_i^j\}_{j=1}^{M}$ between them. The 3D keypoints detection module optimizes the loss function shown in the equation as follow in equation 2

$$L_{keypoints} = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} \| of_i^j - of_i^{j*} \| \mathbb{I}(p_i \in I) \quad (1)$$

Here, N and M are the total seed points and keypoints respectively which are selected from the same object instance I. Note that $\mathbb{I}$ is used as an instant indicator function which is 1 in case, the point p_i is from the same instance I, or 0 in the other case. Interestingly, learning the predicted offsets to keypoints obtains the information related to the object size which helps the network to differentiate between similar objects with different sizes.

The task of the semantic-segmentation module $\mathcal{M}_{\mathcal{S}}$ is to estimate the per-object class labels by utilizing a shared MLP layer. The module is supervised by a loss function shown in equation 3.

$$L_{semantic} = -\alpha(1-q_i)^{\gamma} log(q_i) \quad (2)$$

$$where \quad q_i = c_i \,.\, l_i$$

In equation 3 above, α denotes the α – balancing parameter and γ represents the focus parameter. For the i_{th} point, q_i is the dot product of predicted

$$L_{center} = \frac{1}{N}\sum_{i=1}^{N} \|\nabla x_i - \nabla x_i^*\| I(p_i \in I) \quad (3)$$

confidence c_i and one-hot representation of the class label l_i. The value of the vector q_i is either 0 or 1.

Center-offset module $\mathcal{M}_c$, another shared MLP layer-based module, identifies different instances of the objects by voting for centers of the target objects. Similar

to the keypoint detection module, this module estimates keypoints offset by calculating the distance between input seed points and the object center. The optimization loss function is given by equation 4.

3.5.4 Network Training

The three pvn3d network modules, discussed in the previous section, are supervised in a cascaded manner together to construct a multi-tasked training pipeline for the pose estimation network. The optimization functions from equations (2)(3)(4) along with their corresponding weights can be combined to optimize the loss in multi-tasks, as shown in equation 5.

$$L_{multi-task} = \omega_1 L_{keypoints} + \omega_2 L_{semantic} + \omega_3 L_{center} \quad (4)$$

Here, ω_1, ω_2, ω_3 represent the weights for the losses in the corresponding modules.

During the data collection stage, 2026 synthetic data samples were generated from the custom dataset in the simulation environment. The data samples were generated simulation only. The duplicate images were removed to avoid overfitting. For training the PVN3D network, the dataset was split into 80 and 20 percent respectively for training and test validation datasets. Each synthetic image is 640×480 pixels in size. As stated in the official article of PVN3D it has been recommended to sample 12288 feature points from the point cloud of dataset objects. In case, the feature points are insufficient, the edges of the point cloud are wrapped to the extent where the optimum points are generated.

The training epoch size was set to 25, mini-batch size to 20 to meet the network training criteria. For evaluation, the parameters were kept same. The computational resources were accessed from the CSC clustering network which provides GPU-enabled supercomputer nodes. As recommended, 4 Nvidia GPUs were utilized for the training. The training process has been discussed elaborately in the discussion chapter.

3.6 Least-Squares-Fitting for Pose Estimation

The last module in the network is the pose estimation module which computes the 6-DoFobject by computing the Rotation (R) and translation (t) pose parameters with the help of the least square fitting algorithm. This algorithm establishes the relationship between the detected 3D keypoints in the images and corresponding points on the object to extract the pose parameters. The optimization function estimates R and t by minimizing the loss function shown in equation 6,

$$L_{least-squares} = \sum_{i=1}^{M} \left\| kp_j - \left(R.kp'_j + t \right) \right\|^2 \quad (5)$$

Where M is the number of selected keypoints on the object [30].

4. RESULTS

4.1 Gear Dataset Generation

Algorithm 1 in section 3.2 provides a script-based approach to automate the modeling process. It generates eight different models of involute gears by iterating through different combinations of parameters. The module size parameter has been kept constant to avoid scaling issues. FreeCAD provides a function called Placement which can store the position and rotation of FreeCAD objects around any axis. While looping through the parameters, the generated gear models are rotated with an increment of 90° around the X-axis to produce flat or upright poses. The resulting CAD models generated for the parametric model dataset with different parameters and poses are shown in Figure 16.

