Robot Grasping Point Estimation via Operator Input through the Usage of Neural Networks

handed in
MASTER'S THESIS

B.Sc. Qiaoyue Yang

Human-centered Assistive Robotics
Technical University of Munich

Univ.-Prof. Dr.-Ing. Dongheui Lee

Supervisor:	Matteo Pantano
Start:	03.05.2021
Intermediate Report:	26.10.2021
Delivery:	22.02.2022

TECHNISCHE UNIVERSITÄT MÜNCHEN
Human-centered Assistive Robotics
UNIV.-PROF. DR.-ING. DONGHEUI LEE

February 16, 2022

MASTER'S THESIS

Robot Grasping Point Estimation via Operator Input through the Usage of Neural Networks

Problem description:

Picking random objects in not-known positions is still a complex task for robotic applications in industrial scenarios [1]. As a matter of facts, market ready systems for solving this issue (e.g. Keyence® Robot Vision) can be come expensive. Therefore, if this task need to be implemented in Small Medium Enterprises (SMEs), which represent the 99% of the European businesses [2], it could become difficult for the companies. Fortunately, other solutions which rely on new developments in Artificial Intelligence (AI) and grasping strategies based on object shape detection can be employed. These leverage different sensors and open-source AI algorithms based on Neural Networks (NN) to detect objects and then generate point where gripping can happen [1].

Therefore, in this master thesis you will delve in this topic and try to came up with your solution for bin picking trying to leverage open source methodologies. However, for making the technology more affordable and compelling to SMEs you will integrate new user input sensors (e.g. Tracepen™) and camera technologies (e.g. Basler™ 2D camera) which could reduce the barrier of adoptions of the technology. More specifically, you will use neural networks for identifying parts with object detection and then estimate grasping poses based on the object geometry and the operator input. Your research should prefer robustness and speed so an appropriate neural network architecture should be chosen. Therefore, your objectives can be summarized as follows:

Tasks:

- Literature research in bin picking technologies for identifying approaches with NN
- Choice of a hardware and software architecture for the bin picking framework considering user input through the Tracepen
- Integration of the architecture on a real manufacturing cell
- Test and evaluate against existing technologies using known metrics from the manufacturing domain

Bibliography:

[1] Fujita, M. et al., "What are the important technologies for bin picking? Technology analysis of robots in competitions based on a set of performance metrics", Advanced Robotics, 2019, pp. 1-15, DOI: 10.1080/01691864.2019.1698463.
[2] European Union, "Unleashing the full potential of European SMEs", Publications Office of the European Union, Luxembourg, 2020, ISBN: 978-92-76-16912-3.

Supervisor:	Prof. Dr. Dongheui Lee, M. Eng. Matteo Pantano
Start:	03/05/2021
Intermediate Report:	26/10/2021
Delivery:	22/02/2022

(D. Lee)
Univ.-Professor

Abstract

Universal bin-picking is a necessary skill for robots, but still challenging. In this master thesis, a robot grasping point estimation leveraging neural network and operator input is proposed for industrial application scenario. Initially, the background of the topic is introduced. State of the art robot end effector design, bin-picking approaches, and robot manipulation are then reviewed. Next, methodologies of this thesis are elaborated in details - dataset processing, neural network training, and a robot teaching device TracePen™. Finally, performances of the proposed robot grasping point estimation approach are evaluated. Overall 82.24% success rate of suction point prediction is realised for vacuum gripper.

Contents

Acronyms and Notations

CNN Convolutional Neural Network

CAD Computer-aided Design

FCN Fully Convolutional Neural Network

MPPH Mean Pick per Hour

UR Universal Robot

3D 3 Dimensional

2D 2 Dimensional

CGD Cornell Grasp Dataset

DNN Deep Neural Network

MTBF Mean Time between Failures

MTTR Mean Time to Repair

RGB-D Red Green Blue-Depth

3D/6D 3 Dimension/6 Dimension

ICP Iterative Closet Point

ROBI Reflective Objects in Bins

GQ-CNN Grasp Quality Convolutional Neural Network

GG-CNN Generative Grasping Convolutional Neural Network

FC-GQ-CNN Fully Convolutional Grasp Quality Convolutional Neural Network

ISF Iterative Surface Fitting

ROI Region of Interest

SVD Support Vector Machine

ResNet Residual Network

RViz ROS Visualization

URDF Unified Robot Description Format

ReLU Rectified Linear Units

MSE Mean Square Error

SGD Stochastic Gradient Decent

RMSprop Root Mean Sqaure Propagation

DoF Degree of Freedom

TCP tool center point

ROS Robot Operating System

xacro Extensible Markup Language Macros

Chapter 1

Introduction

1.1 Background

Universal bin picking is still challenging and unsolved, but very practical for factory automation, warehouse automation, household robots and so on [1]. Robots with robust grasping ability help supply parts in manufacturing, pick-and-place in warehouses, and provide service in daily lives [1].

Current bin picking solutions mainly aim at grasping daily life objects, such as mug, bowl, fruit, food packing box, can, which can be applied to home service robots, and warehouse automation. Some methods target specifically home decluttering such as bed-making [2] and textile pickup [3]. The robots used to evaluate the bin picking systems are mostly robot arms/manipulators, but humanoid robots are also employed [4]. Nevertheless, research on robotic bin picking for industrial scenarios is relatively blank and challenging[5].

Normally, the robot manipulator moves to a certain pose given by a human user via a control panel or other user interfaces to pick up an object. However, it only works when the type and pose of the object never change, which is infeasible in the real world. Since 1980s, researchers started analyzing the object properties such as shape, geometries, and dynamics to compute mathematical and physical models [6]. These analytical approaches analyzed force closure and stability of a grasp, modeled a task, and compared grasps performances[6]. However, simple picking and placing by force closure grasps is far not enough for the tasks in the real-world [6]. Besides, modeling a task is difficult, and complex computations are required to find a grasp for the task[6]. The divergence between mathematical models and real objects is also problematic [7]. Furthermore, the solutions are often not robust to new objects and new tasks[6]. Therefore, researching on empirical methods that mimic human behavior is a growing trend for robotic grasp to improve generality of a grasp method

and avoid complex computations[6].

In the last five years, empirical learning-based robot grasping approaches attract much attention, due to the fact that Convolutional Neural Networks (CNNs) provide a promising performance for various applications such as pose estimation, image/video recognition, voice recognition, and natural language processing [8]. Intuitively, humans observe and locate the objects by eyes, and then their hands reach the object to pick up. Therefore, learning-based approaches rely on camera observations of the objects to be grasped, and then processing the observed images by algorithms, e.g. artificial neural network, and finally actuating the robot manipulator to perform the grasping operation. The whole process imitates natural operations of human-beings. The empirical methods can be categorized as model-based and model-free methods [9]. Model-based approaches utilize specific object knowledge, e.g. CAD models, or pose estimation of objects. Model-free approaches directly map observations to grasping pose [9].

Apart from CNN, fully convolutional networks (FCNs) also have deployment in the fields of image segmentation and object detection due to their scaling and pixel-wise discrimination abilities [10]. It was proven that FCN is also beneficial for robot grasping [1]. Moreover, transfer learning is also employed to adopt weights of previous trained models as initialization for training the grasping prediction networks [2], [11].

1.2 Requirements

The Siemens project that this thesis work for has following requirements:

1. The bin picking framework should be validated on a Universal Robot (UR) 10.
2. Using 2D cameras or low-cost 3D cameras.
3. A no coding device - TracePen™ - should be leveraged to gain operator input.
4. Using a modular gripper consisting of a vacuum gripper and a parallel jaw.
5. Open source technology is preferable.

1.3 Thesis Outline

The main contributions of this thesis are:

1. A grasping pose estimation framework leveraging the deep learning method and operator input by the TracePen™.

2. An industrial robotic grasping dataset.

3. A robotic bin-picking system via Robot Operating System (ROS).

The thesis outline are as follows:

1. Chapter 1 introduces the background of robot grasping/bin picking, and explains some requirements for the thesis.

2. Chapter 2 reviews state of the art of main technologies related to the thesis - the end effector of the robot, vision system for robotics, and the robot actuation.

3. Chapter 3 presents methodologies concentrating on learning based approach to predict grasping pose.

4. Chapter 4 describes experiments where the methodologies are implemented.

5. Chapter 5 discusses several valuable points discovered from the thesis work.

6. Chapter 6 summaries the thesis and expect future work.

Chapter 2

State of the Art

This chapter firstly reviews the design of robot end effector used for grasping. Next, two main types of robotic grasping approaches - analytic and learning-based approach - are studied in detail, and then published datasets and typical neural network architectures for learning-based methods are investigated. Lastly, the robot kinematics of actuation is introduced.

2.1 End Effector

The end effector of a robot, also called gripper, has functions similar to human hand - grasping, holding, and placing an object. Humanoid robots are usually mounted with hands with five fingers [4]. For robot manipulators in industrial scenarios, grippers are mainly categorized into suction gripper, vacuum gripper, and parallel jaw (also called two-fingers in some literature), as shown in Figure 2.1(a) [12]. The suction gripper is good at grasping objects with a flat surface, or textile (e.g. cloth) by suctioning air between the gripper pad and the object [12]. Vacuum gripper is the most widely used in industries. Its principle is the same as that of suction gripper, but the vacuum is stronger to generate negative pressure to make the object surface tightly stick to the pad [12]. Parallel jaw is capable of picking stereoscopic objects when contact surface between the gripper and the object cannot be well formed. The grasping point estimation needs to be more accurate with the parallel jaw, because the jaw may cause collision while picking [12].

Fan et al. [13] designed a customized parallel-jaw with curved fingertips, in order to enlarge the contact area between the object and the jaw. The grasp is more stable when picking up curved objects without a large flat surface.

In recent years, multi-gripper in a combination of two or three grippers gains popularity, in order to be more robust to pick objects with various size, shape, and texture. Zeng et al. [14] deployed a multi-functional gripper consisting of a parallel-

(a) Vacuum gripper (left), suction gripper (middle), parallel jaw (right)[12].

(b) Muti-functional Gripper [14].

Figure 2.1: Gripper Types

jaw and a suction gripper with Multi-Affordance estimation, as shown in Figure 2.1. Fujita at al. [12] employed three grippers including one more vacuum gripper with a gripper switch strategy.

2.2 Vision System for Robotic Grasping

Visual sensor is of great significance on robots for perceiving external environments. In the field of robotic grasping, the visual perception data is necessary for the robot to observe the objects to be grasped, and the formats can be RGB, RGB-D, depth data, point cloud, etc. The robotic grasping estimation system then processes the perception data via various algorithms to estimate object pose, or predict robot grasping pose. The algorithms develop from analytic approaches to learning-based approach explained in details in the following.

2.2.1 Analytic Approaches

In the past four decades, analytic approaches were investigated to perform bin-picking tasks. Spenrath et al. [15] determined gripping point by heuristic search. A gripping library was firstly established by experts including geometric models of the gripper and workpieces, and the gripping points. The workpiece models in the format of point cloud were obtained by 3D sensors such as a camera or a laser scanner. The elements in the library constituted a search tree. A heuristic function was designed to calculate local cost of a node in the tree, and estimated cost for exploring to a final node. Raw gear shafts were used to assess the system, and the robot picked up 245 parts successfully. The author also concluded that depth first search reduced both computation time and collision. Nevertheless, this method is only able to pick up known objects that are available in the library. Furthermore, the final search tree world be very large for complex tasks.

Lin et al. [16] proposed a framework to estimate 6D object pose based on RANdom

SAmple Consensus (RANSAC) algorithm and Perspective-n-point (P-n-P) algorithm. The observed single RGB image was matched with 3D meshes from database via keypoints. The author chose the keypoint feature, rather than edge or optical flow that are variant to orientation and scale. Features on the 2D observed image were extracted by Rotated BRIEF (ORB) algorithm, and filtered by RANSAC. The 2D-to-3D correspondences were then computed by the P-n-P algorithm to estimate the object pose. However, the feature extraction is dependent on the types of objects and application scenarios.

2.2.2 Learning based Approaches

Learning-based approaches (also called data-driven or empirical methods) become prevalent nowadays due to the advents of CNN. Model-based empirical methods usually contains three steps: object pose estimation, grasp pose prediction, and motion planning. In contrast, model-free empirical methods skip the object pose estimation step, and directly predict grasp pose from visual perception data [9]. Supervised learning and reinforcement learning are mainly employed to train the network.

Model-based Methods

Fujita et al. [12] deployed a bin-picking framework using a multi-gripper strategy. The multi-gripper combines a two-finger gripper, a suction gripper, and a vacuum gripper. As illustrated in Figure 2.2, the point cloud data and primitive shape models were matched to estimate item shape and pose by a deep neural network (DNN). According to the object sparseness, a gripper switching strategy was applied: the suction and the vacuum gripper should be used if the bin is crowded, due to the fact that the two-finger gripper possibly causes collision. The picking point of the suction and the vacuum gripper was determined to be close to mass center of the fitted primitive shape. The grasp point of the two-finger was predicted by a Fast Graspability Estimation (FGE). The system was tested by two robot manipulators installed on two sliders with RGB-D cameras and force sensors mounted in the robot manipulators (called eye-in-hand or hand-eye).

Model-free Methods

Compared to model-based approaches, model-free methods do not estimate the object pose, but directly predict grasping point according to visual observations. Zeng et al. [14] provided a promising bin picking solution using fully convolutional neural network with multi-view RGB-D images, and won the first place in stow task of the

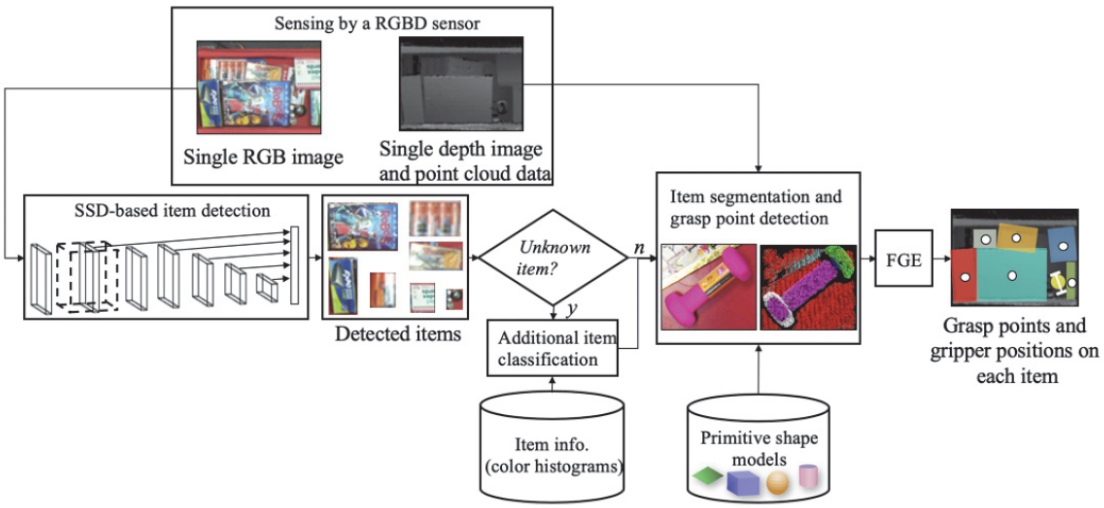


Figure 2.2: Flowchart of the bin-picking system [12]. The item shape and pose are estimated by matching the primitive shape models with the detected item images. Based on the estimated object pose, a Fast Graspability Estimation (FGE) method proposes grasp points.

Amazon Robotics Challenge 2017. The tasks are picking up from a bin, and then recognizing the object to classify.

Four motion primitives as visualized in Figure 2.3 - suction down, suction side, grasp down, flush grasp - were defined to pick an object up by a multi-functional gripper combining a parallel jaw and a suction gripper. Suction and grasp affordance predictions were performed by fully convolutional residual networks (FC-ResNet-101) respectively to rank the probabilities of grasp success for the motion primitives, as interpreted in Figure 2.4. The suction proposal is a 3D position with its surface normal, and a confidence score. The suction contact area with the object should be plane and near the mass center. The grasp proposal is a 3D position and an orientation around the gripper's vertical axis, which is 4 Degree of Freedom (DoF). In addition, the suction and grasp affordances predicted by the FC-ResNets in this method were compared with results of analytic algorithms that compute variance of surface normal for suction, and detect hill-like geometric features for grasping.

After grasping, category recognition was performed by a two-stream convolutional neural network (ConvNet) computing features of observed images and product images. It matched observed images and candidate product images to find semantic image correspondences of the two domains.

The strategy of this method is acting first, then see, which makes object recognition

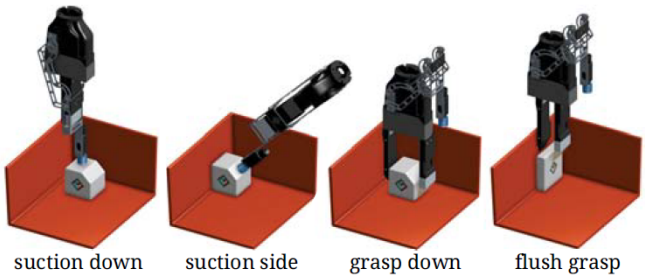


Figure 2.3: Four motion primitives. [14]

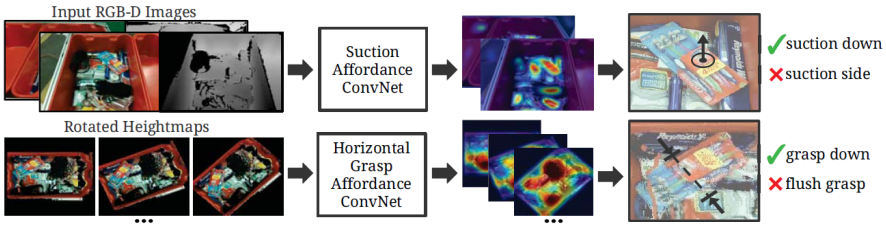
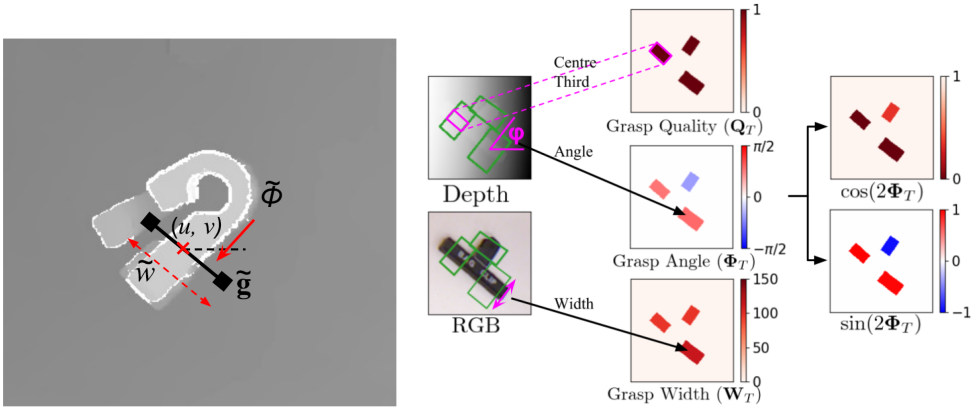


Figure 2.4: Two models were trained respectively for suction and grasping [14]. They directly map from visual perception to grasp proposals.

easier because picking up the object firstly from the cluttered bin. It is beneficial for scenarios, where no specific object is required to be picked up.

Morrison et al. [5] proposed a generative grasp synthesis approach to predict pose and quality of grasps pixel-wisely using a two-fingered gripper. They augmented the Cornell Grasp Dataset (CGD) and Jacquard Dataset by rotating, zooming, and cropping as their training dataset. As illustrated in Figure 2.5, the grasp representation $\tilde{g}$ compromises center pixel position (u, v) of contact points between the two jaws and the object, an orientation angle $\tilde{\Phi}$ around z axis, and the gripper width $\tilde{w}$. Next, four grasp images - grasp quality image, cosine angle image, sine angle image, and grasp width image - are generated based on the inpainted depth images, and then they are fed to the Generative Grasping CNN (GG-CNN) to directly estimate the grasp pose at pixel level.

Fan et al. [13] presented a two-levels framework for robust bin picking with a customized parallel-jaw (as introduced in Section 2.1). The low level is a optimization-based planner with iterative surface fitting (ISF). ISF has two steps: iteratively matching correspondence points on the object surface by nearest neighbor search and outlier/duplication removal, and surface fitting to optimize gripper transformation and finger displacement. The planner searches iteratively the jaw transformation in a local region to minimize surface fitting error, so that the grasp pose fits the object



(a) Grasp Representation[5]. (b) Generated training images[5].

Figure 2.5: Training Data of GG-CNN[5].

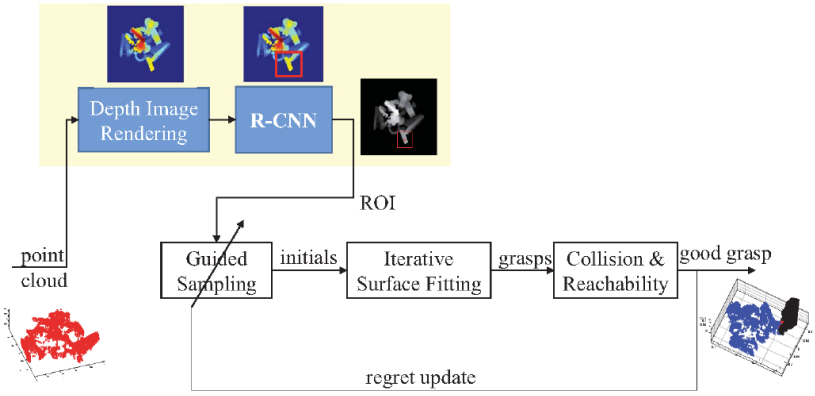


Figure 2.6: RCNN-ISF pipeline. [13]

surface. While the search of the planner is local, the result is significantly dependent on initialization of ISF algorithm. Therefore, a learning-based explorer learns a desired region to initialize the planner by a region-based convolutional neural network (R-CNN). The pipeline of the R-CNN is that a region proposal block proposing potential region of interest (ROI) is fed to a CNN to extract features. Next, regions with the features are classified by Support Vector Machine (SVM), and bounding boxes are refined to acquire the object position. The input to the framework is the point cloud of the object captured by two stereo cameras (IDS Ensenso® N35). The R-CNN was pre-trained by AlexNet and improved by additional objects of 25 different types.

Mahler et al. [17] proposed grasp success probability estimation by Grasp Quality

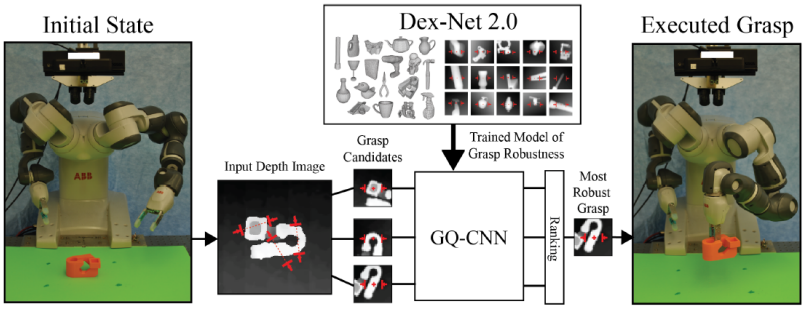


Figure 2.7: GQ-CNN trained on Dex-Net 2.0. [17]

Convolutional Neural Network (GQ-CNN), based on 3D objects meshes and depth images. A parallel jaw was deployed. Initially, hundreds of grasps were generated for each object based on uniformly sampled contact points on the whole object surface and a random direction from friction cone. These generated antipodal grasps were grasp candidates. Rendered depth images (point clouds) were then synthesized from meshes mainly based on uniformly and randomly sampled object shape, pose, and camera pose. Finally, Matching the grasp candidates and the depth images yields grasp images that fed to GQ-CNN to rank the candidates, as illustrated in Figure 2.7. An ABB YUMI robot arm with an eye-to-hand (camera installed overhead) configuration was employed to conduct experiments and evaluate performance.

Datasets

Most of the data used in bin picking solutions are household objects. Almost each solution created its own dataset. There is still no very common dataset in the field of bin picking. Cornell Grasp Dataset (CGD) is a relatively popular manually annotated dataset for robotic grasp detection [9]. Figure 2.8 shows samples of CGD. It contains 885 images, 885 point clouds and about 8019 labelled grasps for 240 different object types from different views [18], such as fruit, beverage can, toothpaste, bowl, mobil phone, ect. The background is a white board.

To the best knowledge of the author, two industrial bin picking datasets were proposed in recent years: Fraunhofer IPA Bin-Picking dataset[19] and ROBI (Reflective Objects in Bins) dataset[20]. The former[19] comprises eight objects from Sileane dataset and two novel industrial parts - a ring screw and a gear shaft, as shown in Figure2.9. The data was generated both in a synthetic way and in real-world. It concludes 520 annotated point clouds and the corresponding depth image in real world, and 206000 synthetic scenes. The synthetic data was generated by importing the CAD models of each object with various positions and random orientations to the bin incrementally, which means starting from one object in the bin, to a

Figure 2.8: Samples of Cornell Grasp Dataset [18].

pre-defined limit. Notably, the bin was cleared and the objects were dropped again after each scene recording. The real world data was captured by an Ensenso® N20-1202-16-BL stereo camera. The number of objects in the bin decreased after each recording starting from a full bin. The 6D poses were annotated by Iterative Closest Point (ICP) algorithm to fit the CAD model to the object in the bin.

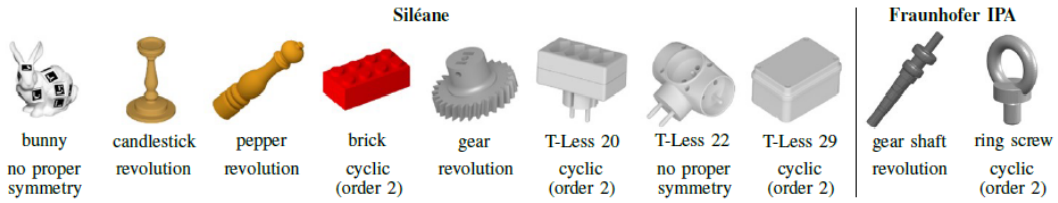


Figure 2.9: Fraunhofer industrial bin-picking Dataset [19].

ROBI[20] includes seven reflective mental parts that diameters vary from 76.2 mm to 24.5 mm. RGB images and depth maps were recorded for each part from multiple spherical views in two scenes (parts fully piled in the bin, and multiple parts without severe overlapping in the bin, as shown in Figure 2.10). Two stereo cameras - a

Ensenso® N35 and a RealSense® D415 - were used to capture the scenes. The scenes models were then reconstructed by two methods - a traditional TSDF (Truncated Signed Distance Function) fusion and a probabilistic fusion approach. The results show that the probabilistic fusion has better reconstruction performance than TSDF fusion for each object and camera. In addition, the reconstruction performance is dependent on the accuracy of depth data. Both probabilistic fusion and TSDF fusion show better performance with data captured by Ensenso® N35 than that by RealSense® D415. The ground truth annotations are 6D poses of visible parts with visibility scores.

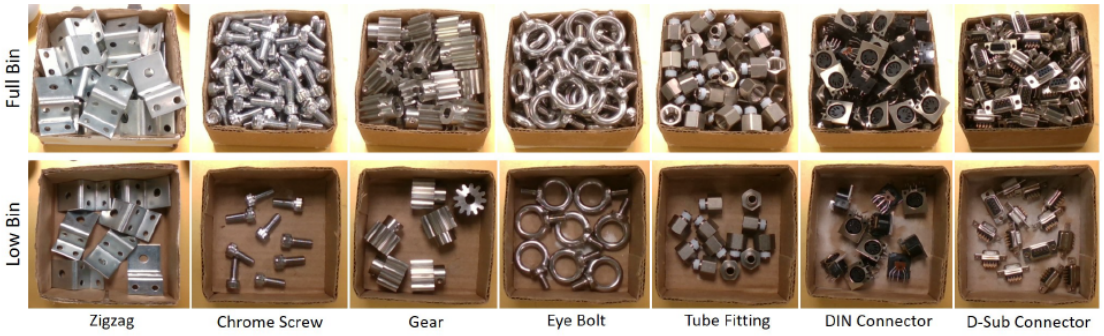


Figure 2.10: ROBI: seven reflective parts in two scenes. [20]

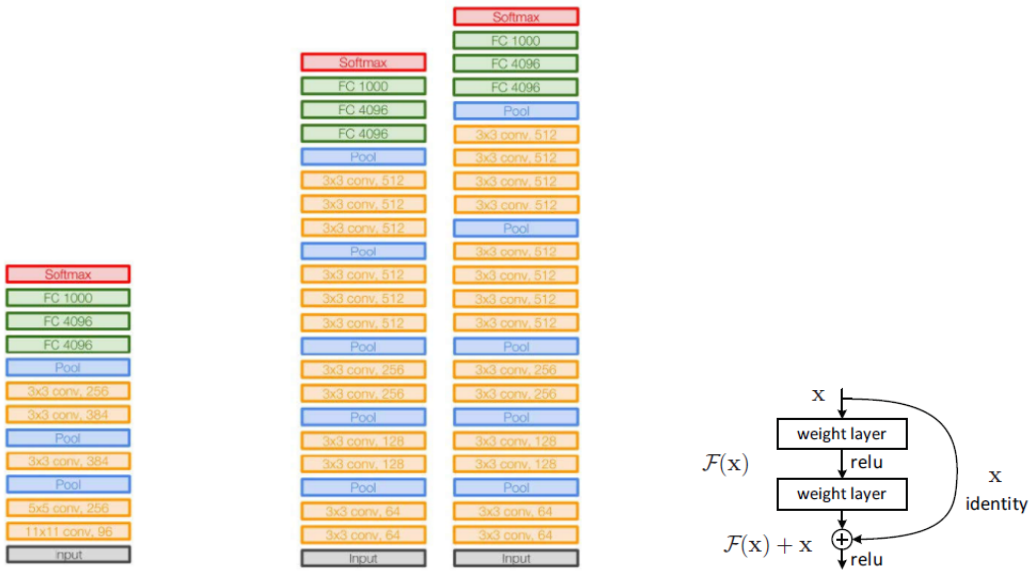
Zeng et al. [14] trained models over a densely human-labeled dataset including 1837 RGB-D heatmaps with suction and grasp labels. The grasp labels were augmented by jittering within 1.6 cm.

Some datasets also leveraged other available datasets. Dex-Net 1.0 [21] contains 13252 3D mesh models from different sources. Dex-Net 2.0 [17] contains over 6.7 million rendered depth images with grasp pixel position and orientation based on 1500 3D meshes from Dex-Net 1.0. Hundreds parallel-jaw grasps were sampled over the whole surface of each object. The grasps were then aligned with the rendered point cloud of objects in stable poses to generate 32×32 grasp images.

Neural Networks

Convolutional Neural Network (CNN) is the backbone of machine learning in computer vision. In the last ten years, some typical CNNs were created and of great progress such as AlexNet, VGG, ResNet. The neural networks of most robot grasp methods reviewed in this report were built based on these typical CNNs.

In 2012, AlexNet [22] used Local Response Normalization (LRN) to improve gener-



(a) AlexNet. (b) VGG 16 & VGG19. (c) Residual Block of ResNet.

Figure 2.11: Typical CNNs.

alization. It contains five convolutional layers with ReLu activation function, and three fully connected layers, as visualized in Figure 2.11 (a). The input dimension is $256 \times 256 \times 3$. VGG [23], introduced in 2014, consists of convolutional layers, fully connected layers, and softmax, with up to 19 layers, as visualized in Figure 2.11 (b). In 2015, ResNet [24] significantly increased the layers depth up to 154 layers by residual block as interpreted in Figure 2.11 (c). It combines VGG and residual network that skips/shortcuts the connections between layers to solve vanishing or exploding gradients during backpropagation.

In 2015, Long et al. [25] presented fully convolutional neural network for Semantic Segmentation. Convolutional neural networks only take fixed dimensional input, due to the neurons number of the fully connected layers. In contrast, fully convolutional neural networks take arbitrary size images as input, and output the corresponding size, due to the lack of fully connected layers.

Fan et al. [13] deployed a region-based CNN to predict a potential grasp region. It fine-tuned the pre-trained AlexNet model with the new collected data. Zeng et al. [14] modifed a ResNet101 with fully convolutional layers to reduce the number of parameters and give dense prediction. RGB-D data was fed to estimate suction and

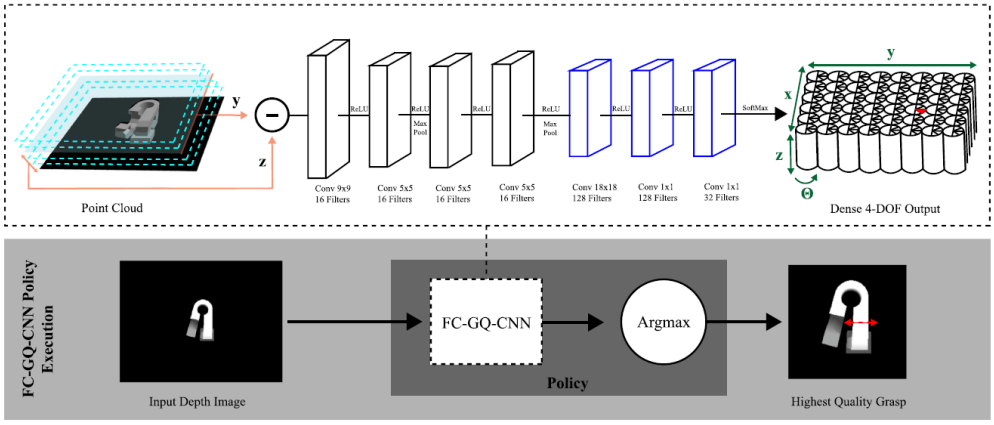


Figure 2.12: FC-GQ-CNN. The convolutional layers highlighted blue were converted from original fully connected layers of GQ-CNN. [10]

horizontal grasp affordance as mentioned in Section 2.2.2.

Mahler et al. [17] proposed a Grasp Quality Convolutional Neural Network (GQ-CNN) trained on Dex-Net 2.0 to predict probability of grasp success based on depth images and sampled grasps. Satish et al. [10] modified the GQ-CNN to fully convolutional GQ-CNN (FC-GQ-CNN) to improve speed by converting the fully connected layers of GQ-CNN into convolutional layers, as illustrated in Figure 2.12. It is not necessary to sample grasp poses over the object surface, due to the fact that fully convolutional neural network (FCN) gives dense pixel-wise grasp estimation. Furthermore, FCN has fewer parameters without fully connected layers. Therefore, the FC-GQ-CNN is faster than sample-based GQ-CNN.

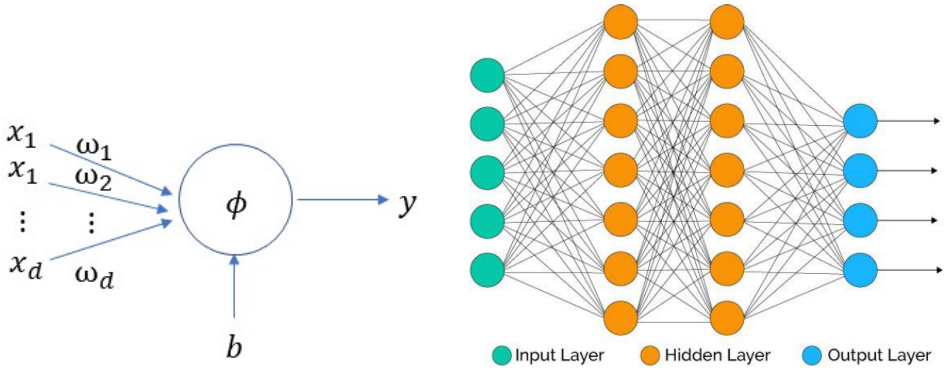
2.2.3 Machine Learning Basics

Artificial Neurons

Artificial neurons and neural networks are built by mimicking humans' neurons. Figure 2.13 (a) presents the basic element of neural network in deep learning - artificial neuron,

$$y = \sigma\left(\begin{bmatrix} w_1 & w_2 & \dots & w_d \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} + b\right) = \sigma\left(\mathbf{w}^T\mathbf{x} + b\right) \text{[45]} \tag{2.1}$$

where $\mathbf{x} \in \mathbb{R}^d$ refer to input tensors (e.g. vector or image), $\mathbf{w} \in \mathbb{R}^d$ represent weights, $b \in \mathbb{R}$ is a scalar number, referring to bias. $\sigma(\cdot)$ is an activation function, y



(a) Basic Element - Neuron [46]. (b) Multi-layer Neurons [45].

Figure 2.13: Artificial Neurons.

is output.

Multiple neurons consist of a layer, and multiple layers comprise a neural network, usually including an input layer, an output layer, and several hidden layers, as visualized in Figure 2.13 (b). The purpose of building a neural network is to learn a map (parameters $\mathbf{w}, b$) between input and real output to estimate unknown output given input [46]. The type of activation function $\sigma(\cdot)$, the number of layers and how the neurons connected with each other form a structure/architecture of a neural network. These are hyper-parameters tuned by the human and cannot be learned.

Activation Function

The first neural network in history is a linear perception by Rosenblatt in 1957, using linear activation function and no hidden layers only for linear tasks [46]. Nowadays, the neural networks usually use nonlinear activation functions and hidden layers to solve complicated tasks in the real world. If the activation function is linear, then the chain connection between multiple layers is meaningless, because the affine functions chain is equivalent to a single affine function [46].

Activation function maps summed weighted and biased input to the output, and governs which neuron to be activated and how much it is activated, as its name suggests. An activation function should be generally nonlinear, differentiable for training, and easily calculating to reduce complexity [46].

Rectified Linear Units (ReLU) is a common activation function,

$$f(x) = \max(0, x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases} \text{[46]} \tag{2.2}$$

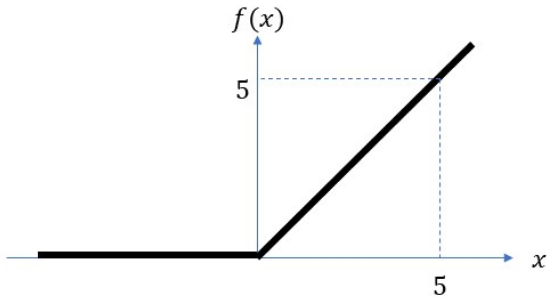


Figure 2.14: Rectified Linear Units (ReLU).

It works like an electrical diode - the neuron is activated only when $x > 0$, otherwise the value of this neuron does not pass on to next layer, and the neuron is dead [46]. Besides, ReLU is simple in mathematics, so it saves computation time [46]. In practice, ReLU is usually a default choice in most cases. Other activation functions are then considered if ReLU does not work well.

Softmax is another popular activation function especially applied in multi-class classification tasks [46],

$$\begin{aligned} &\mathbf{a} = [a_i] \in \mathbb{R}_K \\ &\sigma_i(\mathbf{a}) = \frac{e^{a_i}}{\sum_{k=1}^{K} e^{a_k}} \quad \text{for } i = 1, \ldots, K \\ &\sum_{i=1}^{K} \sigma_i(\mathbf{a}) = 1 \end{aligned} \qquad [46] \tag{2.3}$$

where $\mathbf{a}$ is a set of classes, $\sigma_i(\mathbf{a})$ is the probability of the class a_i, and probabilities sum of all classes equal to 1. In addition, Softmax is often the output layer for classification problems to output probability values.

Loss Function

Loss function (or cost function) L represents the precision of estimated output $\hat{y}_i$ compared to the ground truth value y_i. It is a criteria to measure the performance of the network estimation. For regression tasks, L_1 Loss, andL_2 Loss are widely used,

$$\begin{aligned} L_1 \text{ Loss: } \mathcal{L}_1 &= \sum_{i=1}^{N} |y_i - \hat{y}_i| \\ L_2 \text{ Loss: } \mathcal{L}_2 &= \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \end{aligned} \qquad [45] \tag{2.4}$$

L_1 loss sums the absolute difference between the predicted value and the actual value. L_2 loss calculates the sum of the Euclidean distance between the predicted value and the actual value, so it is also called mean square error (MSE) if averaging that sum.

For classification problem, Cross Entropy Loss $\mathcal{L}_{CE}$ is typical. It evaluates the distance between the predicted probability distribution obtained by training and the true probability distribution of the available training set.

$$\mathcal{L}_{CE} = -\frac{1}{N}\sum_{i=1}^{N} y_i^{a_i} \log_2 \left(\hat{y}_i^{a_i}\right) [45] \tag{2.5}$$

Optimization Algorithms for Machine Learning

The goal is to learn the weights $\mathbf{w}$ that minimize the loss function, which turned into an optimization problem. Assume an objective function $L\left(\boldsymbol{\theta}; \mathbf{x}; \mathbf{y}\right)$, and $\mathbf{x}, \mathbf{y}$ are variables, find a parameter $\boldsymbol{\theta}$ that minimize $L\left(\boldsymbol{\theta}\right)$,

$$\arg \boldsymbol{\theta} = \min_{\boldsymbol{\theta}} L\left(\boldsymbol{\theta}; \mathbf{x}; \mathbf{y}\right) [46] \tag{2.6}$$

Due to the fact that the system is nonlinear, the numerical minimization problem has no closed-form solutions [46]. Gradient descent is an intuitive method to reach the goal step by step. It is commonly used for optimization of neural network [47]. Usual gradient-based optimization algorithms conclude Stochastic Gradient Descent (SGD), Momentum, Nesterov accelerated Gradient, Adaptive Gradient Algorithm (Adagrad), Adadelta, Root Mean Square Propagation (RMSprop), Adaptive Moment Estimation (Adam), AdaMax, Nadam [47].

Stochastic Gradient Descent (SGD) is the fundamental. It randomly takes n samples from the whole training set as a mini-batch iteratively, and then calculate $\nabla_{\boldsymbol{\theta}} L$ and update θ for each mini batch,

$$\boldsymbol{\theta}_{j+1} = \boldsymbol{\theta}_j - \eta \cdot \nabla_{\boldsymbol{\theta}_j} L\left(\boldsymbol{\theta}; x^{(i:i+n)}; y^{(i:i+n)}\right) [47] \tag{2.7}$$

where $\mathbf{x}$ refer to training samples, and $\mathbf{y}$ refer to the labels, η is step size at each update or called learning rate, j is iteration index.

Nevertheless, optimization problem encounters six main difficulties:

1. stochastic gradient.