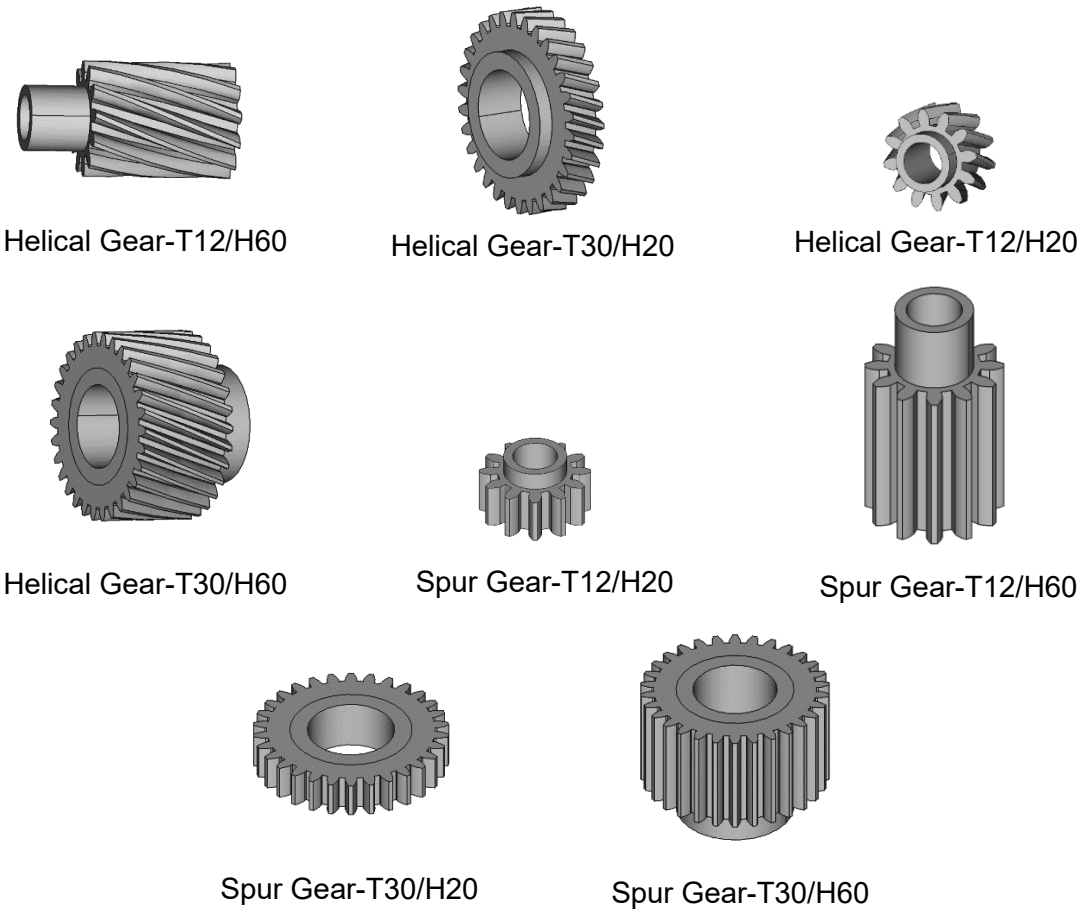


Figure 16. *Automation of parametric involute gear CAD models dataset*

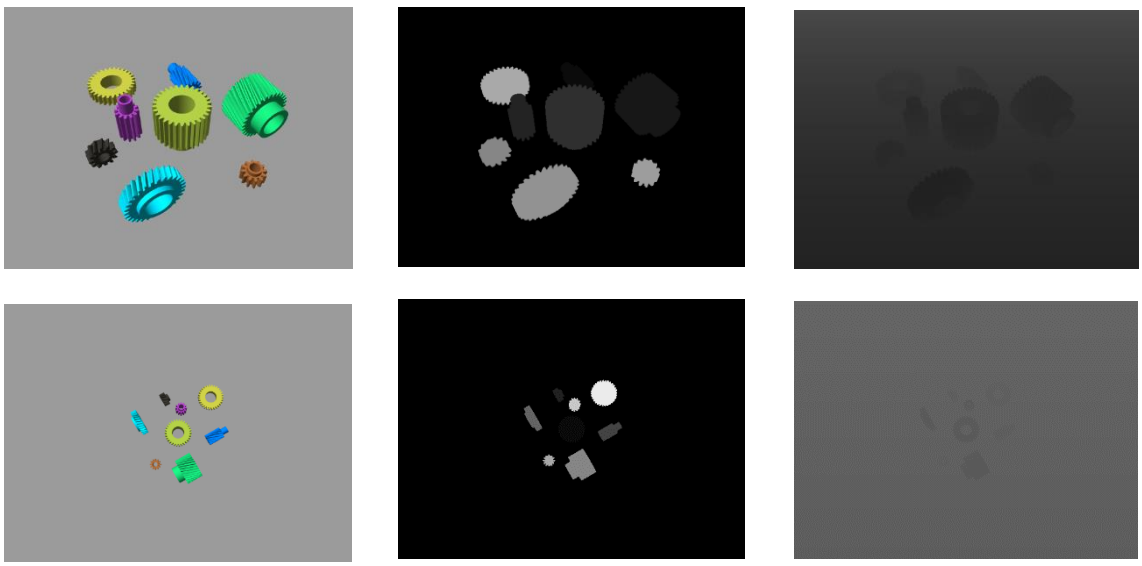
It can be observed in Figure 16 that four of the involute gears are helical type whereas the other four are spur type. T stands for the number of teeth and H stands for gear height. For example, Spur Gear-T12/H20 means Spur Gear having 12 teeth and 20mm height. Additionally, the gears have upright or flat poses.

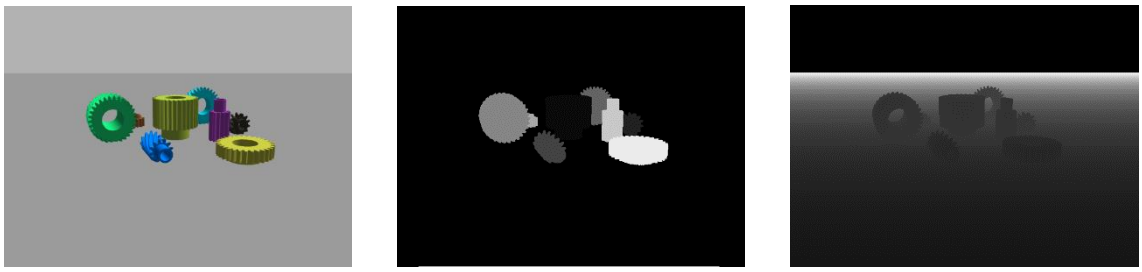
4.2 Training Dataset Generation Results

A total of 2026 data samples were generated in the hemisphere sampling process. For each data sample, the RGB-D camera records four types of synthetic data:

i. RGB image, published to ROS.
ii. Depth image, published to ROS.
iii. Greyscale binary mask image, with class labels of objects in the dataset.
iv. Meta file, that stores the ground truth object poses with respect to the camera coordinates.

The resulting data samples consisting of RGB, depth, and binary mask images are shown in Figure 17 below.





RGB images Mask images Depth images

Figure 17. *RGB, mask and depth image samples from 3 different viewpoints.*

The Kinect sensor in the gazebo is already integrated with ROS which allows publishing the relative topics of RGB and depth images over ROS. The binary mask images are sampled from the point clouds that are obtained from the CAD model of the objects. Since the mask images are grayscale, the intensity of the grayscale values correspond to how far the object is located with respect to the camera coordinate. Therefore, the grayscale value indicates the ground truth label for that object in the RGB image. The range of the grayscale value is 0-255.

The ground truth poses of the objects with respect to camera coordinate are recorded in a meta file with. mat extension. The data format of ground truth poses is python dictionary, with the values of pose coordinates stored as NumPy arrays. Equation 6 defines that the ground truth poses of the objects are calculated by computing the transformation from object coordinate frame to camera coordinate frame [48]. The transformations from camera to world and object to the world is obtained in the simulation.

$$T_{object}^{camera} = (T_{camera}^{world})^{-1} \times T_{object}^{world} \quad (6)$$

Here, T_{object}^{camera}, T_{camera}^{world}, $and\ T_{object}^{world}$ represent 4×4 homogenous transformations from object-to-camera, camera-to-world, and object-to-world coordinate frames respectively.

4.3 Parametric Modeling Computation Time

The computation time to generate and render CAD models is an important factor when designing complex CAD models on large-scale for industrial and commercial applications. The modeling tool and computational factors have a significant impact on the speed of the 3D modeling process. For evaluating involute gears modeling in FreeCAD, the speed metrics are the time taken to generate each model and render the STEP or mesh file. The speed is simply calculated by utilizing the python function *datetime.now()* in the gear modeling python script. Table 2 shows the time taken to generate the parametric involute gear models in FreeCAD and export them as STEP files.

The test results of Table 2 were collected using core i7/2.7GHZ processor, 2 cores and 16 Giga bytes RAM in Ubuntu 18.04. The FreeCAD and python versions were 0.19 and 3.6 respectively. The process runs on a single core for simple models but utilizes multiple cores for complex models. So, the processor speed and number of cores can improve the performance for complex 3D modeling in FreeCAD. The underlying technology of FreeCAD is Open-CASCADE (OCCT) [49] kernel and currently it does not support GPU due to the limitation of the OCCT APIs for GPU computations.

Table 2. *Parametric CAD Models computation time*

Parametric CAD models	Computation Time (in seconds)		
	3D Modeling	**STEP export**	**Total time**
Gear_Helical_12Teeth_20mm	0.28	0.05	0.33
Gear_Helical_12Teeth_60mm	0.37	0.06	0.43
Gear_Helical_30Teeth_20mm	0.44	0.22	0.66
Gear_Helical_30Teeth_60mm	0.62	0.21	0.83
Gear_Spur_12Teeth_20mm	0.25	0.05	0.30
Gear_Spur_12Teeth_60mm	0.77	0.06	0.83
Gear_Spur_30Teeth_20mm	0.44	0.20	0.64
Gear_Spur_30Teeth_60mm	0.92	0.23	1.15
Total elapsed time	**4.09**	**1.08**	**5.17**

In Table 2, it can be observed that the computation time increases with the size and complexity of the models. The larger the gear height and number of teeth, more the computation time required for model generation and rendering. In this case, for the spur gear with 30 teeth and 60mm height, the total computation time is 1.15 seconds.

4.4 Pose Estimation Evaluation

The synthetic dataset was divided into 80 percent training and 20 percent test datasets. The pose estimation network is first trained with the training data and then evaluated for pose estimation accuracies on the unseen test dataset. ADD and ADD-S are the two pose-estimation evaluation metrics used profoundly by researchers, therefore it is implemented for the evaluation purpose in the network. The evaluation metrics are briefly illustrated next.

- **Average Distance of Model Points (ADD)**

ADD [13] [26] generally computes and evaluates the pose estimation accuracies for non-symmetric objects only. It calculates the average distance between a pair of corresponding 3D points on the transformed and the ground truth models. The metric is represented in equation 7 below.

$$ADD = \frac{1}{m}\sum_{x \in M} \left\| (\theta_x + d) - (\tilde{\theta}_x + \tilde{d}) \right\| \quad (7)$$

Here m denotes the points computed from the total number of points on the model represented by *M*. Mathematically, the ground truth reference model is represented by *M* having 3D rotation θ and translation d. The projected rotation and translation are denoted by $\tilde{\theta}$ and $\tilde{d}$ respectively. [13]

- **Average Distance of Model Points (ADD)**

For correct pose estimation, the average distance between the corresponding points should be less than a preset threshold, generally a percentage of the model diameter. However, for symmetric models ADD does not fit well because of repeating points on the models. For such objects another method is used called ADD-S, which calculates the distances among the pair of points but considers only the minimum distance among them, neglecting all other points. ADD-S is represented in equation 8 [13].

$$ADD - S = \frac{1}{m}\sum_{x_1 \in M} min_{x_2 \in M} \left\| (\theta_x + d) - (\tilde{\theta}_x + \tilde{d}) \right\| \quad (8)$$

A more general form of metric is ADD(-S) which explicitly applies either ADD or ADD-S, depending on the object symmetry. The network was trained on 8 and 16 keypoints. The batch size was adjusted to 20 and Table 3 shows the results of the above-mentioned pose estimation metrics on the test dataset of custom gears models.

The network was trained for 8-keypoints and 16-keypoints. For each setting of keypoints the number of epochs were set as 25 with batch sizes of 20.