2. ill conditioning.

3. saddle point/plateau.

4. sensitive to step size.

5. local minimum.

6. vanishing gradient.

Therefore, variants of the optimization techniques are proposed to improve.

Momentum is a significant modification of SGD. It reduces noise in stochastic gradient, decreases oscillation and accelerates convergence by adding a factor related to the previous step,

$$\begin{aligned} \Delta\boldsymbol{\theta}_j &= \gamma\Delta\boldsymbol{\theta}_{j-1} + \eta\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})[47] \\ \boldsymbol{\theta}_{j+1} &= \boldsymbol{\theta}_j - \Delta\boldsymbol{\theta}_j[47] \end{aligned} \tag{2.8}$$

where $\gamma\Delta\boldsymbol{\theta}_{j-1}$ is a momentum part, γ is momentum factor, $0 \leq \gamma < 1$, and $\eta\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$ is a gradient part.

So far, the learning rate is fixed and tuned manually. However, if the constant learning rate is too small, the convergence is slow. If it is too large, then there might be oscillation around local minimum [46]. Therefore, other methods such as AdaGrad, AdaDelta, RMSprop, and Adam, are employed to adaptively schedule the learning rate [47].

Among them, Adam [49] is usually a default optimizer in practice to do backpropagation. The pseudo-code of Adam algorithm is summarized in Figure 2.15. Assume $g_{t,i}$ is the partial derivative/gradient of the objective function L w.r.t. the parameter θ at time step t,

$$g_t = \nabla_{\theta_t} L\left(\theta_t\right)[49] \tag{2.9}$$

Adam employs not only the first moment estimate - exponential moving averages of the gradient m_t, and the second moment estimate - the squared gradient v_t, where the hyper-parameters $\beta_1, \beta_2 \in [0, 1)$ control the exponential decay rates of these moving averages, g_t^2 means $g_t \odot g_t$,

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + \left(1 - \beta_1\right) g_t[47] \\ v_t &= \beta_2 v_{t-1} + \left(1 - \beta_2\right) g_t^2[47] \end{aligned} \tag{2.10}$$

However, m_t and v_t are biased towards zero at initial steps or if β_1 and $\beta_2 \approx 1$, because they are initialized as zeros. Accordingly, bias-corrected moments estimates $\hat{m}_t$, $\hat{v}_t$ are applied,

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1 - \beta_1^t}[47] \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t}[47] \end{aligned} \tag{2.11}$$

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Figure 2.15: Pseudo-Code of Adam Optimization Algorithm [48].

Finally, the parameter θ_t updates as follows,

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t [47] \tag{2.12}$$

Convolutional Neural Network

Figure 2.13(b) visualizes a fully connected network. Every neuron in a layer is connected to all neurons in the next layer, which generates a huge number of parameters. Convolutional neural network (CNN) proves promising performance especially in image processing [50]. It is capable of abstracting local features by deploying a two dimensional digital filter sliding over the input image [50]. Cross-Correlation (also called convolution in machine learning libraries) is usually implemented in practice, which is equivalent to flipped commutative convolution [49],

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i + m, j + n) K(m, n) [49] \tag{2.13}$$

the two-dimensional filter K is also called kernel, or filter matrix, $\mathbf{I}$ is a two-dimensional input image, and $\mathbf{S}$ is output in the next layer. $\mathbf{S}$ aggregates dot products between the kernel and a part of the image. Figure 2.16 demonstrates a simple convolution over an image - a 4×3 input convolves a 2×2 kernel results

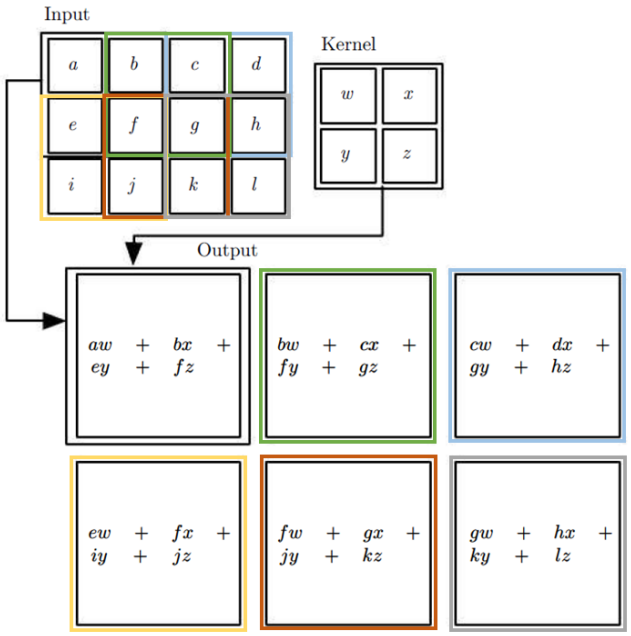


Figure 2.16: An example of 2-D convolution [49].

in a 3×2 output.

2.2.4 Summary of Current Approaches

Significant information of the learning-based methods reviewed above are summarized in the Table 2.1.

2.3 Actuation

Robot kinematics considers robot motion, without considering forces and joint torque of dynamics, As written in [28]. In robot motion, Cartesian space (also called workspace, task space) and joint space (also called configuration space) define robot movement in two different reference coordinates. Forward kinematics solves position and orientation of the end effector according to joint angle values in the joint space. In contrast, inverse kinematics computes joint angles given a desired end effector pose in the Cartesian space.

Table 2.1: Overview of learning-based approaches

Method	Gripper Type	Camera Installation	Input Data Type	Prior Object Model Knowledge?	Grasp Candidates Sampling?	Neural Network Type	Output Type
Morrison et al. [5]	parallel jaw	eye-in-hand	depth image	no	no	fully convolutional NN	3D position, 1D rotation
Zeng at al. [14]	vacuum gripper, parallel jaw	eye-to-hand	RGB-D	no	no	fully convolutional ResNet101	3D suction, 4D grasp
Fan et al. [13]	parallel jaw	eye-to-hand	point cloud	no	yes, a guided sampling	region-based CNN	n.a.
Dex-Net 2.0 [17]	parallel jaw	eye-to-hand	point cloud	yes	yes, uniformly and randomly	CNN	3D position, 1D rotation
Fujita et al. [12]	parallel jaw, vacuum gripper, suction gripper	eye-in-hand	single depth image, point cloud	primitive shape models	no	n.a.	object pose
Satish et al. [10]	parallel jaw	eye-to-hand	point cloud	yes	yes, sampling uniformly	fully convolutional NN	3D position, 1D rotation
GDP [26]	parallel jaw	eye-in-hand	point cloud	no	yes, sampling uniformly randomly in ROI	CNN	6DoF Grasp Pose
GraspNet [27]	parallel jaw	eye-in-hand	point cloud	no	yes, sampling via Variantional Autoencoder(VAE)	PointNet++ (encoder, decoder)	6DoF

2.3.1 Inverse Kinematics

Inverse Kinematic problem can be written as:

$$q = \kappa^{-1}(\xi_E)[28] \tag{2.14}$$

where q is joint angles vector, and ξ_E is the desired pose of end effector in Cartesian space.

In simple cases, the problem can be solved by analyzing geometry of the robot manipulator and algebraic calculation, which is called closed form or analytic solution [28]. However, in the real world, the robot manipulators usually have more than three joints, so the closed form solution does not always exist. Therefore, numeric approach is more common. It iteratively minimizes the error between the forward kinematics result and the desired pose [28]. Notably, the solution is not unique [28]. It is possible that different joint angles vectors lead to the same end effector pose.

2.3.2 Motion Planning

Motion planning is a critical research topic for autonomous robots [29]. The robot executes planned motions to perform tasks in the workspace in real world [29]. Automatic motion planning solves a collision-free path from robot current state to a goal state by interpolating points between the two states [30]. It also considers robotic kinematic constraints and computation complexity of the algorithm [29].

The basic motion planning problem is formulated as follows:
- A rigid robot moves in a 2 dimensional or 3 dimensional Euclidean space (also called workspace) [29].
- Fixed rigid objects are placed in the workspace, called obstacles. Geometric relationships between the obstacles and the robot are known [29].
- The robot motion has kinematic constraints due to the mechanical and electronic designs, such as robot pose, maximal speed, acceleration [30].
- The goal is - planning a sequence of positions and orientations of the robot in the workspace, so that the robot reaches a goal pose from an initial pose, without crashing obstacles [29]. Report error if no feasible path found [29].

Recent motion planning algorithms are mostly sampling-based [30]. Sampling-based motion planning samples the state space of the robot that refers to all possibilities of robot configuration in the workspace [31]. The samples are then connected from a start state to a goal state to propose a collision-free path satisfying constraints [31]. Sampling strategies are various, such as Probabilistic Roadmap (PRM), and Tree-based Planners [31]. PRM samples randomly the state space and builds a roadmap in the free space that is similar to a city street map [31]. Tree-based planners start

with a root node at the robot start state, and then expand towards to the goal [31]. Except from sampling-based approaches, other algorithms such as Control-based Methods, Potential Fields, Randomized Planning, are also available [31].

Kinematics Libraries

Hundreds of open source kinematics libraries are available [32] to optimize robot motion planning using various algorithms. For instance, Kinematics and Dynamics library (KDL) [33] and IKFast Kinematics Solver [34] are widely used. The Open Motion Planning Library (OMPL) [31] and Pilz industrial motion planner [35] are state-of-the-art algorithms offering promising solutions. The OROCOS KDL solves inverse kinematic problems by iteratively solving linear least squares problem [36]. It showed 85% average success rate and less than 2.4 milliseconds average planning time [36]. The success rate is not high, and the KDL is not robust enough. IKFast is a powerful inverse kinematics solver. It automatically analyses complex kinematic chain for common patterns, so that an analytic solution can be found [37]. It is also capable of finding extremely stable solutions in microseconds [37]. OMPL is a collection of sampling-based motion planning algorithms [31]. Pilz industrial motion planner [35] was presented at Moveit Workshop by Pilz Automation in 2019. It is capable of checking collision with the environment model. In this thesis, the inverse kinematic problem is solved via an open source software - ***MoveIt***, incorporating motion planning, manipulation, 3D perception, kinematics, control and navigation [38]. It integrates various kinematic solvers as plugins. The implementation of robot motion planning in *MoveIt* will be elucidated in Chapter 4.

Chapter 3

Methodologies

This chapter initially introduces how an industrial robotic grasp dataset is generated for this thesis, and then data preprocessing preparing for network training. Secondly, basics of neural network is reviewed. The network architecture applied in the thesis is then investigated in detail. Additionally, training pipelines of suction model and grasp model are illustrated respectively. Thirdly, calibration methods of a robot online teaching device - Tracepen™- are explicated.

3.1 Dataset

First of all, a data collection method with eye-in-hand camera configuration is proposed. Subsequently, captured data format and ground truth labeling is introduced. Last but not least, pre-process of raw data for neural network training is elaborated in detail.

3.1.1 Data Collection

Single or multiple parts are placed on a work table surface or in a bin. A camera is mounted on the end effector (eye-in-hand as shown in Figure 3.1). A robot trajectory that moves a constant distance at each step is feasible to capture the part from different views. However, in that way only one pose of the object is captured, because the camera orientation does not change. Therefore, it is necessary to create a trajectory that position and orientation change simultaneously, so that every face of the object can be captured. The movement trajectory is dome-shaped as visualized in Figure 3.2. The robot poses along the dome trajectory should be generated based on ellipsoid formulas and fitted to the robot coordinate system, as following equations 3.1 to 3.3:

Figure 3.1: Eye-in-Hand Camera Setup. Cameras are mounted on the robot end effector, and move along with the robot motion.

$$x = a \times \sin\theta \times \sin\phi + x_0 \tag{3.1}$$

$$y = -b \times \sin\theta \times \sin\theta + y_0 \tag{3.2}$$

$$z = c \times \cos\theta + z_0 \tag{3.3}$$

where a, b, c are ellipsoid parameters that are the farthest points along x, y, z axis respectively. θ is zenith angle, and ϕ is azimuth angle. θ and ϕ should be defined according to the robot workspace and the data requirement. In this case, $\theta = 175°, \phi = 45°$. (x_0, y_0, z_0) is the initial position of the camera.

3.1.2 Data Format

Color images, depth images, and the pose of end effector (in the format of homogeneous transformation matrix) are captured at each step of the dome trajectory, as visualized in Figure 3.3. Color images are only for better visualization to humans, and unused for robot grasping estimation. Depth images are deployed to predict robot grasp points. Depth image is the input data format that many robotic bin-picking approaches [5] [12] [14] used, and they achieved promising performances.

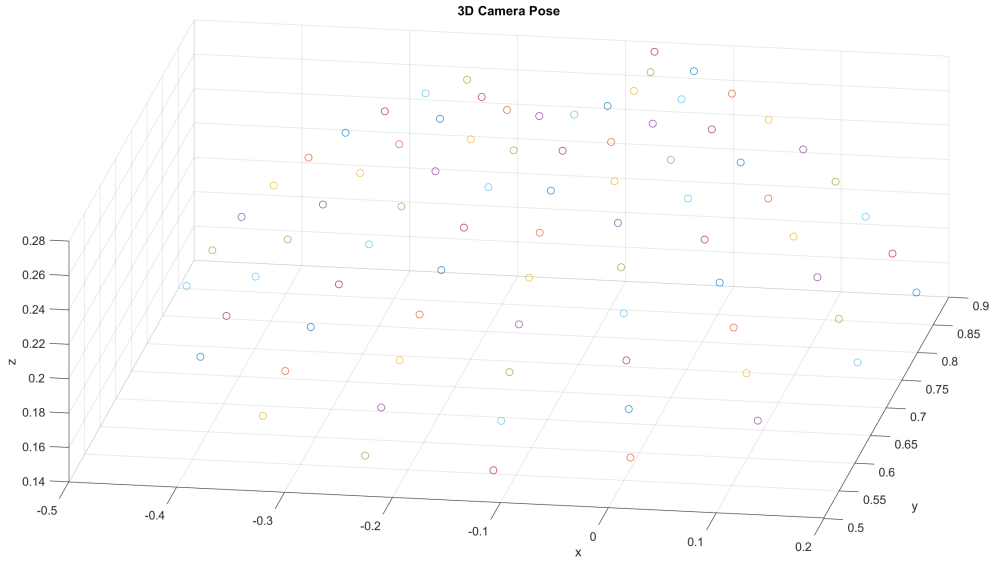
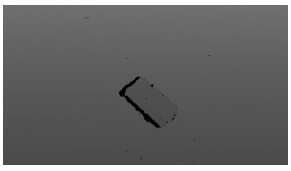


Figure 3.2: Robot Dome Trajectory to Scan around the Part.

(a) Color Image Example 1: Housing.

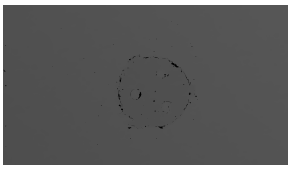


(b) Depth Image Example 1: Housing.

$$\begin{bmatrix} 2.174e-01 & -7.801e-01 & -5.865e-01 & 2.325e-02 \\ -2.396e-01 & -6.252e-01 & 7.427e-01 & 7.267e-01 \\ -9.461e-01 & -2.094e-02 & -3.229e-01 & 2.238e-01 \\ 0.000e+00 & 0.000e+00 & 0.000e+00 & 1.000e+00 \end{bmatrix}$$

(c) Camera Pose of Example 1.

(d) Color Image Example 2: Support Gearwheel.



(e) Depth Image Example 2: Support Gearwheel.

$$\begin{bmatrix} 6.758e-01 & -1.401e-01 & -7.236e-01 & -7.0834e-02 \\ -2.793e-02 & -9.859e-01 & 1.648e-01 & 8.913e-01 \\ -7.365e-01 & -9.120e-02 & -6.702e-01 & 2.769e-01 \\ 0.000e+00 & 0.000e+00 & 0.000e+00 & 1.000e+00 \end{bmatrix}$$

(f) Camera Pose of Example 2.

Figure 3.3: Raw Color & Depth Images Examples.

To author's understanding, depth image is employed instead of color image for the following reasons:

1) the object physical structure is more interesting for the bin-picking tasks, for instance, flatness, thickness, height. These properties decide how this object can be picked up successfully. It is of difference from classification problems using

computer vision, such as semantic segmentation and object detection, of which color information is essential to recognize.

2) It is more difficult to train the network if the objects have strong color gradients [42].

3) The trained network could be sensitive to colors. The performance would decrease, for example if color of background - bin or table surface, changes.

Nevertheless, in the work of thesis, I found that low cost depth cameras have fatal flaw - tiny objects are invisible under the depth camera. Moreover, depth perception of reflective parts often failed.

The amount of data including real-world data and augmented data (introduced in the following Section 3.1.4) is summarized in Table 3.1 and Table 3.2 for suction training and grasp training respectively. In total, 1464 data is deployed to train suction model and 240 data is used to validate. The grasp model is trained with 679 data, and validated with 60 data.

Table 3.1: Data Distribution for Suction Model Training & Validation

Part Name	Part Size(mm)	Data Amount for Training	Data Amount for Validation	Data Amount for Test
Housing	78×40×30	366	60	60
Gearwheel	130×130×25	366	60	60
Support Gearwheel	108×108×4	122	20	20
Baseplate	190×140×5	366	60	54
Motor	28×28×45	244	40	40
Total		1464	240	234

Table 3.2: Data Distribution for Grasp Model Training

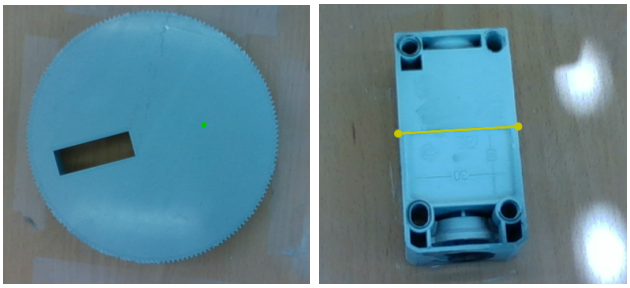
Part Name	Part Size(mm)	Data Amount for Training	Data Amount for Validation	Data Amount for Test
Housing	78×40×30	291	30	30
Gear Wheel	130×130×25	170	10	10
Motor	28×28×45	218	20	20
Total		679	60	60

3.1.3 Label

The ground truth of grasps is manually labeled using a label software called LabelMe [43]. Figure 3.4 presents suction and grasp labels, and the label file consists of

coordinates of label points in the image. The suction point represents a suctionable area for the vacuum gripper, which should be close to the center of mass and flat. The anti-podal grasp points are grasping contact points between the parallel-jaw and the object, represented by two end points of a straight line.

Suction labels have 3 groups: suctionL, suctionM, suctionS, representing the vacuum gripper with a large, a medium, and a small size of pad respectively. Grasp labels have two groups: graspI, and graspO, standing for parallel-jaw gripper outside and inside gripping respectively.



(a) Suction Label.

(b) Grasp Label.


```
"version": "4.5.10",
"flags": {},
"shapes": [
  {
    "label": "suctionL",
    "points": [
      [
        630.6774193548387,
        429.51612903225805
      ]
    ],
    "group_id": 0,
    "shape_type": "point",
    "flags": {}
  },
  {
    "label": "graspO",
    "points": [
      [
        577.4516129032259,
        373.06451612903226
      ],
      [
        701.6451612903226,
        387.5806451612903
      ]
    ],
    "group_id": null,
    "shape_type": "line",
    "flags": {}
  }
],
"imagePath": "..\\Color\\0000_image.png",
"imageData": null,
"imageHeight": 720,
"imageWidth": 1280
```


(c) Label Format.

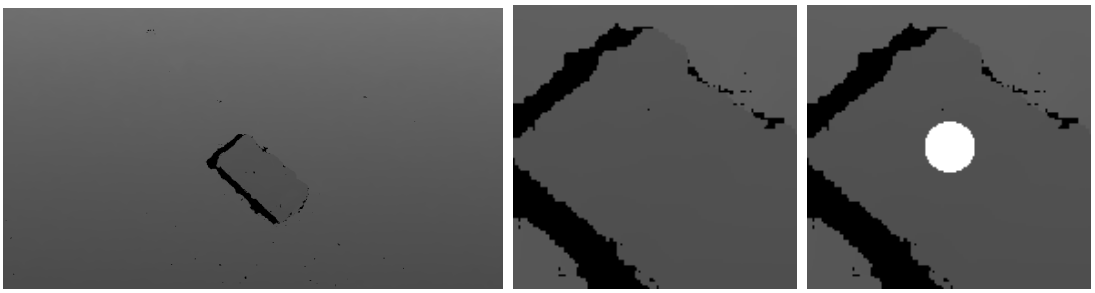
Figure 3.4: Label Examples.

3.1.4 Data Processing (Training Dataset Generation)

The data captured as in the Section 3.1.1 must be processed before feeding to the neural network. In this thesis, a suction model and a grasp model are to be trained, so the data should be processed correspondingly. The processed data is augmented to enlarge the dataset.