Table 3. *6-DoF pose estimation accuracies for custom gear test dataset*

PARAMETRIC CAD MODELS	ADD		ADDS		ADD(-S)	
	8-kps	16-kps	8-kps	16-kps	8-kps	16-kps
Gear_Helical_12Teeth_20mm	90.58	89.97	90.58	89.97	90.58	89.97
Gear_Helical_12Teeth_60mm	89.67	88.28	89.67	88.28	89.67	88.28
Gear_Helical_30Teeth_20mm	93.90	94.39	93.90	94.39	93.90	94.39
Gear_Helical_30Teeth_60mm	92.78	92.80	92.78	92.80	92.78	92.80
Gear_Spur_12Teeth_20mm	89.26	88.66	89.26	88.66	89.26	88.66
Gear_Spur_12Teeth_60mm	93.57	93.92	93.57	93.92	93.57	93.92
Gear_Spur_30Teeth_20mm	94.61	94.28	94.61	94.28	94.61	94.28
Gear_Spur_30Teeth_60mm	95.85	94.62	95.85	94.62	95.85	94.62
Average Accuracies	91.27	92.31	91.27	92.31	91.27	92.31

5. DISCUSSION

5.1 Parametric Modeling Automation

The core purpose of implementing parametric modeling in this study was to automate the customization of an industrial part through parameters. Such technique allows to recursively generate several models for a part which can be used to generate synthetic data for deep learning methods. The FreeCAD modeler was employed to generate parametric models for an involute gear part iteratively through parametrization and 8 different involute gear models were generated. The key design features: such as pressure angle, backlash, and undercutting were kept intact, and other parameters were varied to produce varying models for the same gear design. The same models can be generated using a conventional approach which would require each model to be modeled exclusively due to varying features. Therefore, this conventional approach comparatively requires a lot of time and effort to generate the whole dataset.

The parametric software was installed in Ubuntu 18.04 through the terminal. This installation also installs the necessary python packages for some fundamental inbuilt FreeCAD workbenches, but external workbenches need to be installed separately. The package for FC Gears workbench was also installed which is used as the foundation for gear modeling, therefore it eliminated the necessity to model the gears from scratch. The scripting interface is provided as macro editor and the codes are saved as macro files with FCMacro extension. These are regular python codes saved under macro files. The models are exported as a STEP file and can be rendered and modified in any modeler which supports the STEP file format.

5.2 Synthetic Data Generation

Training data is a prerequisite for deep learning methods which can be generated manually in a real-world environment from real objects and sensors. On the other hand, synthetic data can be generated in a simulated environment by viewpoint sampling of the target objects and this approach has the capability to generate thousands of dataset samples. Comparatively, the later approach tends to be

automatic, easier, quicker, and efficient. Regardless of the approach intended, any alteration in CAD models via parametric modeling requires re-sampling and regeneration of the whole training dataset.

A list of dependencies for synthetic data generation algorithm were installed in a conda environment to create a project specific environment. The data generation procedure requires a computer with good computational and graphical capability. Nevertheless, the process does not necessitate the deployment of a GPU.

The hemisphere sampling method generated more than 2100 samples of RGB, Depth, and mask images in about 2 hours. It also generated the same number of meta files that contain the ground truth poses of the target objects. This resulted in generation of more than 8000 files to be processed by the pose estimation network. The data generation time increases proportionally with the desired number of samples. The number of samples generated depends on the calibration of viewpoint sampling parameters, however, this must be done carefully to avoid replication of viewpoints. Generally, network overfitting issues may occur due to the duplicate images. Prior to network training, preprocessing of the synthetic data is also required. Some of the preprocessing tasks are elimination of duplicate images, point cloud generation, keypoints generation, surface normal computation. The preprocessing must be done before providing the dataset to the training network.

5.3 Network Training in CSC

PVN3D pose estimation method requires high computation power and GPU-enabled processing unit for training the deep neural network. This scenario dictates the utilization of cluster to get access to multiple core processors for fast and efficient computations. In addition, multiple GPUs can be run in parallel to meet the training requirements.

CSC is a Finnish company owned by Finnish government and universities. It provides data management and scientific solutions for the educational institutes for research purposes. Puhti and Mahti are the two supercomputers available in CSC cluster. Both have numerous CPU nodes and provide latest Nvidia GPUs for machine learning tasks. Some of the significant features of CSC cluster is listed below:

- The connection to CSC cluster can be established virtually through the terminal in Linux, Mac and PowerShell windows. An alternative way to connect in windows is via PuTTy.
- The user needs to request for resources specific to project requirements. These resources include CPU cores, GPUs, and memory allocations. Both Puhti and Mahti provide several processing cores, GPUs, and temporary memories for computations. For this project, the access to Puhti computer was provided by Tampere university.
- CSC also provides storage areas for project files. The storage areas are divided into disks. These disks are named as home, projappl and scratch. The PVN3D project files were transferred from local computer to projappl disk using bash commands.
- The CSC computing resources are accessed through batch job systems. These batch jobs can be submitted using a batch job scripts or via an interactive session. The job requests are put into queue and resources are allocated upon availability.
- A conda environment, similar to the one for data generation, was created in Puhti with additional packages required for deep learning tasks. These additional packages are PointNet++ and point cloud libraries. PointNet++ is required for CUDA related tasks. CUDA and GCC compilers are already available in CSC and needed to be loaded for every session.