Data for Training Suction Model

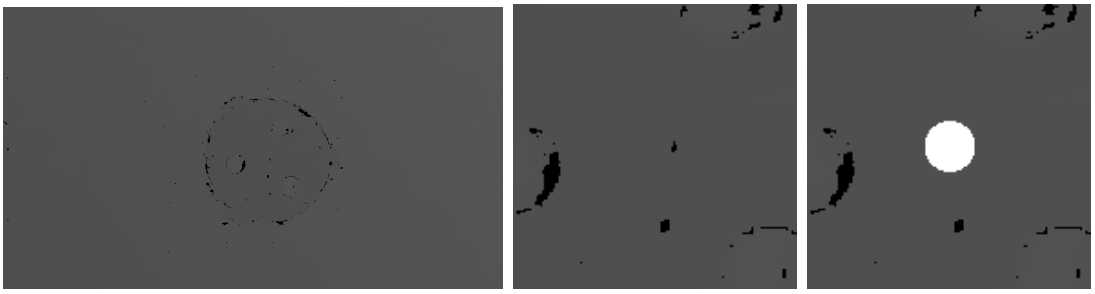
The suction model prediction is the 3D position of the vacuum gripper - $S = (x, y, z)$. Initially, the raw depth frames are converted into depth images in range of $(0, 255)$. The depth images are then center cropped in the label point (for instance, the suctionL label shown in the Figure 3.4). Next, the cropped depth images are normalized in $(0, 1)$, which is beneficial for the neural network to learn the model parameters. Furthermore, the labels loaded for training should also be the processed label images, instead of the label coordinates in label files shown Figure 3.4 (c). The neural network architecture planned to employ in this thesis predicts pixel-wisely, which will be elucidated in details in the next section, so the labels should be also densely labeled images according to the labeled suction point. Figure 3.5 visualizes depth images, cropped depth images, and position label images. The white circles in the Figure 3.5(c) and (f) are the labeled suctionable areas. The diameter of the suctionable areas are dependent on the label groups - suctionL: 15 pixels, suctionM: 20 pixels, and suctionS: 15 pixels. These pixel values correspond 4.5mm, 6mm, and 4.5mm of length in the real world respectively.



(a) Depth Image: Housing.

(b) Cropped Depth Image: Housing.

(c) Croppped Position Label Image: Housing.



(d) Depth Image: Support Gearwheel.

(e) Cropped Depth Image: Support Gearwheel.

(f) Croppped Position Label Image: Support Gearwheel.

Figure 3.5: Depth Images & Position Label Images Examples.

Data for Training Grasp Model

The data processing for grasp model is more complicated than that for suction model, because the grasp model should predict not only the grasp position, but also an orientation. Therefore, the label images consists of position images, and angle images. As illustrated in Figure 3.6, two types of label are possible according to collected data - $L_1L_2, \text{and} L_3L_4$. In the image coordinate $x-y$, the rotation angle θ around z axis is calculated by slope s of the label straight line $L_1L_2, L_1 = (x_1, y_1), L_2 = (x_2, y_2)$,

$$\begin{aligned} s &= \frac{y_2 - y_1}{x_2 - x_1} \\ \theta &= \arctan(s) \end{aligned} \tag{3.4}$$

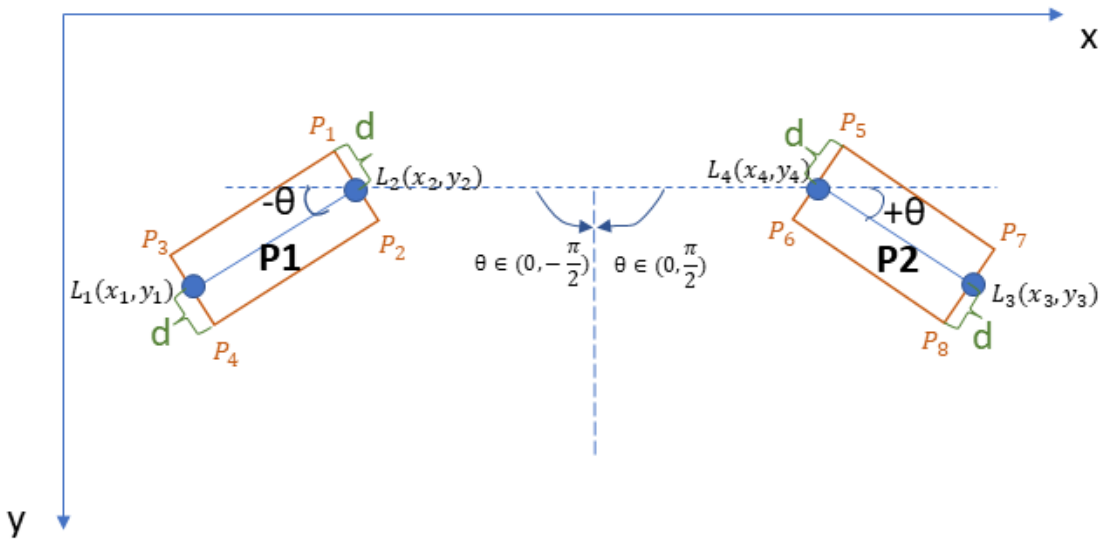


Figure 3.6: Grasp Angle Calculation.

The rectangle areas **P1**, **P2** shown in Figure 3.6 generated based on the label points are graspable. The four vertices of the rectangle **P1**: $P_1\left(x_{P1}, y_{P1}\right), P_2\left(x_{P2}, y_{P2}\right), P_3\left(x_{P3}, y_{P3}\right), P_4(x_{P4}, y_{P4})$ are calculated from label points L_1, L_2, and d is half of the rectangle side length,

$$\begin{aligned} x_{Pi} &= x_i + j \cdot d \cdot sin(\theta) \\ y_{Pi} &= y_i + k \cdot d \cdot cos(\theta) \end{aligned} \tag{3.5}$$

where $i \in (1, 4)$, when $i = 1$ or 3, $j = 1$ $k = -1$, and when $i = 2$ or 4, $j = -1$ $k = 1$.

The four vertices of the rectangle **P2**: $P_5\left(x_{P5}, y_{P5}\right), P_6\left(x_{P6}, y_{P6}\right), P_7\left(x_{P7}, y_{P7}\right), P_8(x_{P8}, y_{P8})$

are calculated from label points L_3, L_4,

$$\begin{aligned} x_{Pi} &= x_i + j \cdot d \cdot cos\left(\frac{\pi}{2} - \|\theta\|\right) \\ y_{Pi} &= y_i + j \cdot d \cdot sin\left(\frac{\pi}{2} - \|\theta\|\right) \end{aligned} \tag{3.6}$$

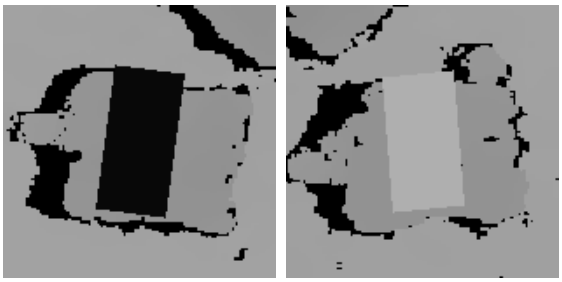
where $i \in (5, 8)$, when $i = 5$ or 7, $j = -1$, and when $i = 6$ or 8, $j = 1$. The position label image is generated by drawing a rectangle like **P1** and **P2**, to be a graspable area, as shown in Figure 3.7.

(a) Position Image, when $\theta > 0$.

(b) Position Image, when $\theta < 0$.

Figure 3.7: Position Images.

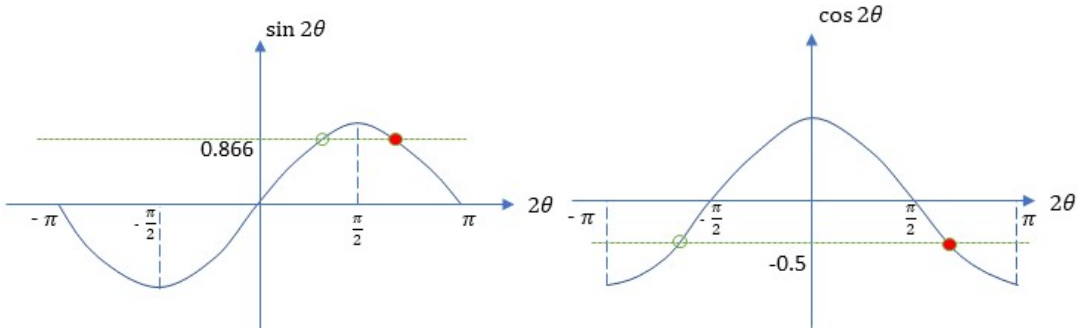
The angle images would be more complex. The raw rotation angle around z axis $\theta \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$. However, the angle image has discontinuity around $-\frac{\pi}{2}$ and $\frac{\pi}{2}$, if the raw angle is used to generate angle image. Figure 3.8 presents that the rotation angles around $-\frac{\pi}{2}$ and $\frac{\pi}{2}$ are close, but the encoding by raw angle image values is of great difference. The discontinuity is difficult for the neural network to learn the object orientation [5]. Furthermore, if sine and cosine function are utilized to encode the orientation, it is still discontinuous, because $\sin\frac{\pi}{2} = 1, \cos\frac{\pi}{2} = 0$, but $\sin\left(-\frac{\pi}{2}\right) = -1, \cos\left(-\frac{\pi}{2}\right) = 0$. $\sin\theta$ still has discontinuity around $-\frac{\pi}{2}$ and $\frac{\pi}{2}$, and using only $\cos\theta$ cannot distinguish the sign of the angle value. Therefore, in order to solve the discontinuity and represent the orientation uniquely, $\sin(2\theta), \cos(2\theta)$ are deployed [5]. For instance, when $\theta = \frac{\pi}{3}, \sin(2\theta) = 0.866, \cos(2\theta) = -0.5$. A unique θ can be deduced from $\sin(2\theta)$ and $\cos(2\theta)$. As illustrated in Figure 3.9, $\sin(2\theta)$ determines that the sign of the angular value must be positive, and $\cos(2\theta)$ clarifies the angular value $|2\theta| = \frac{2\pi}{3}$. Therefore, $2\theta = \frac{2\pi}{3}$ and $\theta = \frac{\pi}{3}$.



(a) θ around $-\frac{\pi}{2}$, the angle label encoded by raw angular value.

(b) θ around $\frac{\pi}{2}$, the angle label encoded by raw angular value.

Figure 3.8: Orientation Discontinuity. The angular values of $\theta = -\frac{\pi}{2}$ or $\theta = \frac{\pi}{2}$ are of big difference, even though they are actually very close.



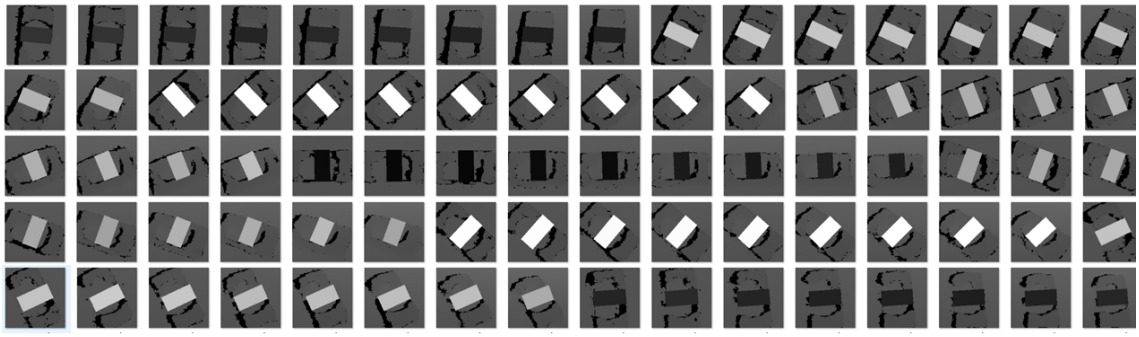
(a) $\sin(2\theta), \theta \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$.

(b) $\cos(2\theta), \theta \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$.

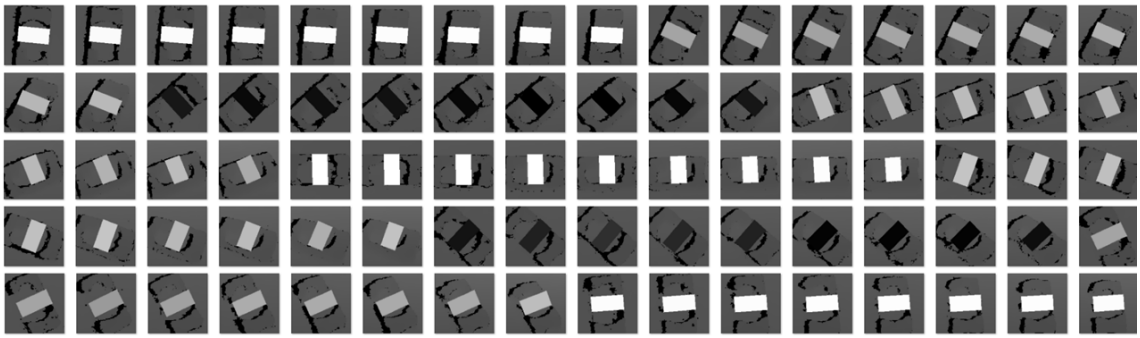
Figure 3.9: Angle Representation. The orientation θ is represented by $\sin(2\theta)$ and $\cos(2\theta)$, in order to solve discontinuity and deduce an unique angle back.

The $\sin(2\theta)$ images and the $\cos(2\theta)$ images are visualized in Figure 3.10. Only for image visualization purpose, the negative angle trigonometric function values are setup positive. Figure 3.10 (a) shows $\sin(2\theta)$ images when the rotation angle θ varies from around $0 \to \frac{\pi}{2} \to -\frac{\pi}{2} \to 0$. The $|\sin(2\theta)|$ has maximal value 1, when $2\theta = \pm\frac{\pi}{2}$, and $|\sin(2\theta)|$ has minimal value 0, when $2\theta = 0, \pm\pi$. Thus, the $\sin(2\theta)$ image has a label with a maximal pixel value, when when $\theta = \pm\frac{\pi}{4}$, and the $\sin(2\theta)$ image has a label with a minimal pixel value, when when $\theta = 0, \pm\frac{\pi}{2}$. The $\cos(2\theta)$ images are similar, varying from around $0 \to \frac{\pi}{2} \to -\frac{\pi}{2} \to 0$, as presented in Figure

3.10 (b).



(a) $\sin(2\theta)$ images, $\theta \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$. At begining, θ starts at around $\frac{\pi}{2}$, decreases to 0 in the middle of the third row, and then keeps dropping until $-\frac{\pi}{2}$ in the end.



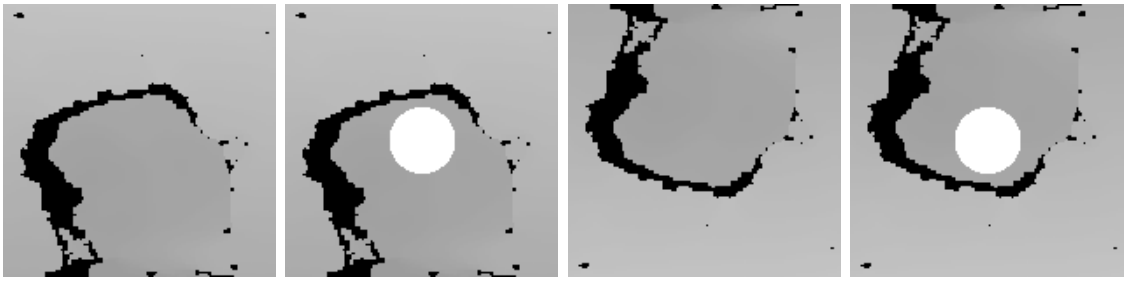
(b) $\cos(2\theta)$ images, $\theta \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$. θ changes in the same way as in (a) above.

Figure 3.10: Angle Label Images.

Data Augment

Data augmentation is a common technique to enlarge the dataset. It modifies data collected from real world by geometric transformations, flipping, color modification, cropping, rotation, noise injection and random erasing [44]. In the thesis, the processed training data for suction model is augmented by vertical flipping, because information of the object on one side is fully captured by the dome trajectory as shown in Figure 3.2, and the information from the other side lacks. Figure 3.11 represents processed data, the label image of a motor, and their augmentation by

vertical flipping.



(a) Processed Depth Image. (b) Processed Label Image (Label Group: suctionM). (c) Augmented Depth Image. (d) Augmented Label Image (Label Group: suctionM).

Figure 3.11: Data Augmentation.

The training data for grasp model can be also augmented in this way. The depth image and the position label image can be directly vertical flipped as suction data. Nevertheless, the rotation angle must be calculated again based on the geometry relationship, when flipping the sine and cosine angle images. Thus, the angle label encoding is different from the angle encoding before flipping.

3.2 Neural Network

Convolutional Neural Network (CNN) is a common and promising solution for computer vision tasksc [8]. In order to avoid grasps sampling and speed up computation time as introduced in Section 2.2.2, a variant of CNN - Fully Convolutional Neural Network (FCNN) is initially taken into consideration in this thesis. A CNN can be converted to a FCNN in two ways: firstly, directly training the FCNN without fully connected layers given pixel-wisely labeled dataset; secondly, training the CNN with normal dataset (not densely labeled), and then convert the fully connected layers of the trained model to convolutional layers [10].

3.2.1 Network Architecture

A fully convolutional neural network (FNN) is applied in the thesis, inspired by [5]. The reasons for using FNN are:

1. Arbitrary size of images can be fed to the network, which is more flexible than normal convolutional neural network (CNN) with a fixed input size. Convolu-

tional neural network contains fully connected layers whose neuron numbers are dependent on the input scale. Therefore, the network architecture has to be modified if the input size varies.

2. FCNN takes densely labeled images as input, and also predicts grasp quality pixel-wisely. On the contrary, the robotic grasping methods using CNN must firstly sample grasp candidates over images or point clouds, and then rank the grasp success probability of the candidates by the CNN, such as Dex-Net 2.0 [17]. Therefore, generating grasp candidates can be skipped with FCNN.

3. Compared to CNN (usually millions parameters), the parameter number of FCNN is much fewer (tens of thousands), due to the lack of fully connected layers and shared parameters. It leads to faster training and running speed, which is beneficial in industrial application scenarios.

Figure 3.12 illustrates the network architecture in details. Nine convolutional layers are deployed including the output layer. ReLU activation function, as elucidated in Section 2.2.3, is applied after each convolutional layer. Other techniques such as padding, stride, pooling, dilation, and upsampling are also employed. In the first layer, 16 kernels in the size of 11×11 are used, which means the input depth image is passed into 16 independent channels. Padding extends the image with zeros around the image [46], as shown in Figure 3.13. When padding = 5, five rows and five columns zeros are padded all around the 156×156 image, so it outputs a 166×166 image. Stride defines how many steps the kernel moves between each convolution [46]. When stride is 2, the kernel scans over the image and moves 2 steps each time. Max pooling is a common technique to downsample the image [46]. It replaces a slice with a pixel of maximal value in the slice, so the feature maps' dimensions are reduced. Figure 3.14 demonstrates max pool with 2×2 filters and stride 2. The dimension of the input is halved. Dilated convolution (dilation) is a modification of convolution applied to input signals with a dilation distance larger than 1. It increases the receptive field on the image, so that the receptive field becomes larger than the kernel size [46]. Upsampling interpolates two-dimensionally between pixels on the image [46]. It has different modes that determines values of the interpolated pixels. For instance, bilinear upsampling interpolates new pixels whose values are linearly related to the old pixels between two old pixels [51]. Bilinear interpolation is used in this FCNN architecture. Finally, the last layer is also a convolutional layer, instead of a fully connected layer in normal CNN, so the FCNN output is pixel-wise suction quality images. The pixel value represents the suction quality, so the position of which pixel value is the largest is the predicted suction point with best suction quality (the red point shown in the output image of Figure 3.12).

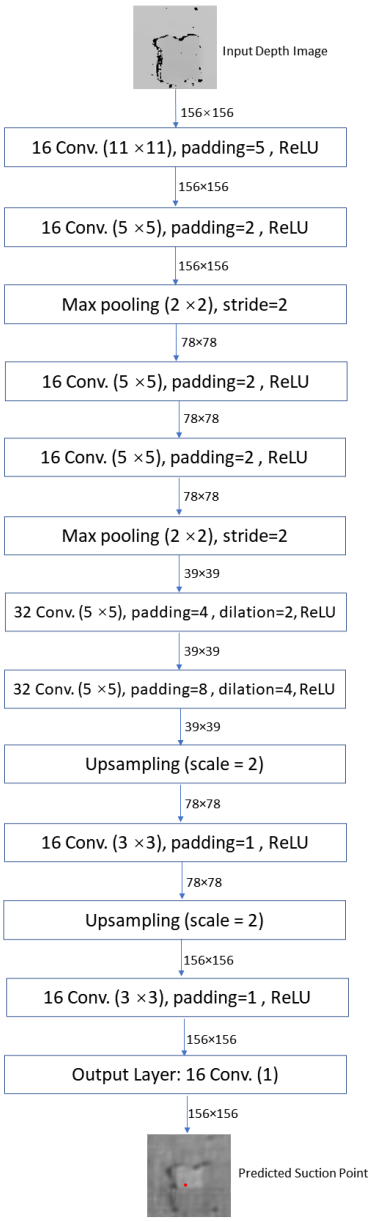


Figure 3.12: FNN Architecture. It consists of nine convolutional layers without fully connected layer.

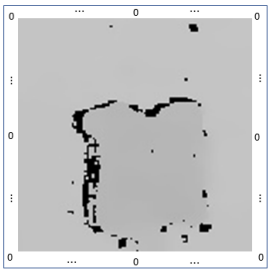


Figure 3.13: Zero-Padding. Padding zeros all around the image to enlarge size of image, so that pixels near edges can also be processed.

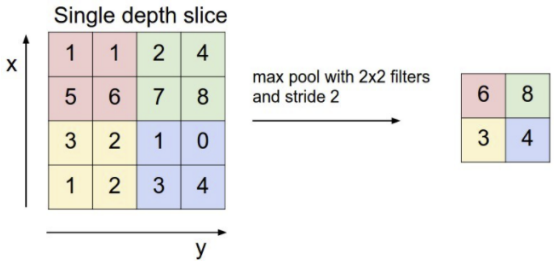


Figure 3.14: 2×2 Max Pooling. The maximal value in each 2×2 slice is kept to represent the slice [50].

3.2.2 Training Pipeline

The models for grasping with parallel-jaw and suction with vacuum gripper are trained respectively to predict suctionable point and anti-podal grasp pose. The suction prediction is a position in 3D space $S = (x, y, z)$, and the grasp prediction is 4 degree of freedom (DoF) $G = (x, y, z, \theta)$ consisting of 3D position and rotation angle θ around z axis.

Training for Suction Position Prediction

The suction model has one input channel, and one suction position groudtruth channel. Figure 3.15 illustrated the training and validation pipeline of the suction model. The raw depth image is preprocessed by cropping around the labeled suction center point, and the pixel values are normalized in the range of $(0, 1)$, as demonstrated in the Section 3.1.4. The cropped Region of Interest (ROI) of depth images in the format of numpy array is input to the Xavier initialized neural network. The mean square error (MSE, or L2 norm) $\ell(x, y)$, as elaborated in Section 2.2.3, is applied to compute the loss between the neural network output x and the target position label image y,

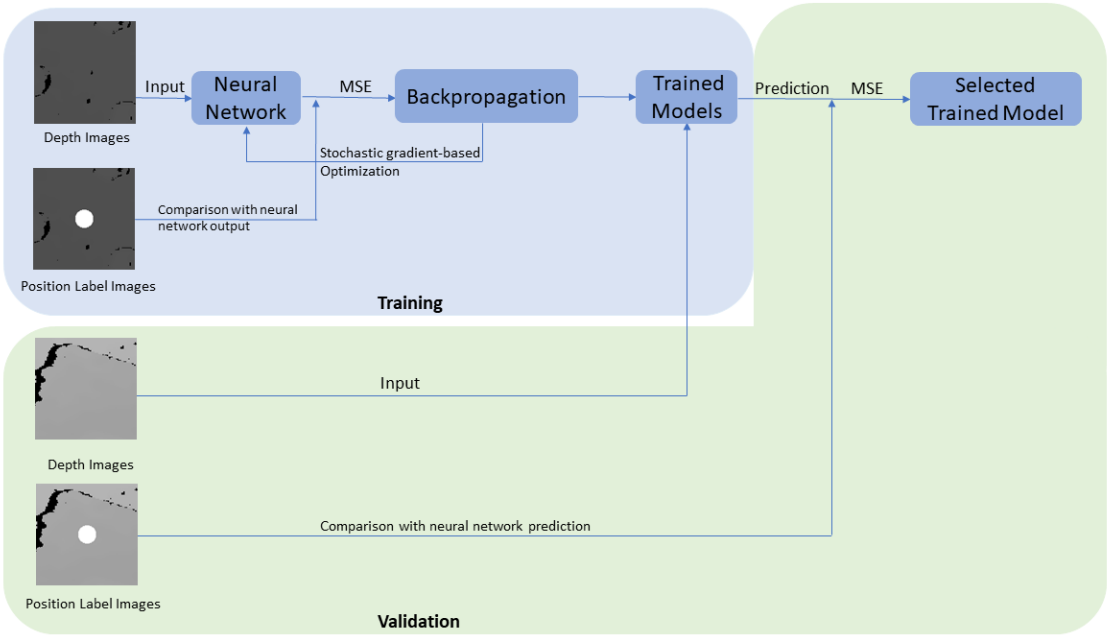


Figure 3.15: Training Pipeline of Suction Model. The blue part shows training of . The part in green validates trained models from training part by selecting the model with minimal validation loss.

$$
\begin{aligned}
l_n &= (x_n - y_n)^2 , \\
\ell(x, y) &= \text{mean}(L) ,[22] \\
L &= \{l_1, \ldots, l_N\}^\top .
\end{aligned}
\tag{3.7}
$$

where x_n, y_n are pixel values of x, y respectively, N is the batch size.

Backpropagation is the most popular learning algorithm to train the neural network. It computes the gradient of loss function L based on the optimization algorithms like gradient descent elucidated in Section 2.2.3, in order to learn the network weights and minimize the loss (deviation between the network output and the ground truth). The network is trained with 1464 training data using Adam optimizer with a learning rate of 0.001, and validated with 240 validation data.

Examples of preprocessed input images, and the corresponding ground truth images fed to neural network training are visualized in Figure 3.5. The pixel position with maximal value on the grasp image will be the proposed as the suction point, because the pixel value represents the grasping quality (higher the pixel value, better the quality). The training should be validated by comparing the output grasp images processed by the trained models with the ground truth labeled images to select a

final trained model with the lowest validation error.

Training for Grasp Pose Prediction

The training of grasp model takes processed depth images as input, and has three output channels - position label images, cosine images, and sine images. The position image represents quality of the grasp point aligned with the tool center point (TCP) of the jaws. The sine image and cosine image indicate the rotation angle of the parallel jaw. The training and validation pipeline of grasp model is similar to the suction model as illustrated in Figure 3.16.

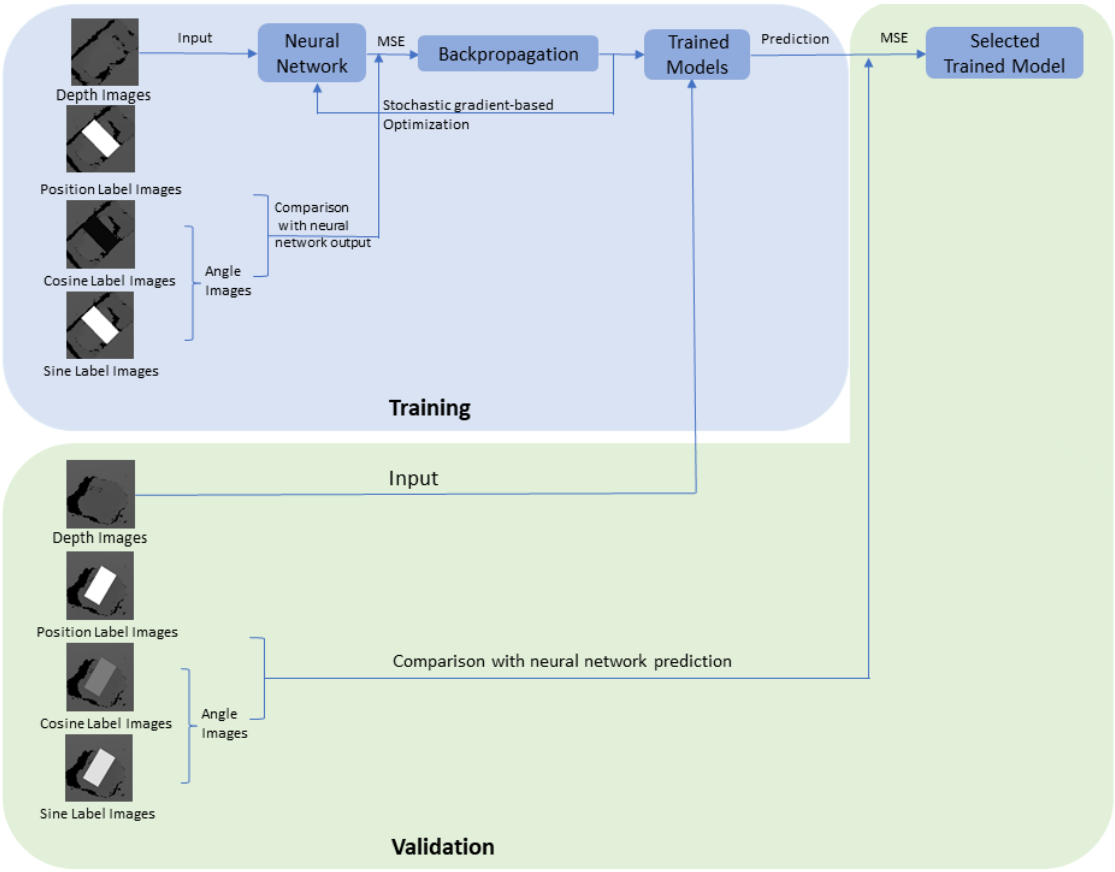


Figure 3.16: Training Pipeline of Grasp Model. Two more channels representing parallel jaw's orientation are added comparing with training of suction model.

The processed depth images are fed to the neural network. The mean square errors (MSEs) are computed from the network estimation and the position and angle ground truth. The gradient of the loss is back propagated to update parameters of the network. Each iteration over the all training data results in a trained model (one iteration is known as an epoch). The trained models, of which the number

depends on the number of epochs, are validated by validation data that is unseen during training stage. The models are trained with 679 training data, using Adam optimizer with a learning rate of 0.001, and validated with 60 validation data. Computing the loss between the trained models estimation and ground truth, the model with minimal validation error is selected.

3.3 TracePen™

TracePen™ [53] is a handheld, no coding device using StreamVR tracking developed by Wandelbots®. It allows human operators without programming skills to quickly and easily teach the robot by moving the trace pen and showing the path. The Tracepen™ system mainly consists of a trace pen and two lighthouse stations as illustrated in the Figure 3.17. The installed lighthouse stations record the position and orientation of the pen tip in the TracePen™ coordinate P. The coordinate might be located around one of the stations, which is unknown to users. Thus, a calibration between TracePen™ coordinate P and robot coordinate R is necessary, in order to make the robot follows the TracePen™ trajectory.

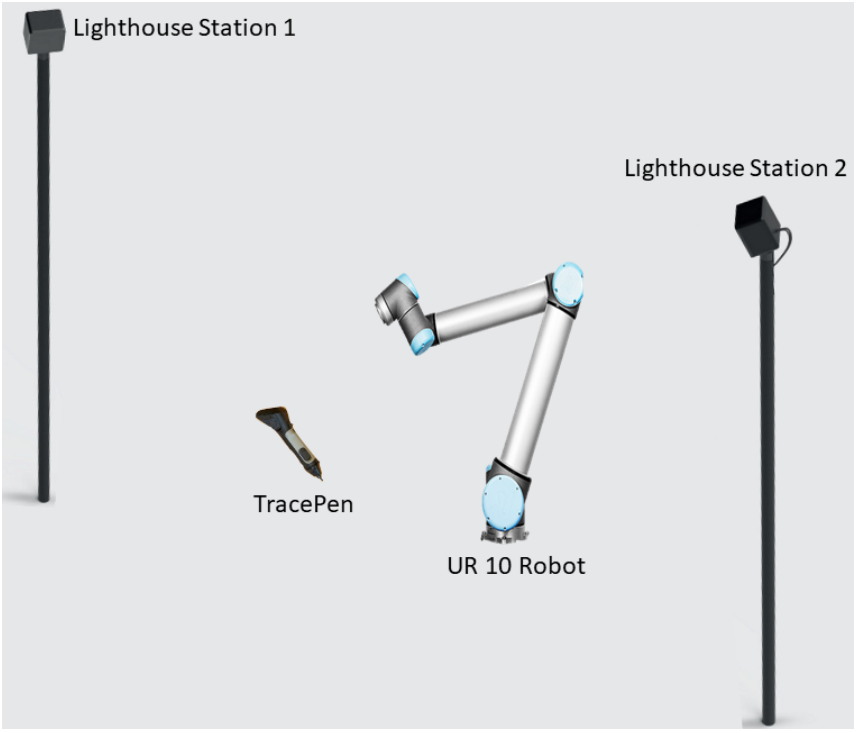
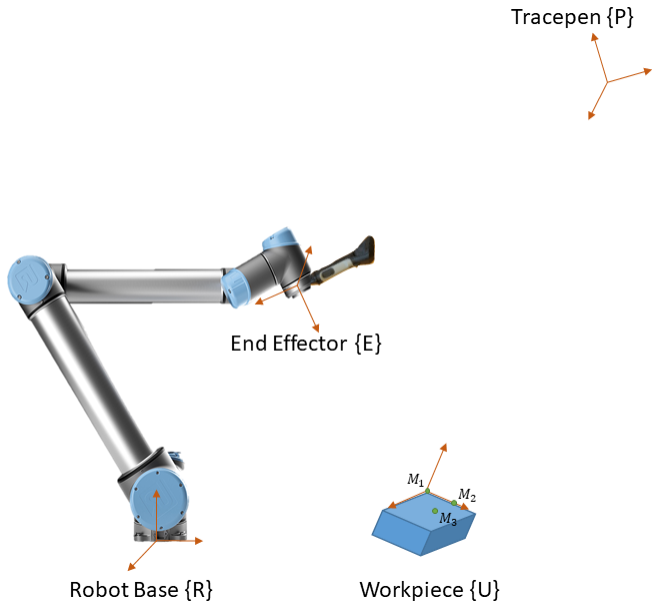


Figure 3.17: A TracePen syatem with UR 10 robot. Two lighthouse stations track pose of the TracePen tip.

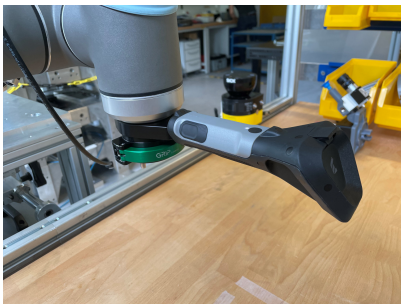
Overall, three coordinates are deployed in the bin-picking system - TracePen™ coordinate P, robot coordinate R, and image coordinate I. The transformations between these three coordinate systems are required. The methods to transform between these coordinates are elucidated in the following.

3.3.1 TracePen™ Calibration in Robot Coordinate

During calibration, the trace pen is inserted to an adapter mounted on the end effector, provided by the Wandelbots® company, as shown in Figure 3.18. With known geometry of the adapter, the robot and trace pen system can record positions of the same point in the two coordinates for calibration.



(a) Robot and TracePen Calibration Overview.



(b) Robot and TracePen Configuration for Calibration.

Figure 3.18: Robot and TracePen Calibration

A three point calibration method [54] is employed to calibrate the orientation between P and R. Based on the calibrated orientation relationship, the translation between P and R can be deduced via a point. A middle workpiece coordinate U is assumed to sample the three points M_1, M_2, M_3. The coordinates of the three

points $U_{mi}, (i = 1, 2, 3)$ in U are:

$$U_{m1} = [0, 0, 0, 1]^{\mathsf{T}} \tag{3.8}$$
$$U_{m2} = [x_{u2}, 0, 0, 1]^{\mathsf{T}} \tag{3.9}$$
$$U_{m3} = [x_{u3}, y_{u3}, 0, 1]^{\mathsf{T}} \tag{3.10}$$

The coordinates of the three points $R_{mi}, (i = 1, 2, 3)$ in R can be obtained from robot states:

$$R_{m1} = [x_{r1}, y_{r1}, z_{r1}, 1]^{\mathsf{T}} \tag{3.11}$$
$$R_{m2} = [x_{r2}, y_{r2}, z_{r2}, 1]^{\mathsf{T}} \tag{3.12}$$
$$R_{m3} = [x_{r3}, y_{r3}, z_{r3}, 1]^{\mathsf{T}} \tag{3.13}$$

Assume homogeneous transformation from workpiece coordinate U to robot coordinate R is ${}^{U}T_{R}$, then

$$U_{mi} = {}^{U}T_{R} \times R_{mi}\,(i = 1, 2, 3) \tag{3.14}$$

Usually a homogeneous transformation matrix representation in roll-pitch-yaw (ϕ, θ, ψ) angles is:

$${}^{U}T_{R} = \begin{bmatrix} \mathrm{c}\phi\mathrm{c}\theta & \mathrm{c}\phi\mathrm{s}\theta\mathrm{s}\psi - \mathrm{s}\phi\mathrm{c}\psi & \mathrm{c}\phi\mathrm{s}\theta\mathrm{c}\psi + \mathrm{s}\phi\mathrm{s}\psi & x \\ \mathrm{s}\phi\mathrm{c}\theta & \mathrm{s}\phi\mathrm{s}\theta\mathrm{s}\psi + \mathrm{c}\phi\mathrm{c}\psi & \mathrm{s}\phi\mathrm{s}\theta\mathrm{c}\psi - \mathrm{c}\phi\mathrm{s}\psi & y \\ -\mathrm{s}\theta & \mathrm{c}\theta\mathrm{s}\psi & \mathrm{c}\theta\mathrm{c}\psi & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ [54]} \tag{3.15}$$

where $x, y, z, \phi, \theta, \psi$ are 6 independent variables to be solved and c is cosine, s stands for sine.

Substitute 3.8, 3.11 and 3.15 to 3.14,

$$\begin{cases} x_{u2}\cos\phi\cos\theta + x = x_{r2} \\ x_{u2}\sin\phi\cos\theta + y = y_{r2} \\ -x_{u2}\sin\theta + z = z_{r2} \end{cases} \text{. [54]} \tag{3.16}$$

Solve 3.16,

$$\begin{aligned} \phi &= \operatorname{atan2}\left(y_{r2} - y, x_{r2} - x\right), \\ \theta &= \operatorname{atan2}\left(z - z_{r2}, \frac{(y_{r2} - y)}{\sin\phi}\right). \end{aligned} \text{ [54]} \tag{3.17}$$

Substitute 3.9, 3.12 and 3.15 to 3.14,

$$\begin{cases} x_{u3}\mathrm{c}\phi\mathrm{c}\theta + y_{u3}(\mathrm{c}\phi\mathrm{s}\theta\mathrm{s}\psi - \mathrm{s}\phi\mathrm{c}\psi) + x = x_{r3} \\ x_{u3}\ \mathrm{s}\phi\mathrm{c}\theta + y_{u3}(\ \mathrm{s}\phi\mathrm{s}\theta\mathrm{s}\psi + \mathrm{c}\phi\mathrm{c}\psi) + y = y_{r3} \\ -x_{u3}\ \mathrm{s}\theta + y_{u3}\mathrm{c}\theta\mathrm{s}\psi + z = z_{r3} \end{cases} \text{ [54]} \tag{3.18}$$

Solve 3.18,

$$\psi = \operatorname{atan2}(M_S, M_C)\,[54] \tag{3.19}$$

where,

$$\begin{aligned} M_S &= \frac{S_1 - M_C \cos\phi}{F_1}, \\ M_C &= \frac{F_2 S_1 - F_1 S_2}{F_2 \cos\phi + F_1 \sin\phi}, \\ F_1 &= \cos^2\theta + \sin\phi \sin\theta, \qquad [54] \\ F_2 &= \cos\phi / \sin\theta, \\ S_1 &= y_{r3} - \cos\theta \cdot z + z_{r3} - y, \\ S_2 &= x_{r3} - \cos\phi \cot\theta\,(z + z_{r3}) - x. \end{aligned} \tag{3.20}$$

Once θ, ϕ, ψ are solved, the rotation matrix ${}^{U}R_R$ from U to R is obtained. In addition, the trace pen system records the position and orientation of the trace pen tip in the trace pen coordinate system P. Similarly as above, the rotation matrix ${}^{P}R_U$from P to U can be computed as well. Thus, the transformation matrix ${}^{P}R_R$ from P to R is calculated,

$${}^{P}R_R = {}^{P}R_U \times {}^{U}R_R \tag{3.21}$$

Next, the translation $[x, y, z, 1]^{\mathsf{T}}$ from P to R can be calculated via a point based on the 3×3 rotation matrix. Given a point in P and in R:

$$\begin{aligned} P_{m4} &= [x_{p4}, y_{p4}, z_{p4}, 1] \\ R_{m4} &= [x_{r4}, y_{r4}, z_{r4}, 1] \end{aligned} \tag{3.22}$$

The relationship between P and R is,

$$R_{mi} = {}^{P}T_R \times P_{mi}\,(i = 1, 2, 3, 4, ...) \tag{3.23}$$

Substitute 3.22 to 3.23,

$$\begin{pmatrix} x_{r4} \\ y_{r4} \\ z_{r4} \\ 1 \end{pmatrix} = \begin{pmatrix} & {}^{P}R_R & & x \\ & & & y \\ & & & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{p4} \\ y_{p4} \\ z_{p4} \\ 1 \end{pmatrix} \tag{3.24}$$

In which, ${}^{P}R_R$ represents a 3×3 transformation matrix.

Solve 3.24,

$$\begin{aligned} x &= x_{r4} - {}^{P}R_R(1, 1:3) \times \begin{bmatrix} x_{p4} & y_{p4} & z_{p4} \end{bmatrix} \\ y &= y_{r4} - {}^{P}R_R(2, 1:3) \times \begin{bmatrix} x_{p4} & y_{p4} & z_{p4} \end{bmatrix} \\ z &= z_{r4} - {}^{P}R_R(3, 1:3) \times \begin{bmatrix} x_{p4} & y_{p4} & z_{p4} \end{bmatrix} \end{aligned} \tag{3.25}$$

So far, the homogeneous transformation matrix ${}^{P}T_{R}$ from trace pen coordinate P to robot coordinate R is solved. The trace pen pose in the robot coordinate system can be then calculated by multiplying the transformation matrix with the trace pen pose in the trace pen coordinate system, as Equation 3.23. Thus, the robot is capable of following the trace pen position given by the operator.