Some other considerations need to be made before initiating the training. These considerations include number of keypoints to train the network on, mini-batch size, and epoch size. The training time depends on the training data size. It took approximately 2 hours to train the network on 2026 data samples.

6. CONCLUSION

6.1 Achieving Research Objectives

To accomplish the first objective, a comprehensive literature review was conducted to analyze and compare modern FOSS parametric modelers. This analysis assisted in choosing FreeCAD as the 3D modeling tool for generation and automation of parametric gear modeling. The most important factors considered for this choice were the availability of python-based scripting and graphical modeling interface in FreeCAD. Moreover, literature studies conducted for pose-estimation methods identified that voting based approaches yield better results for pose-estimation. Therefore, PVN3D method was implemented to estimate the poses of gear models.

The second objective was achieved by implementing FreeCAD script to automate the modeling of involute gear models. The customized involute gear models were generated autonomously through python scripting in FreeCAD by varying different parameters. A total of 8 involute gears with varying parameters were generated in the process which fulfilled the object of parts customization.

The third objective was fulfilled by exporting the parametric models as STEP and mesh files from the script. Due to lack of support of gazebo simulator for STEP files, the models had to be exported as Collada files and loaded into gazebo simulator for synthetic data generation. The reason for exporting STEP as Collada was to resolve the scaling issue which occurs due to the mismatch of modeler and gazebo dimensions. The data generation pipeline was implemented to generate over 2000 data samples. Although, replicating the synthetic data generation pipeline for custom dataset is complex and time-consuming, nevertheless, it provides an efficient method to generate the data as compared to data generation from real objects which is a laborious task.

The final objective was to evaluate the 6-DoF pose-estimation accuracies of the 8 gear models using state-of-the-art PVN3D method. The accuracies were estimated using ADD, ADD(S) and ADD(-s) evaluation metrics. The network was evaluated for 8 and 16 keypoints and the results in Table 3 validate that the pose

estimation on 8 keypoints performs better than the estimation on higher keypoints. According to the author [30] of the official paper, greater number of keypoints is arduous for the network to learn due to larger output space, therefore 8 keypoints is the optimum selection.

6.2 Delimits and Future Works

A delimit of gazebo simulator is the lack of python interface due to which the python based FreeCAD models are manually loaded into the simulation using graphical interface operations. The python interfacing with gazebo can automate the integration of FreeCAD with gazebo simulator for any such releases in the future. Furthermore, the 6DoF pose estimation tasks can be extended to real objects and real sensors. Also, 6DoF grasp manipulation tasks can be implemented for the custom gears in a simulated or real-time environment.

REFERENCES

[1] Z. Kootbally, "Industrial robot capability models for agile manufacturing," *An International Journal. 10.1108/IR-02-2016-0071,* vol. 43, pp. 481- 494, 2016.

[2] N. H. S. R. W. Halil Erhan, "ViSA: A Parametric Design Modeling Method to Enhance Visual Sensitivity Control and Analysis," *International Journal of Architectural Computing,* vol. 8, no. 4, p. 461–483, 2010.

[3] J. J. Shah, "Designing with parametric CAD: Classification and comparison of construction techniques," in *IFIP Advances in Information and Communication Technology, vol.75, pp. 53-68*, 2001.

[4] R. Aish and R. Woodbury, "Multi-level Interaction in Parametric Design," in *Smart Graphics, vol. 3638, pp. 151-162*, 2005.

[5] M. J. Pratt, "Introduction to ISO 10303—the STEP Standard for Product Data Exchange," *Journal of Computing and Information Science in Engineering, https://doi.org/10.1115/1.1354995,* vol. 1, no. 1, pp. 102-103, 2001.

[6] F. Machado, N. Malpica and S. A.-A. T. Borromeo, "Parametric CAD modeling for open source scientific hardware: Comparing OpenSCAD and FreeCAD Python scripts," *PloS One, 14(12), e0225795–e0225795. https://doi.org/10.1371/journal.pone.0225795,* 2019.