3.3.2 TracePen™ in Image Coordinate

In the thesis, the trace pen touches a suctionable/graspable area over the object that is the input Region of Interest (RoI) to the neural network. In other words, the position of the trace pen tip should be the center of the input image RoI. Therefore, the tracepen pose in robot coordinate attained in the Section 3.3.1 should be transformed to the image coordinate via camera intrinsics and camera pose in the robot coordinate.

The camera intrinsics representation:

$$K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} [23] \tag{3.26}$$

Usually, it is deterministic and given for a camera. In some cases, camera intrinsics calibration using a chessboard is necessary.

The real-time camera pose in the robot coordinate ${}^{C}T_{R}$ can be inquired by the real-time end effector pose ${}^{E}T_{R}$ and fixed transformation from end effector to the camera ${}^{C}T_{E}$ determined by camera installation,

$${}^{C}T_{R} = {}^{C}T_{E} \times {}^{E}T_{R} \tag{3.27}$$

So far, the trace pen position in the image coordinate $P_I = (x, y, 0)$ is calculated from the trace pen position in the robot coordinate P_R as,

$$P_I = \begin{pmatrix} & & & 0 \\ & K & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times {}^{C}T_{R} \times P_R \tag{3.28}$$

Inverse Transformation

After the prediction of neural network, the predicted grasp point $G_I = (x_I, y_I, 0)$ in the image coordinate must be transformed to the robot coordinate $G_R = (x_R, y_R, z_R)$,

so that the robot goes to this position to pick. This transformation can be easily realized by inversing the transformations 3.28 above,

$$G_R = {}^{I}T_R \times G_I \tag{3.29}$$

$${}^{I}T_R = ({}^{R}T_I)^{-1} \tag{3.30}$$

$${}^{R}T_I = \begin{pmatrix} & & & 0 \\ & K & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times {}^{C}T_R \tag{3.31}$$

Due to the fact that the image loses depth information (z value of G_I is 0), z_R is determined by distance between the point on the object and the camera calculated from camera depth stream, and the distance between the camera and gripper tip.

Chapter 4

Implementation

This chapter clarifies conducted experiments to implement methodologies in Chapter 3. Initially, Universal Robot (UR) 10 with a modular gripper is modeled, and the robot motion configuration is considered. Next, a bin-picking pipeline is illustrated in detail. Neural network training using Pytorch is then elaborated. Lastly, camera extrinsic calibration with respect to robot is explained.

4.1 Hardware Setup

The methodologies in Chapter 3 were implemented on a Universal Robot (UR) 10 with a low cost RGB-D camera - Intel® RealSense™ D435i mounted on end effector. A WandelBots TracePen™ was applied to propose a region of interest (ROI) by operator. The neural networks were trained on a server with a Nvidia GTX 2060 graphics card. The robot motion is realized in ROS noetic, and computations of TracePen™ position and suction/grasp point predictions are implemented in ROS foxy.

4.2 Robot Motion Planning

Universal Robot (UR) 10 was applied to conduct robotic bin-picking. The robot motion was controlled via the middle ware - Robot Operating System (ROS). Therefore, building a robot model and configuring robot controlling are required.

4.2.1 Robot Model with Grippers

The UR 10 model in the format of XML Macros (xacro) is provided by the robot company. The xacro model describes dynamic and kinematic properties of the robot. The UR 10 robot has 6 joints as illustrated in Figure 4.1. Each joint angles is limited

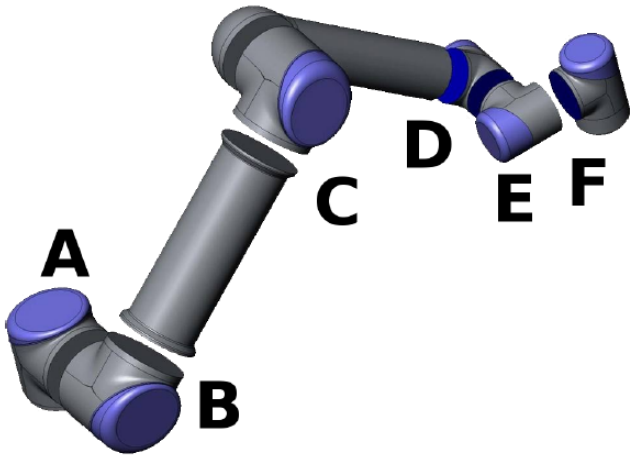


Figure 4.1: 6 Joints of UR 10 robot: A: Base, B: Shoulder, C: Elbow and D, E, F: Wrist 1, 2, 3 [55].

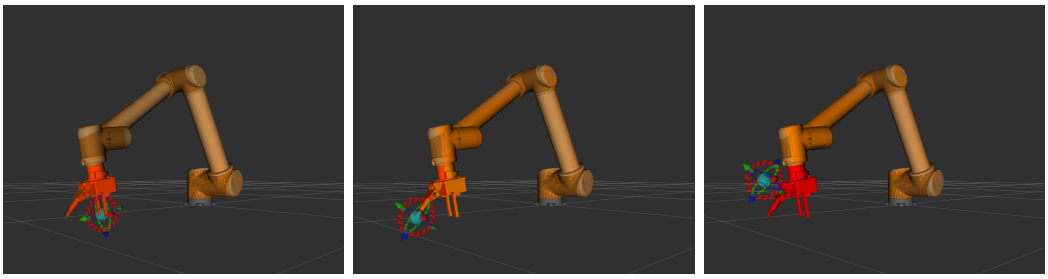
in the range from $-360°$ to $+360°$ [55]. Significantly, from practical experience, it is easier to find an effective kinematic solution if limiting the joint angles, though the default joint limits have overlapped ranges (from $-360°$ to $0°$ is the same as from $0°$ to $360°$ in the joint space). The joint angles should be at least limited in the range of 2π depending on the real application scenario. In the thesis, the robot should move within space on a work table. Therefore, the joint limits were configured as listed in the Table 4.1.

Table 4.1: Joint Limits. The joints are limited based on the robot work range to improve success rate of looking for kinematic solutions.

Joint Name	Limit(in rad)
Shoulder Pan Joint	$(2.094, 2.16)$
Shoulder Lift Joint	$(-\pi, \pi)$
Elbow	$(1.0472, \pi)$
Wrist 1	$(2.9, 2\pi)$
Wrist 2	$(-\pi, \pi)$
Wrist 3	$(-\pi, \pi)$

Additionally, in the bin-picking system, the robot achieves picking and placing with the help of grippers. As reviewed in the Section 2.1, multi-gripper gained popularity in recent years to adopt to different picking tasks. Thus, a modular gripper comprising a parallel jaw and a vacuum gripper with 3 different size (Large: 30 mm, Medium: 10 mm, Small: 6 mm), was employed. The designed grippers model was

connected to the end effector of the robot, constructing a robot with grippers model as visualized in Figure 4.2.



(a) The tool center point (TCP) at center tip of two gripper. jaws.

(b) TCP at tip of the vacuum

(c) TCP at camera surface.

Figure 4.2: UR 10 Robot with a Modular Gripper Model.

4.2.2 Planning Groups

In order to control the robot movement, robot kinematic chains must be defined to plan trajectories from a current state to a desired goal state. Basically, it is an inverse kinematic problem mentioned in Section 2.3.1 to solve joint angles in joint space from given goal pose in the workspace (Cartesian space). Three planning groups were defined for three kinematic chains, as shown in Figure 4.2: 1) base link → center tip of parallel jaw. 2) base link → tip of vacuum gripper. 3) base link → camera. The former two planning groups compute kinematic solutions for goal poses of parallel jaw and vacuum gripper respectively. The last planning group makes capturing sensor data in a desired pose possible. They are named by the author as *tcp*, *tcp_s* (s stands for suction), *tcp_c* (c stands for camera) respectively.

4.2.3 Robot Motion Planning Configuration

The open source software - ***MoveIt*** is adopted to realize robot motion planning as mentioned in Section 2.3.2. ***MoveIt* Setup Assistant** [39] is a user interface to configure the robot to be used with MoveIt, as shown in Figure 4.3. First of all, a robot model in the format of Unified Robot Description Format (URDF) should be loaded, then a **self-collision** matrix that defines pairs of links that are always in collision, impossible to collide, and adjacent links, can be generated based on the model information. These pairs of links are free from collision checking, so it saves processing time of collision check when planning. **Virtual Joints** refer to joints connecting the robot to world, so its parent frame must be *world*, and it is

a fixed joint. **Planning Groups** are essential in motion planning. A set of joints (kinematic chain on a robot) are defined to plan motions and check collisions by kinematic solvers. Multiple planning groups can be defined for different motion chains on a robot. Especially for humanoid robots, each leg and each arm should be a kinematic chain. In the thesis, three planning groups - *tcp*, *tcp_s*, *tcp_c* are defined as described in Section 4.2.2. **Robot Poses** set up poses via joint angles. **End Effector** is defined as a special group, if internal motions happen in the end effector. **Passive Joints** refer to unactuated joints that cannot be controlled. **3D Perception, Simulation, and ROS Control** tabs seem to be setups for sensors in simulation, generate URDF in gazebo simulation, and simulated control respectively, but they are not deployed in this work. After these configurations, a ROS package comprising a set of launch and config files is generated, which is required for motion planning using *MoveIt*.

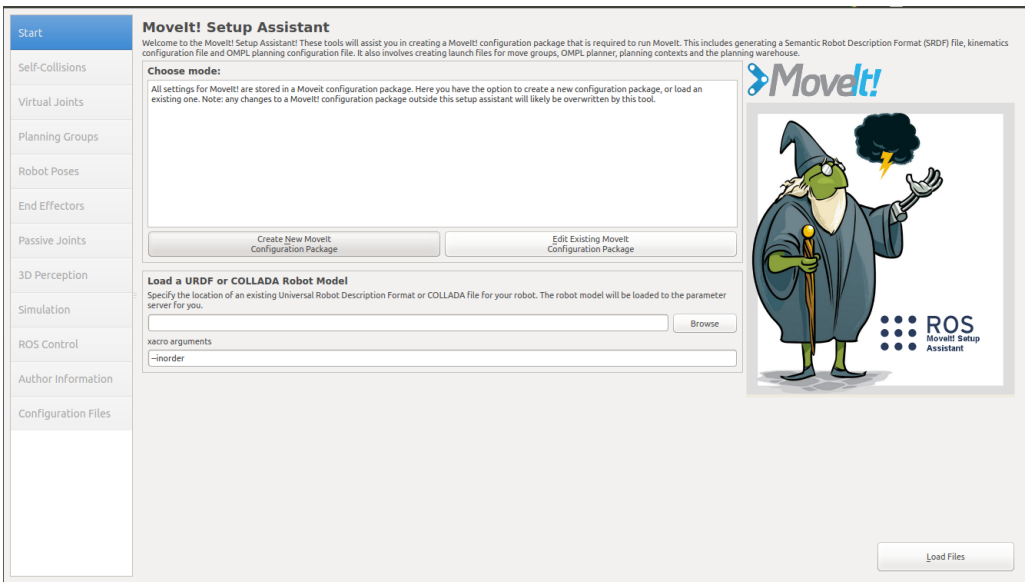


Figure 4.3: Moveit Setup Assistant User Interface [39].

With the configuration package, the robot can be manipulated via MoveIt Python/C++ interface [40], or a ROS Visualization - *RViz* [41] that is a useful debug tool, as visualized in Figure 4.4 . In *RViz*, the *MoveIt* motion planning is loaded as a plugin. Next, the robot must be configured in **Global Options** tab and **Motion Planning** tab within the Display subwindow on the upper left, such as setting up fixed frame, planning scene. In the motion planning plugin subwindow on the bottom left, a planning group should be selected from the previous defined planning groups. Finally, set a goal state, plan the path, and execute. Furthermore, these configurations can also be done via Python/C++ interface in scripts. The desired robot pose can be given by joint angles or desired position and orientation of the end effector.

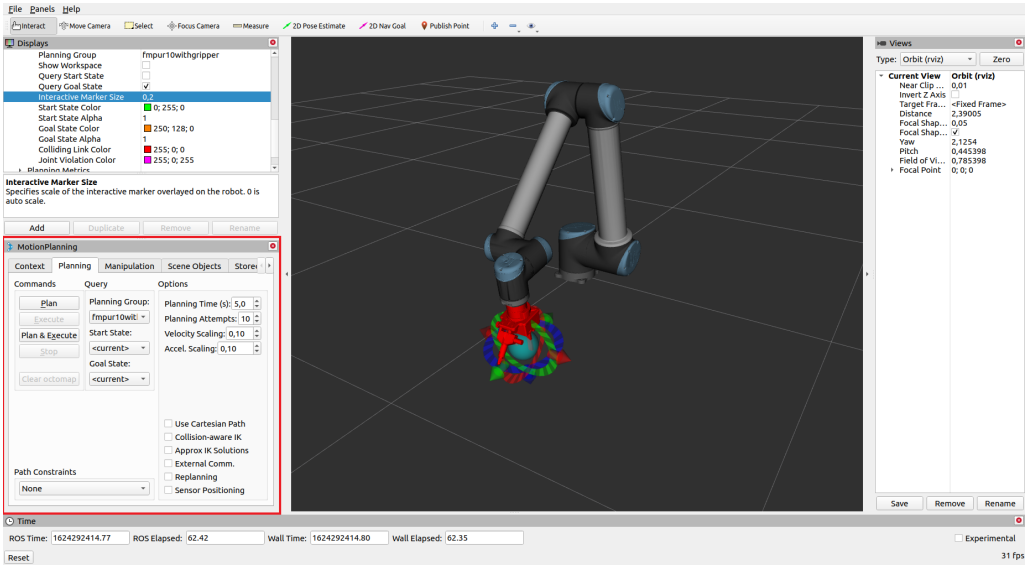


Figure 4.4: RViz. The motion planning plugin is framed in red.

4.3 Bin-picking System

This section firstly elaborates design of a robotic bin-picking pipeline via ROS. The Pytorch implementation for neural network training is then described. Next, gripper manipulation via ROS action is explained. Lastly, camera extrinsic calibration for determining camera pose related to robot is elucidated.

4.3.1 Bin-picking Pipeline

In the bin-picking system, the trace pen was implemented to leverage operator input, and trained neural networks predict suction/grasp points. The trace pen pose was calibrated to the robot coordinate system as stated in the Section 3.3. Figure 4.5 briefly introduces the bin-picking flow. An experienced operator inputs a region of interest (ROI) on the object via a TracePen™. The robot then goes to a pre-grasp pose above the ROI, and camera stream starts to capture visual data. Next, the visual data is fed to a trained model to predict a robot suction/grasp pose. Last but not least, the robot goes to the suction/grasp pose, activating the modular gripper to pick up the object.

The bin-picking pipeline is more specifically illustrated in the Figure 4.6. In total three main nodes are employed. The node Action Manager for robot movement is in ROS 1 noetic, because stability is the most significant for robot motion in the real world. Other nodes for computation are in ROS 2 foxy, due to the fact that ROS 2 provides better real-time capability. A ROS bridge is necessary to connect ROS 1 and ROS 2. Initially, an operator touches the object surface with the trace pen,

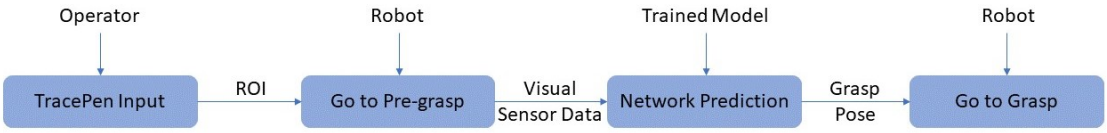


Figure 4.5: Bin-picking Flow.

leveraging operator's expert knowledge to provide a potential suctionable/graspable region. The trace pen position in robot coordinate (published as $\backslash tracepeninR$ topic) is acquired by the trace pen calibration matrix ${}^{P}T_{R}$. The robot with a camera then goes to around 30 mm above the trace pen position, which is pre-grasp position. The planning group, introduced in the Section 4.2.1, is *tcp_c* in this stage. In this pre-grasp position, the camera (RGB-D RealSense™) starts depth streams and captures an depth image over the object. Besides, the trace pen position is also transformed to the image coordinate via camera intrinsics and camera pose based on robot coordinate, as elucidated in Section 3.3.2. The captured depth image in the pre-grasp pose is center cropped according to the trace pen position in the image. The cropped depth images in the format of array are then fed to the trained pytorch neural network models. Subsequently, the suction model predicts a suction point $S_I = (x_I, y_I, 0)$ in the image. In addition, the grasp model predicts a grasp pose $G_I = (x_I, y_I, 0, \theta)$. Comparing pixel values of the suction point and the grasp point, the point with larger pixel value is proposed as a robot picking point. The points (x_I, y_I) of S_I, G_I in the image coordinate must be again transformed back to the robot coordinate S_R, G_R, as described in the Section 3.3.2. The z value of S_R, G_R is calculated from distance (queried from the camera depth frame) between the camera and the predicted grasp point, and the length from camera to *tcp* or *tcp_s*. Meanwhile, the Action Manager subscribes the topic $\backslash grasppointinR$ to move the gripper (*tcp* or *tcp_s*) to the grasp position. Next, the gripper action is triggered to activate vacuum gripper or the parallel jaw to pick. Finally, the robot lifts up, moves to a placing position, goes downwards to place the object.

4.3.2 Pytorch Implementation for Network Training

The neural network training was realized by Pytorch - an open source machine learning framework [55]. Figure 4.7 demonstrates the Pytorch implementation pipeline. There are four main classes. The *Dataset* is a customized dataset class to generate a dataset with input images and label images. The *Model* class defines architecture of the neural network, such as type and number of layers, type of activation function,

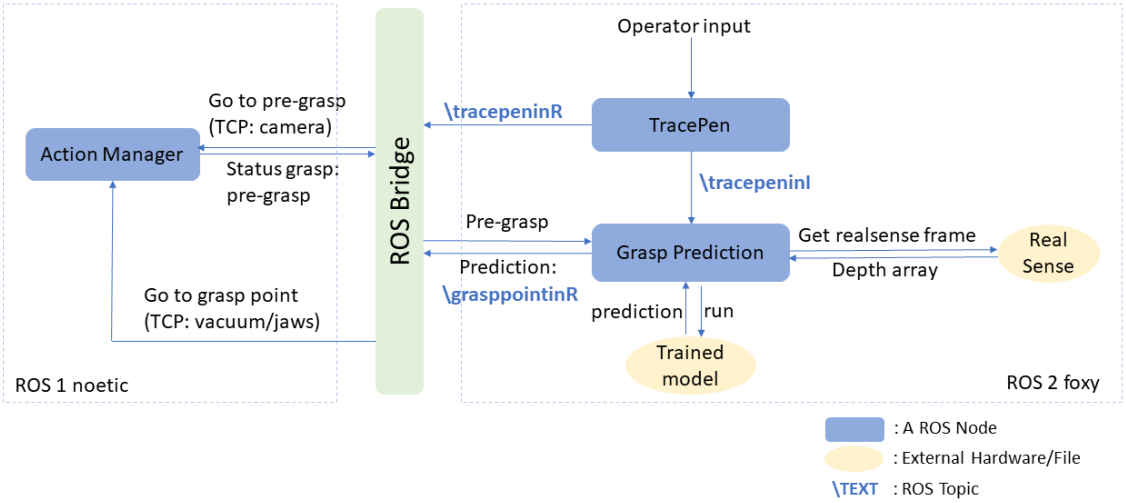


Figure 4.6: Bin-picking Pipeline. Operator input via trace pen → Go to pre-grasp pose → Capture depth image, run model to predict grasp position → Go to grasp position → Pick → Place.

kernel size. In the *train* class, the loaded model is trained with the loaded dataset with an optimizer (e.g. Adam, Stochastic Gradient Decent). Next, validation data is processed by trained models. Mean Sqaure Error (MSE) is again computed between predictions from trained model and label images to select a model. Finally, the model is run in *Run_model* class to process any test data and estimate grasp points. Notably, Each class has two versions - one for suction estimation, and another one for grasp prediction, though the responsibilities of every class are the same.

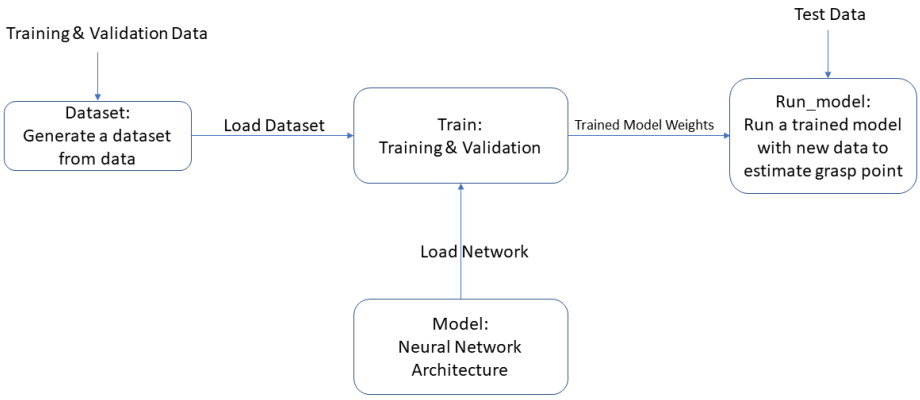


Figure 4.7: Pytorch Implementation Pipeline.