[7] "OpenSCAD: The programmers Solid 3D CAD Modeller," [Online]. Available: https://openscad.org/. [Accessed 2021 09 22].

[8] "FreeCAD," [Online]. Available: https://www.freecadweb.org/. [Accessed 15 9 2021].

[9] "CadQuery," [Online]. Available: https://github.com/CadQuery/cadquery. [Accessed 22 September 2021].

[10] T. Paviot, "pythonOCC portal," [Online]. Available: https://github.com/tpaviot/pythonocc. [Accessed 22 September 2021].

[11] D. Hrg, "OpenJSCAD," [Online]. Available: https://github.com/jscad/OpenJSCAD.org.

[12] M. Roberts, "Comprehensive BRL-CAD Primitive Database," 2015.

[13] G. Du, K. Wang, S. Lian and K. Zhao, "Vision-based robotic grasping from object localization, object pose estimation to grasp estimation for parallel

grippers: a review," *The Artificial intelligence review, 2021-03,* vol. 54, no. 3, pp. 1677-1734, 2021.

[14] K. Kleeberger, R. Bormann, W. Kraus and M. Huber, "A Survey on Learning-Based Robotic Grasping," vol. 1, pp. 239-249, 2020.

[15] K. He, G. Gkioxari, P. Dollar and R. Girshick, "Mask R-CNN," *2017 IEEE International Conference on Computer Vision (ICCV), 2017-10,* vol. 2017, pp. 2980-2988, 2017.

[16] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE transactions on pattern analysis and machine intelligence, 2017-06-01,* vol. 39, no. 6, pp. 1137-1149, 2017.

[17] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan and S. Belongie, "Feature Pyramid Networks for Object Detection," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). vol. 2017, pp. 936-944*, 2017.

[18] D. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the IEEE International Conference on Computer Vision, 1999, Vol.2, p.1150-1157*, 1999.

[19] H. Bay, T. Tuytelaars and L. Van Gool, "SURF: Speeded Up Robust Features," in *Computer Vision – ECCV 2006, 2006, Vol.3951, p.404-417*, 2006.

[20] E. Rosten and T. Drummond, "Fusing points and lines for high performance tracking," in *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1, 2005, Vol.2, p.1508-1515 Vol. 2*, 2005.

[21] E. Rublee, V. Rabaud, K. Konolige and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *2011 International Conference on Computer Vision, 2011-11, p.2564-2571*, 2011.

[22] A. Aldoma, M. Vincze, N. Blodow, D. Gossow, S. Gedikli, R. B. Rusu and G. Bradski, "CAD-model recognition and 6DOF pose estimation using 3D cues," in *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops), 2011-11, pp.585-592*, 2011.

[23] S. Salti, F. Tombari and L. Di Stefano, "SHOT: Unique signatures of histograms for surface and texture description," *Computer vision and image understanding, 2014-08,* vol. 125, pp. 251-264, 2014.

[24] K. M. Yi, E. Trulls, V. Lepetit and P. Fua, "LIFT: Learned Invariant Feature Transform," *European Conference on Computer Vision,,* p. 467–483, 2016.

[25] P. Truong, S. Apostolopoulos, A. Mosinska, S. Stucky, C. Ciller and S. De Zanet, "GLAMpoints: Greedily Learned Accurate Match points," *In the IEEE International Conference on Computer Vision (ICCV),* pp. 10732-10741, 2019.

[26] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige and N. Navab, "Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes," *Computer Vision – ACCV 2012,* vol. 7724, no. 1, pp. 548-562, 2013.

[27] Y. Xiang, T. Schmidt, V. Narayanan and D. Fox, "PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered," *https://arxiv.org/abs/1711.00199,* 2017.

[28] C. Capellen, M. Schwarz and S. Behnke, "ConvPoseCNN: Dense Convolutional 6D Object Pose Estimation," *https://arxiv.org/abs/1912.07333,* 2019.

[29] S. Peng, Y. Liu, Q. Huang, X. Zhou and H. Bao, "PVNET: Pixel-wise voting network for 6dof pose estimation," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2019-06-01, vol. 2019, pp. 4556–4565*, 2019.

[30] Y. He, W. Sun, H. Huang, J. Liu, H. Fan and J. Sun, "PVN3D: A deep point-wise 3D keypoints voting network for 6DoF pose estimation," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 11629-11638*, 2020.