4.3.3 Gripper Manipulation

The modular gripper was manipulated by a ROS Action. The Action Manager node illustrated in the Figure 4.6 sends gripping goals listed in the Table 4.2 as a client. The gripper server node executes the action, sends feedback and result back. Table 4.2 presents five defined messages for manipulating parallel jaw, and two messages for activating the vacuum gripper. InsideGrip represents that the two jaws start with closing, and then open. In contrast, OutsideGrip means the two jaws start with opening, and then close. The other three parameters define the speed, force, and distance of jaws motion. Moreover, ActivateModule message activates the vacuum, and ModuleStrength defines how much suction the vacuum should be.

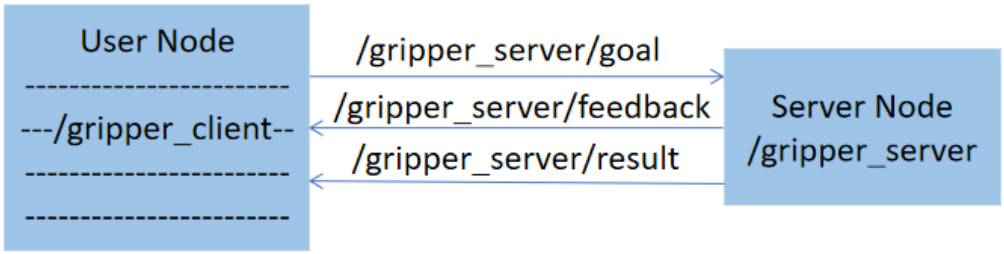


Figure 4.8: Gripper Action Flow.

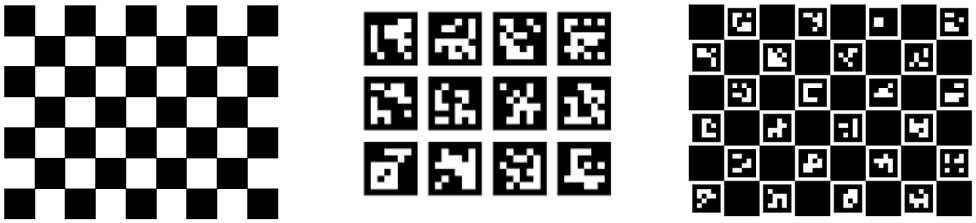
Table 4.2: Action Goal

Message Name	Datatype	Range	Unit	Comment
InsideGrip	bool	0:off, 1:on	n.a.	Parallel Jaw
OutsideGrip	bool	0:off, 1:on	n.a.	Parallel Jaw
Speed	Int8	2 - 120	mm/s	Parallel Jaw
Force	Int8	1 - 10	N	Parallel Jaw
Distance	Int8	0 - 40	mm	Parallel Jaw
ActivateModule	bool	0:off, 1:on	n.a.	Vacuum Gripper
ModuleStrength	UInt8	0 - 255	n.a.	Vacuum Gripper

4.3.4 Camera Extrinsic Calibration

A lost-cost 3D camera - RealSense™D435i - was mounted on the robot (eye-in-hand), as visualized in Figure 3.1. Thus, the camera pose relative to the robot must be figured out. *MoveIt* developed a Hand-Eye Calibration Plugin [56] to calibrate eye-in-hand and eye-to-hand extrinsics. Calibration is performed by detecting corners of chessboard, ArUco marker, or ChArUco marker, as shown in Figure 4.9. Using

ArUco allows occlusion or partial views, so it is more versatile than chessbord patterns [57]. However, ChArUco marker combines chessboard and ArUco marker, and provides much more accurate corners comparing with the ArUco marker corners [57].



(a) 9 × 7 Chessboard Example.

(b) 3 × 4 ArUco Marker Example.

(c) 8 × 6 ChArUco Marker Example.

Figure 4.9: Patterns for Calibration. Chessboard + ArUco Marker → ChArUco Marker [57].

In order to obtain a precise calibration result, samples must be from different views. A promising strategy is as follows:

1. Rotate around x, y, z axes respectively, in a direction (e.g. clockwise).

2. Rotate around $x - y, y - z, x - z$ axes respectively, in a direction.

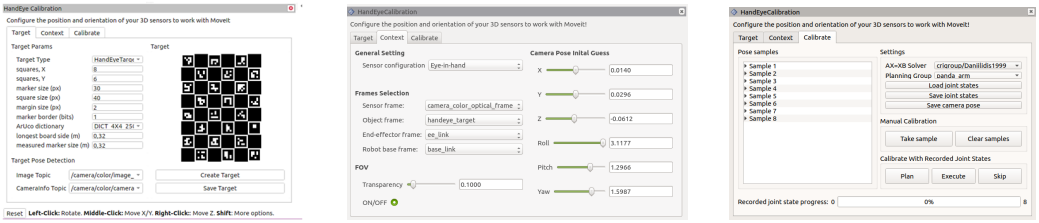
3. Rotate around $x - y - z$ axis, in a direction.

4. Repeat step 1, 2, 3, in the opposite direction (e.g. anti-clockwise).

5. Repeat step 1, 2, 3, 4 to gain more samples if necessary.

Figure 4.10 shows the interface of the Hand-Eye Plugin. In the toolbar **Target**, parameters of the ArUco or ChArUco marker should be defined, and the Camera Image topic and Camera Info topic must be selected. Next, in the toolbar **Context**, sensor configuration (eye-in-hand or eye-to-hand) and various frames are set up. Finally, in the toolbar **Calibrate**, samples are recorded and then camera pose is

solved. The number of samples should be at least 12 - 15 sampled with the strategy.



(a) Target. (b) Context. (c) Calibration.

Figure 4.10: MoveIt Hand-Eye Calibration Plugin [56].

Chapter 5

Evaluation

Chapter 5 evaluates the robot grasping point estimation framework proposed in the master thesis. Firstly, training results of neural networks for suction prediction and grasp prediction are illustrated respectively. The selected trained models are then tested with more test data to evaluate performances. Lastly, the Tracepen™ calibration precision is assessed briefly and qualitatively.

5.1 Training Results

The neural networks were trained for suction prediction and grasp prediction respectively. The training and validation of suction model cost around 4 minutes. Figure 5.1 visualizes the training loss and validation loss for suction model with 70 epochs. The training loss decreases from 0.01355 at beginning to 0.0018341 at epoch 50. In the first 10 epochs, the loss decreases dramatically, and oscillates afterwards. The validation loss is also reduced from 0.3962 at the first epoch to 0.1536 at the last epoch. Although the training loss does not obviously decline after around 30 epochs, the validation loss keeps dropping off. It is possible that the validation loss could be even smaller if more training data would be available.

The training losses and validation losses for grasp model consist of position loss, sine loss, and cosine loss, which represent losses of position images, sine images, and cosine images respectively as the names indicate. Figure 5.2 presents the training loss and validation loss that are sums of the three losses with 45 epochs. The training loss diminishes from 0.2455 to about 0.1272 in the last several epochs. However, the validation loss is much larger, starts from 5.9 and finally oscillates at around 4.5. It indicates that the model is not trained well enough to process new data. Figure 5.3 and Figure 5.4 show specific position loss, sine loss, and cosine loss of training and validation respectively. The position training loss descends from 0.07013 to 0.02194 at epoch 42. The position validation loss falls from 1.924 to 0.7576. The sine loss and cosine loss of training go down from 0.07936 and 0.09605 to 0.05173 and 0.05223 respectively, whereas the sine loss and cosine loss of validation fluctuate around 1.7

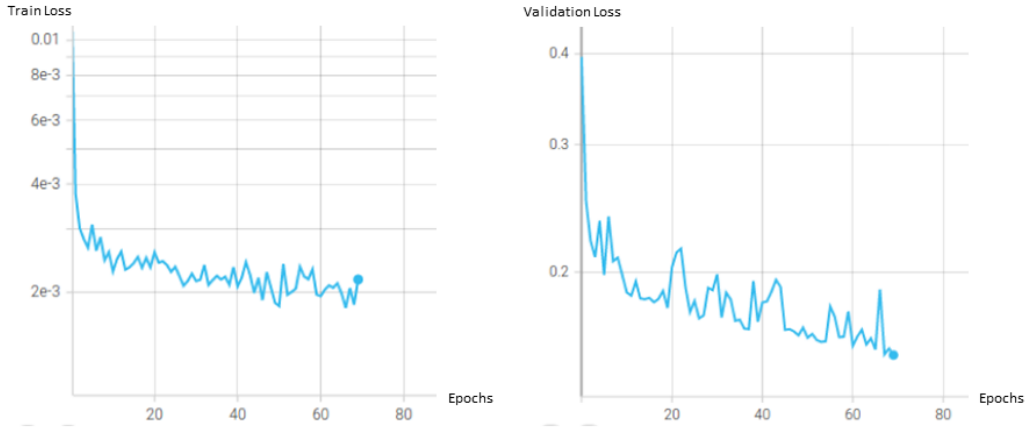


Figure 5.1: Training Loss & Validation Loss of Suction Model for Usage of the Vacuum Gripper. The training loss decreases dramatically, and then oscillates. The validation loss keeps falling.

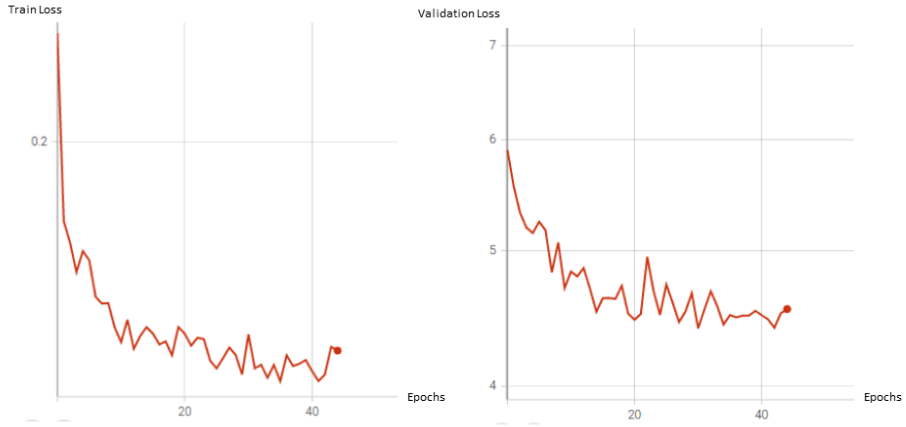


Figure 5.2: Training Loss & Validation Loss of Grasp Model for Usage of Parallel-jaw. The training loss reduces to around 0.1272, whereas the validation loss is obviously large, oscillating at 4.5.

and 2. Thus, the position prediction still needs improvement, and the angle prediction is not robust to new data at all. The main reason could be: the grasping data size is too small (only 739 data as introduced in Section 3.2.2) to train a model with good performances. Besides, the angle estimation is more difficult than position estimation, because the sine and cosine images are of great difference at every single angle, which leads to angle prediction training requires even much more data than position prediction training. Therefore, further evaluation is conducted on the

vacuum suction model only.

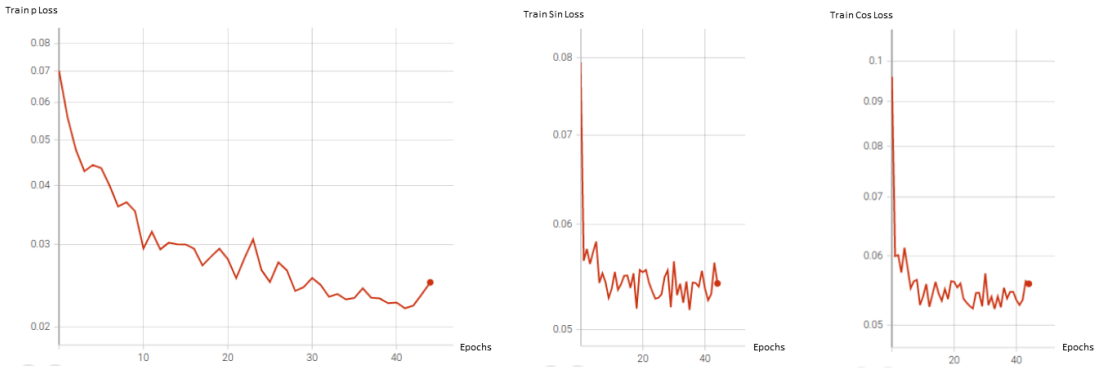


Figure 5.3: Grasp Training Losses - Position Loss (left), Sine Loss (middle), Cosine Loss (right).

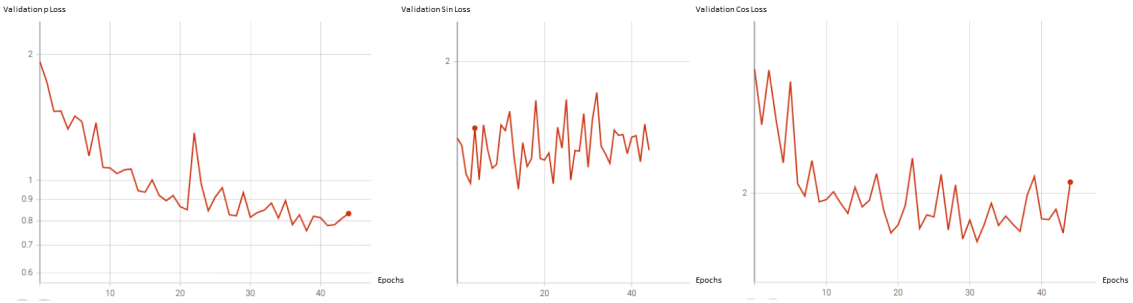


Figure 5.4: Grasp Validation Losses - Position Loss (left), Sine Loss (middle), Cosine Loss (right).

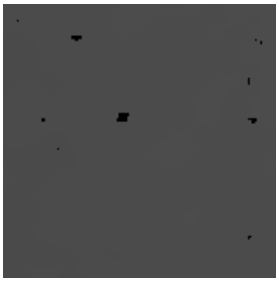
5.2 Test Results

A trained model at epoch 50 is selected as final suction model from in total 70 models, because it has relatively low training loss and also low validation loss. In order to further test performance of the selected model, test data from captured dataset in Section 3.1.1 is fed to the trained neural network to propose suction points. Examples of input test data is shown in Figure 5.5. The outputs from model prediction are still images that indicate suction quality. The output suction quality images are then evaluated to find a point with best suction quality. The point with maximal pixel value is finally proposed as the suction point, because the suction

quality is represented by the pixel value. Figure 5.6 presents the corresponding prediction results. A suction proposal is regarded as success, if the predicted suction point is within area of the label. Table 5.1 summarizes success rates of suction point prediction for test data of five parts from dataset. Average success rate accounts for 85.65%, calculating by number of success predictions divided by total number of test data as listed in Table 3.1.

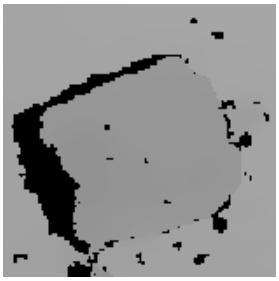
(a) Example of Test Data: Housing.

(b) Example of Test Data: Baseplate.



(c) Example of Test Data: Gearwheel.

(d) Example of Test Data: Support Gearwheel.

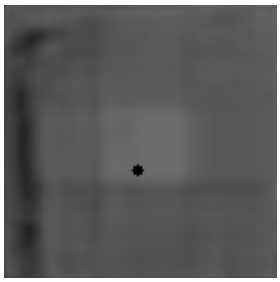


(e) Example of Test Data: Motor.

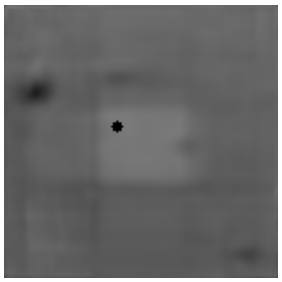
Figure 5.5: Examples of Input Test Data.

Table 5.1: Suction Success Rate of Test Data from Dataset.

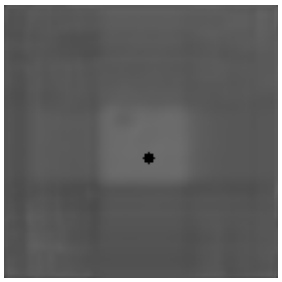
Part Name	Success Rate
Housing	81%
Gearwheel	91.4%
Support Gearwheel	100%
Baseplate	79.6%
Motor	85%
Average	85.65%



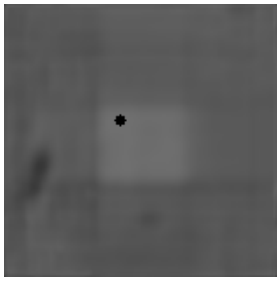
(a) Example of Test Result: Housing.



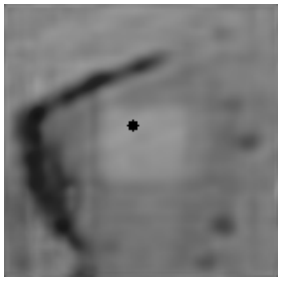
(b) Example of Test Result: Baseplate.



(c) Example of Test Result: Gearwheel.



(d) Example of Test Result: Support Gearwheel.



(e) Example of Test Result: Motor.

Figure 5.6: Examples of Test Results. The dots represent predicted suction points.

Moreover, more test data is captured via a depth camera and Tracepen™ when the robot system is running. As the bin-picking pipeline elaborated in Section 4.3.1, the calibrated trace pen firstly pointed a suctionable area on the object based on experiences of the operator, then the robot goes to a pre-grasp pose above the suggested suction area and captures depth images. Samples from depth stream according to trace pen input are visualized in Figure 5.7. Subsequently, the trained model is run to process depth frames, and finally predicts suction points.

Figure 5.8 shows model prediction results from fed test data in Figure 5.7. The red dots in the suction quality images are estimated suction points. The predicted suction points for gearwheel, and housing are located within the label area on the object plane surface, whereas one of 45 predictions for motor is obviously out of the object and eight predictions are close to the object edge and lie out of label area. The prediction for motor is not robust enough because its size is small comparing with other parts. Moreover, seven of 23 support gearwheel predictions are out of the label area. Lastly, only three of 18 predictions for base plate are successful, but others are far deviated from human desired suction areas. A potential reason could be that only straight edges of the base plate is contained in training data, as presented in Figure 5.5 (b), while captured data for testing in Figure 5.7 mainly

includes a curved edge. The suction prediction success rates for the five parts are summarized in Table 5.2. Average success rate accounts for 79.7%.

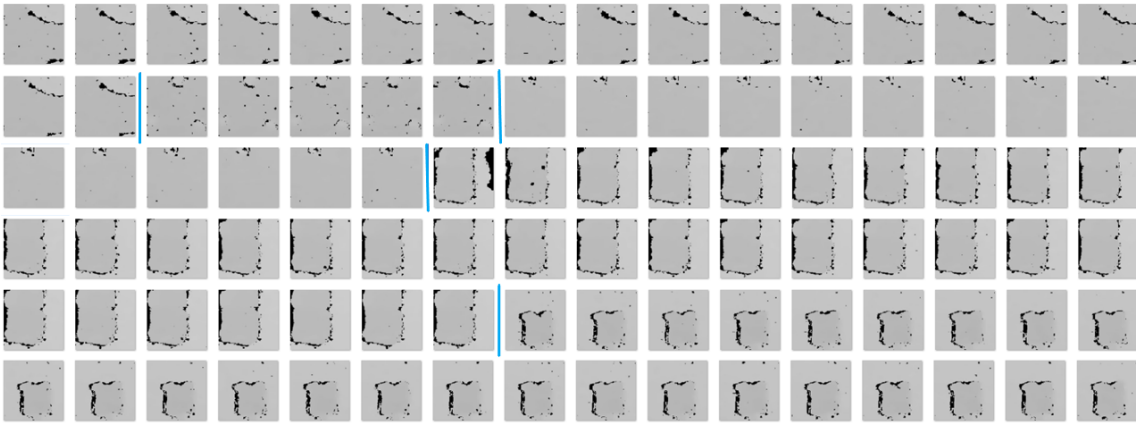


Figure 5.7: Test Data captured via the Bin-picking System. In order from beginning to end - Baseplate, Support Gearwheel, Gearwheel, Housing, Motor, separated by blue lines.

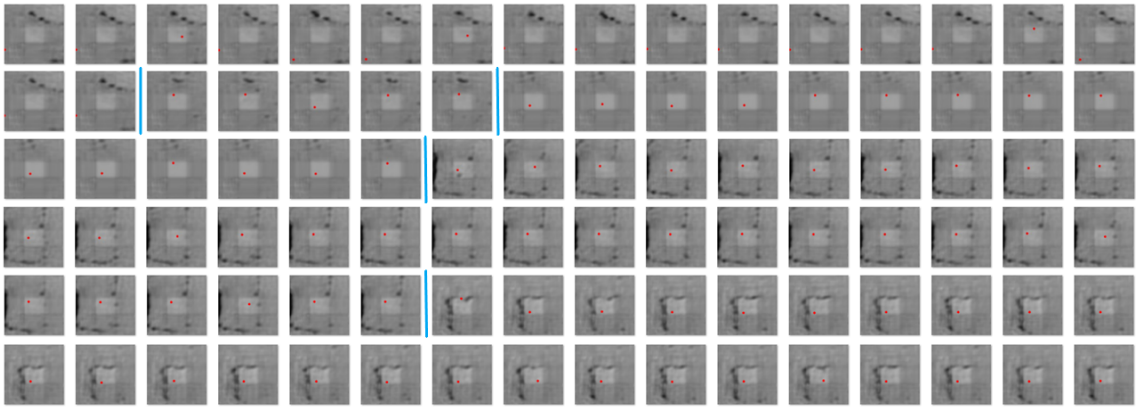


Figure 5.8: Test Results. Prediction from the trained model. The red dots represent predicted suction points.

Comparing Table 5.1 with Table 5.2, prediction success rate for support gearwheel is 100% whe testing with data from dataset, whereas it reduced dramatically to 70% with data captured from bin-picking system. Considering training data amount of support gearwheel is minimal (only 122 data listed in Table 3.1), it indicates

that prediction for this part could be overfitting. More training data is available for housing and gearwheel, so their predictions show better robustness with unseen data.

Table 5.2: Suction Success Rate of Test Data captured via Robot Bin-picking System.

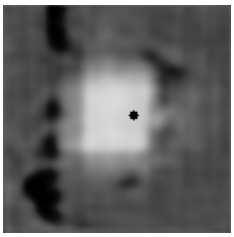
Part Name	Success Rate
Housing	100%
Gearwheel	100%
Support Gearwheel	70%
Baseplate	16.7%
Motor	80%
Average	79.7%

In summary, overall success rate of test data in dataset and test data captured by bin-picking system comprises 82.24%. Small data volume leads to overfitting, i.e. prediction for new data is much worse that that for data in dataset. On the contrary, more training data improves robustness. Notably, the prediction success rate drops drastically if new features that is not included in training data appears in test data.

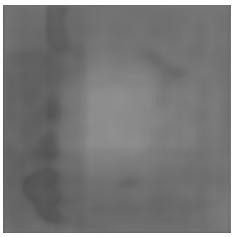
In addition, a grasp prediction model is also selected, though the training performance requires improvement. The trained model takes grasp test data as visualized in Figure 5.9 as input, and outputs grasp position quality image, angle images (sine image and cosine image clarify an angular value.) as presented in Figure 5.10. The grasp position estimation is correct, while sine and cosine estimation does not perform well. It is of difficulty to deduce the orientation angle from the sine and cosine prediction outputs. The main reason is that data amount for grasping training is too small (739 data in total as summarized in Section 3.1.2).

(a) Housing. (b) Gearwheel. (c) Motor.

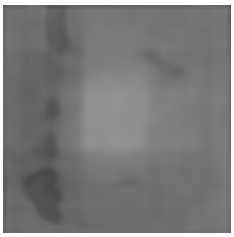
Figure 5.9: Examples of Test Data for Grasping Model.



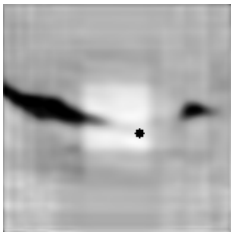
(a) Housing Position Image.



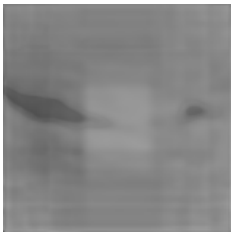
(b) Housing Sine Image.



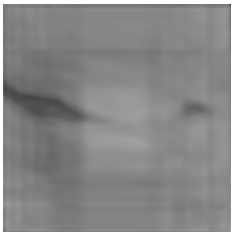
(c) Housing Cosine Image.



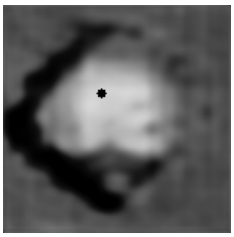
(d) Gearwheel Position Image.



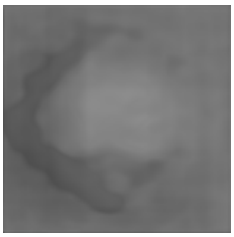
(e) Gearwheel Sine Image.



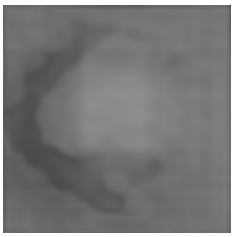
(f) Gearwheel Cosine Image.



(g) Motor Position Image.



(h) Motor Sine Image.



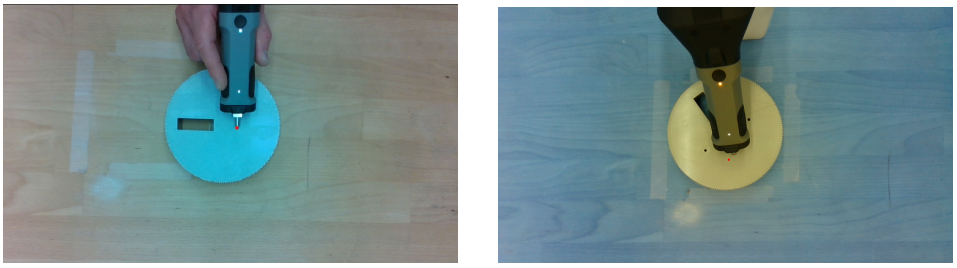
(i) Motor Cosine Image.

Figure 5.10: Examples of Test Data for Grasping Model. The grasp position prediction proposes grasp candidates, while the rotation angles cannot be estimated by output sine and cosine images.

5.3 TracePen™

The TracePen™ system works by tracking the trace pen. Notably, the recorded pen pose could be affected by environment. For example, if many irrelevant objects especially reflective ones are in the perceptual domain of stations. Moreover, the precision of recorded trace pen pose is the highest if the pen is located in the center of station's perceptual domain. Otherwise, the recorded poses are imprecise. It also influences the trace pen calibration elaborated in Section 3.3, because the calibration matrix was calculated based on three recorded points. As a result, the calibrated trace pen position in robot coordinate is 100% precise in some area on the work table, as presented in Figure 5.11 (a). The point transformed to image coordinate

(elaborated in Section 3.3.2) is exactly on the pen tip. However, the pen tip position in image can be deviated as visualized in Figure 5.11 (b). Notably, the two stations should be installed in the way that included angle between them is larger than 120°.



(a) Precise Transformed Tracepen Position.

(b) Deviated Transformed Tracepen Position.

Figure 5.11: Calibrated Tracepen Tip Position. The red dot is position of pen tip transformed to robot coordinate and finally to image coordinate.

Chapter 6

Discussion

This chapter discusses notable points found during the work of thesis - data size, label design, gripper switch, data collection view, data distribution, distance measurement.

In the thesis, the performance of training the neural network is very restricted by the data amount. The industrial robotic grasp dataset generated during the thesis is small comparing with usual data volume for deep learning approaches. The main reason is that capturing real-world data with robot, and manually label cost much time. Especially for the grasp model training, data size is far fewer than enough, due to time limitation. An attempted but not made idea to enlarge the dataset is that generating synthetic data, for instance, using blender. It is a usual way to generate dataset in published robotic bin-picking approaches. Nevertheless, author's knowledge about computer aided design software is limited and automatically generating depth images using Blender is more difficult than handling color images. Thus, this idea has not been realized yet, but it is feasible.

In addition, the label shape of suction is designed as circular, because suction pads are round. Nonetheless, according to prediction results, the circular shape dose not play a part, because predicted suctionable areas are always square-shaped. To the knowledge of author, the reason should be the kernel of convolution is usually a square slice, e.g. 3×3, 5×5, 7×7, 11×11. Consequently, the input image is processed in small square units during operations in the neural network. In a circular edge unit containing both a suctionable area and an unsuctionable area, a main feature is extracted to represent this unit.

Moreover, expert knowledge about how to pick up the parts and which gripper is better to pick, is necessary. Especially the gripper switching strategy (comparing predicted suction quality and predicted grasp quality, choosing the gripper with predicted higher quality) is highly dependent on the labeled ground truth. Usually, an object with a flat surface is better to be suctioned by the vacuum gripper. In

contrast, parallel-jaw is expert in grasping bumpy objects. However, which gripper is more suitable to pick which object is still uncertain, which requires more bin-picking experiments. Accordingly, in the thesis, the suction and grasp position labels are set to the same pixel values that represent the same gripping quality. If expert knowledge about gripper switch strategy is available, pixel values of labels can be set up to different levels to stand for different gripping quality.

Furthermore, an advantage of collected data for this thesis, compared to data used in other published bin-pick approaches, is that the object is scanned from various poses via camera moving with robot. On the contrary, data in other methods are captured in a fixed pose, usually from above view, for instance, Dex-Net 2.0 [17]. Therefore, the trained models over data collected from different views are not so sensitive to the camera installation pose of eye-to-hand configuration, and camera movement pose of eye-in-hand configuration.

Additionally, data distribution regarding objects is also a point worth considering. When collecting data for the thesis, more data is captured for objects with more different faces to show the objects' structure completely. Consequently, data amount for each object is not even, and depends on complexity of the object structure. Another possibility is that making data distribution for each object even, so that performance of grasping point prediction is not of much difference for each object.

Besides, the z value of grasping pose in bin-picking pipeline that represents distance to trace pen point on the object is mainly obtained from depth camera stream. When the trace pen point in the image coordinate is exactly on the pen tip, the distance from depth camera might be not very precise, due to occlusion of the pen upper part. Thus, a more stable way to measure distance to the object is necessary.

Lastly, it is possible that size of industrial parts is tiny, for example, the largest side of the part is smaller than 35mm. In this case, it is of difficulty to capture its depth image via a depth camera (e.g. RealSense™D435i in this thesis). Therefore, the author would suggest considering other data formats fed to network training like mesh and point cloud, because they could present more information of the parts.

Chapter 7

Conclusion

In summary, the thesis introduces background and requirements of the topic, and then reviews state of the art about robot end effector design, robotic grasping pose estimation approaches mainly using deep learning technique, and robot actuation methods. Significantly, methodologies of the thesis are elaborated in detail - industrial bin-picking dataset collection and generation methods, neural networks training, and usage of a Tracepen™ system with robot. Next, implementation of robot motion and bin-picking system are elucidated. Finally, performances of trained neural networks and Tracepen™ input precision are evaluated, and some thoughts inspired through the thesis work are discussed.

The work of the master thesis is concluded as follows:

1. An elliptic robot trajectory for automatic data collection from various views.

2. Dataset generation.

3. Investigation of neural networks' architecture, training and validation of fully convolutional neural networks via Pytorch over the dataset.

4. Integration of the Tracepen™ system into the robot system.

5. Robot configuration for using Robot Operating System (ROS).

6. High-level robot control through ROS.

7. A bin-picking system constructed in ROS.

8. Camera configuration.

The results show that a robotic grasp framework based on fully convolutional neural networks (FCN) provides a feasible solution. It is verified that FCN architecture

saves training and running time. Fast speed is also a highlight of FCN compared to normal convolutional neural network (CNN).

In the future, the dataset should be enlarged to support network training by capturing more real-world data and generating synthetic data. An automatic labeling tool is a good idea to speed up the data ground truth labeling. Besides, a more robust distance measurement method is necessary for both bin-picking execution and data automatic labeling. Furthermore, other neural network architectures such as fully convolutional ResNet 101 [14], GQ-CNN [17] fully convolutional GQ-CNN [10], are worth investigating. Lastly, other input data formats are of consideration, to improve robotic bin-picking system's capability of dealing with tiny parts.

Appendix A

Appendix

A.1 Automatic Labeling

It is possible to develop an automatic labeling tool to label all images captured from one dome trajectory by once manual label, because in one recording the object is static and labels of captured different images should be the same point on the object in real world. The idea is illustrated in Figure A.1. Camera intrinsic K is available, and homogeneous transformation matrix T_i in representation of camera poses are recorded at each step. Manually labeling one image, so a pixel coordinate of the label $L_1 = (x_1, y_1)$ is obtained. Origin of pixel coordinate $(0, 0)$ is 2D located on the upper left of the image. Camera coordinate C usually lies in the camera in the 3D workspace, so in this case it is a moving coordinate system. Origin of the robot base coordinate R is the fixed robot base.

The labeled 2D pixel coordinate L_1 with its depth value z_1 multiplies with inverse of homogeneous camera intrinsic matrix (projection matrix) P^{-1} leads to a 3D position in the camera coordinate L_{1C},

$$L_{1C} = P^{-1} \cdot (x_1, y_1, z_1)^T \tag{A.1}$$

The position L_{1C} is then transformed to robot base coordinate R by multiplying inverse of the recorded camera pose T_1^{-1} in this step,

$$L_{1R} = T_1^{-1} \cdot L_{1C} \tag{A.2}$$

To label a new image based on the labeled pixel coordinate, the label point in static robot base coordinate L_{1R} should multiply a new camera pose T_2 recorded when capturing the new image, so the label position L_{2C}in a new camera coordinate is obtained,

$$L_{2C} = T_2 \cdot L_{1R} \tag{A.3}$$

Lastly, multiplying that position with camera intrinsic P, the label point in the

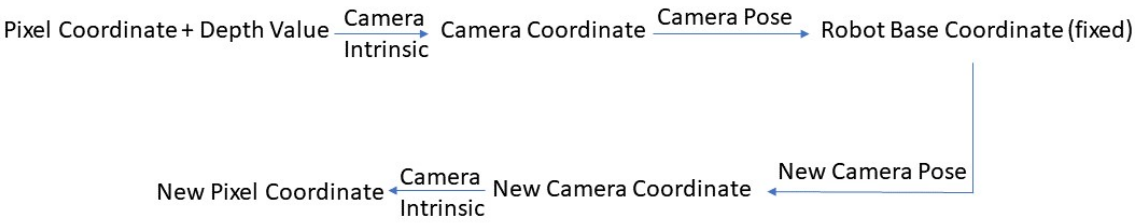


Figure A.1: Automatic Generation of Labels.

space is marked in the new image,

$$L_2 = P \cdot L_{2C} \tag{A.4}$$

Unfortunately, this idea has not been realized in this thesis yet, because the depth values of pixel coordinates are not saved when recording the dataset.

List of Figures

Bibliography

[1] A. Noda G.a Ricardez T. Nagatani A. Zeng M. Fujita, Y. Domae and A. Causo I. Chen T. Ogasawara S. Song, A. Rodriguez. What are the important technologies for bin picking? technology analysis of robots in competitions based on a set of performance metrics. *Advanced Robotics*, 34:560 – 574, 2020.

[2] M. Laskey A. Tanwani R. Berenstein P. Baskaran S. Iba J. Canny K. Goldberg D. Seita, N. Jamali. Deep transfer learning of pick points on fabric for robot bed-making, 2019.

[3] F. Moreno-Noguer C. Torras A. Ramisa, G. AlenyÃ . Using depth and appearance features for informed robot grasping of highly wrinkled clothes. In *2012 IEEE International Conference on Robotics and Automation*, pages 1703–1708, 2012.

[4] M. Waechter-T. Asfour P. Schmidt, N. Vahrenkamp. Grasping of unknown objects using deep convolutional neural networks based on depth images. pages 6831–6838, 05 2018.

[5] J. Leitner D. Morrison, P. Corke. Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach. *CoRR*, abs/1804.05172, 2018.

[6] Anis Sahbani, S. El-Khoury, and Philippe Bidaud. An overview of 3d object grasp synthesis algorithms. *Robotics and Autonomous Systems*, 60:326–336, 03 2012.

[7] A. Moreira J. Cunha P. Oliveira J. Carvalho de Souza, L. Rocha. Robotic grasping: from wrench space heuristics to deep learning policies. *Robotics and Computer-Integrated Manufacturing*, 71, 04 2021.

[8] S. Al-Zawi S. Albawi, T. Mohammed. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.

[9] W. Kraus K. Kleeberger, R. Bormann. A survey on learning-based robotic grasping. *Current Robotics Reports*, 1:239–249, 2020.

[10] K. Goldberg V. Satish, J. Mahler. On-policy dataset synthesis for learning robot grasping policies using fully convolutional deep networks. *IEEE Robotics and Automation Letters*, 4(2):1357–1364, 2019.

[11] Y. Lin and P. Isola-T. Lin A. Zeng, S. Song. Learning to see before learning to act: Visual pre-training for manipulation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7286–7293, 2020.

[12] R. Kawanishi-G. Ricardez K. Kato K. Shiratsuchi R. Haraguchi R. Araki H. Fujiyoshi S. Akizuki M. Hashimoto A. Causo A. Noda H. Okuda T. Ogasawara M. Fujita, Y. Domae. Bin-picking robot using a multi-gripper switching strategy based on object sparseness. In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pages 1540–1547, 2019.

[13] T. Tang-M. Tomizuka Y. Fan, H. Lin. A learning framework for robust bin picking by customized grippers. *CoRR*, abs/1809.08546, 2018.

[14] K. Yu-E. Donlon F. Hogan M. Bauza D. Ma O. Taylor M. Liu E. Romo N. Fazeli F. Alet N. Dafle R. Holladay I. Morena N. Qu Nair D. Green I. Taylor W. Liu T. Funkhouser A. Rodriguez A. Zeng, S. Song. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3750–3757, 2018.

[15] Felix Spenrath and Andreas Pott. Gripping point determination for bin picking using heuristic search. *Procedia CIRP*, 62:606–611, 12 2017.

[16] Hsien-I. Lin and Subhajit Nanda. 6 dof pose estimation for efficient robot manipulation. *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*, 1:279–284, 2020.

[17] S. Niyaz-M. Laskey R. Doan X. Liu J. Ojea K. Goldberg J. Mahler, J. Liang. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *CoRR*, abs/1703.09312, 2017.

[18] D. Chai S. Caldera, A. Rassau. Review of deep learning methods in robotic grasp detection. *Multimodal Technologies and Interaction*, 2(3), 2018.

[19] M. Huber K. Kleeberger, C. Landgraf. Large-scale 6d object pose estimation dataset for industrial bin-picking. *CoRR*, abs/1912.12125, 2019.

[20] Y. Gao J. Yang, D. Li, and S. Waslander. ROBI: A multi-view dataset for reflective objects in robotic bin-picking. *CoRR*, abs/2105.04112, 2021.

[21] B. Hou-M. Roderick M. Laskey M. Aubry K. Kohlhoff T. Kroeger J. Kuffner K. Goldberg J. Mahler, F. Pokorny. Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with

correlated rewards. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1957–1964, 2016.

[22] G. Hinton A. Krizhevsky, I. Sutskever. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[23] A. Zisserman K. Simonyan. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[24] S. Ren-J. Sun K. He, X. Zhang. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[25] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):640–651, 2017.

[26] Andreas ten Pas, Marcus Gualtieri, Kate Saenko, and Robert W. Platt. Grasp pose detection in point clouds. *The International Journal of Robotics Research*, 36:1455 – 1473, 2017.

[27] Arsalan Mousavian, Clemens Eppner, and Dieter Fox. 6-dof graspnet: Variational grasp generation for object manipulation. In *International Conference on Computer Vision (ICCV)*, 2019.

[28] Peter Corke. *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*. Springer Publishing Company, Incorporated, 1st edition, 2013.

[29] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, USA, 1991.

[30] Shuai Liu and Pengcheng Liu. A review of motion planning algorithms for robotic arm systems. 11 2020.

[31] Kavraki Lab. Open motion planning library: A primer, 2021.

[32] Awesome Open Source. The top 154 kinematics open source projects on github, 2021.

[33] Orocos. Orocos kinematics and dynamics.

[34] Openrave. Ikfast: The robot kinematics compiler.

[35] Christian Henkel. The industrial trajectory generation and python api of pilz industrial motion, 2019.

[36] Karan Khokar, Patrick Beeson, and Robert Burridge. Implementation of kdl inverse kinematics routine on the atlas humanoid robot. *Procedia Computer Science*, 46:1441–1448, 12 2015.

[37] MoveIt! Ikfast kinematics solver, 2021.

[38] MoveIt. Moving robots into the future, 2020.

[39] MoveIt. Moveit setup assistant, 2021.

[40] MoveIt. Move group python interface, 2021.

[41] MoveIt. Moveit quickstart in rviz, 2021.

[42] D. Morrison. The benefit of feeding the rgb data, 2019.

[43] Kentaro Wada. Labelme releases, 2021.

[44] Wikipedia. Data augmentation.

[45] Dongheui Lee. Machine learning in robotics lecture slides. unpublished, 2020.

[46] Bing Yang. Deep learning lecture notes. unpublished, 2019.

[47] Sebastian Ruder. An overview of gradient descent optimization algorithms. 09 2016.

[48] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[49] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[50] Saad Albawi, Tareq Abed Mohammed, and Saad ALZAWI. Understanding of a convolutional neural network. 08 2017.

[51] Pytorch. Upsample.

[52] Sik-Ho Tsang. Review dilatednet, dilated convolution (semantic segmentation), 2018.

[53] WandelBots. *The Revolution of Robot Programming.*, 2020.

[54] Wenzeng Zhang, Xiande Ma, Leqin Cui, and Qiang Chen. 3 points calibration method of part coordinates for arc welding robot. pages 216–224, 10 2008.

[55] Universal Robot. *Universal Robot User Manual*, 2020.

[56] MoveIt. Hand-eye calibration, 2021.

[57] Opencv. Calibration with aruco and charuco, 2021.

License