[31] D. Xu, D. Anguelov and A. Jain, "PointFusion: Deep Sensor Fusion for 3D Bounding Box Estimation," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2018-12-14, pp. 244-253*, 2018.

[32] C. R. Qi, O. Litany, K. He and L. Guibas, "Deep Hough Voting for 3D Object Detection in Point Clouds," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV), 2019-10, vol. 2019, pp. 9276-9285*, 2019.

[33] C. R. Qi, W. Liu, C. Wu, H. Su and L. J. Guibas, "Frustum PointNets for 3D Object Detection from RGB-D Data," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2018-12-14, pp. 918-927*, 2018.

[34] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu and M. Bennamoun, "Deep Learning for 3D Point Clouds: A Survey," in *IEEE transactions on pattern analysis and machine intelligence, 2021-12-01, Vol.43 (12), pp. 4338-4364*, 2020.

[35] H. Su, S. Maji, E. Kalogerakis and E. Learned-Miller, "Multi-view Convolutional Neural Networks for 3D Shape Recognition," in *2015 IEEE*

International Conference on Computer Vision (ICCV), 2015-12, vol. 2015, pp. 945-953, 2015.

[36] Y. Wang, Y. Sun, Z. Liu, S. Sarma, M. Bronstein and J. Solomon, "Dynamic Graph CNN for Learning on Point Clouds," *ACM transactions on graphics, 2019-11-05,* vol. 38, no. 5, pp. 1-12, 2018.

[37] W. Shi and R. Rajkumar, "Point-GNN: Graph neural network for 3D object detection in a point cloud," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 1708-1716*, 2020.

[38] R. Q. Charles, H. Su, M. Kaichun and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017-07, pp. 1708-1716*, 2017.

[39] C. R. Qi, L. Yi, H. Su and L. J. Guibas, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space," in *Advances in Neural Information Processing Systems, 2017, Vol.2017-, p.5100-5109, https://arxiv.org/abs/1706.02413*, 2017.

[40] A. Zeng, S. Song, M. Niessner, M. Fisher, J. Xiao and T. Funkhouser, "3DMatch: Learning Local Geometric Descriptors from RGB-D," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017-07,* vol. 2017, pp. 199-208, 2017.

[41] A. ten Pas, M. Gualtieri, K. Saenko and R. Platt, "Grasp Pose Detection in Point Clouds," *The International journal of robotics research, 2017-12, Vol.36 (13-14),* pp. 1455-1473, 2017.

[42] A. Mousavian, C. Eppner and D. Fox, "6-DOF GraspNet: Variational Grasp Generation for Object Manipulation," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV), 2019-10, Vol.2019-, p.2901-2910*, 2019.

[43] looooo, "FGGear Workbench," [Online]. Available: https://wiki.freecadweb.org/FCGear_Workbench. [Accessed 2021 07 10].

[44] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige and N. Navab, "Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes," in *Computer Vision – ACCV 2012*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 548-562.

[45] K. Samarawickrama, "MSc Thesis-RGB-D Based Deep Learning Methods for Robotic Perception and Grasping," Faculty of Information Technology

and Communication Sciences, Tampere University, 2021. [Online]. Available: https://urn.fi/URN:NBN:fi:tuni-202105185131.

[46] H. Zhao, J. Shi, X. Qi, X. Wang and J. Jia, "Pyramid Scene Parsing Network," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017-07, vol. 2017, pp. 6230-6239*, 2017.

[47] C. Wang, D. Xu, Y. Zhu, R. Martin-Martin, C. Lu, L. Fei-Fei and S. Savarese, "DenseFusion: 6D object pose estimation by iterative dense fusion," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2019-06-01, pp. 3338-3347*, 2019.

[48] M. Tian, L. Pan, M. H. Ang and G. Hee Lee, "Robust 6D Object Pose Estimation by Learning RGB-D Features," in *Proceedings - IEEE International Conference on Robotics and Automation, 2020-05-01, p.6218-6224*.

[49] "Open CASCAD Technology," [Online]. Available: https://dev.opencascade.org/doc/overview/html/index.html. [Accessed 2021 11 26].

[50] A. Eltaweel and Y. SU, "Parametric design and daylighting: A literature review," *Renewable & Sustainable Energy Reviews, 73, 1086–1103,* p. 1087, 2017.