Delta CODESYS robotics solution for random bin picking tasks using a 3D ToF camera

Zhang, Jie

Award date:
2023

Technische Universiteit
Eindhoven
University of Technology

Department of Mechanical Engineering
Microsystems Research Group

Delta CODESYS robotics solution for random bin picking tasks using a 3D ToF camera

Master Thesis

Jie Zhang

Supervisors:
Prof. Dr.Ir. J.M.J. (Jaap) Den Toonder
Jason Wu

final version

This report was made in accordance with the TU/e Code of Scientific Conduct for the Master thesis

Eindhoven, March 2023

Abstract

In recent years, industrial automation technology has evolved significantly and is constantly being improved to increase productivity and reduce the need for direct human intervention. It is a major challenge in many automated assembly processes to locate or grasp components in the correct orientation from a bin in which they are randomly positioned and unsorted. The process is known as random bin picking (RBP), and the difficulties associated with it have long been recognized.

For this graduation project, we used a Delta Electronics' 3D ToF camera with a CODESYS-based motion controller. The proposed system utilises a combination of computer vision modules in 2D and 3D, robotics and deep neural networks. We demonstrated the kinamatic relationship between the camera and the robotic manipulator. We developed grasping algorithms based on deep neural networks. The results of the algorithms were presented and tested in simulations of a planar grasp and six-degrees-of-freedom grasp. Additionally, we developed a demonstration application with a planar grasp algorithm.

Preface

This is a master's thesis in the Department of Mechanical Engineering at TU/e under the study program Artificial Intelligence Engineering Systems. In developing this master's thesis, I would like to acknowledge the following individuals for their contributions:

- Firstly, I would like to thank my supervisors, Jason Wu and Jaap de Toonder, for their contribution to this project and their invaluable guidance and advice
- Secondly, I would like to express my gratitude to my family and friends for their love and support
- Lastly, thanks to the other interns and employees at Delta Electronics for their constant support and encouragement

Contents

Contents iv

List of Figures v

List of Tables vi

1 Introduction 1
- 1.1 Background . . . 1
  - 1.1.1 Project Background . . . 1
  - 1.1.2 Company Background . . . 2
- 1.2 Problem Description . . . 3
- 1.3 Research Objective . . . 3
- 1.4 Research Questions . . . 4
- 1.5 Outline of contents . . . 4

2 Literature Review 5
- 2.1 Gripper oriented methods . . . 5
  - 2.1.1 Planar vs 6DoF . . . 6
- 2.2 Object oriented methods . . . 8

3 Theoretical Background 10
- 3.1 Deep Learning . . . 10
- 3.2 Neural Network . . . 10
- 3.3 Convolutional Neural Networks . . . 12
  - 3.3.1 Convolution Layer . . . 12
  - 3.3.2 Pooling Layer . . . 12
  - 3.3.3 Residual Blocks . . . 13
- 3.4 Transfer Learning . . . 14
- 3.5 Robotic Fundamentals . . . 14
  - 3.5.1 Coordinate Frame and Notation . . . 14
  - 3.5.2 Inverse Kinematics . . . 15
- 3.6 Camera . . . 16
  - 3.6.1 Hand Eye Calibration . . . 17

4 Method 19
- 4.1 Tools and Equipment . . . 19
  - 4.1.1 Robot arm . . . 20
  - 4.1.2 AX-8 motion controller . . . 20
  - 4.1.3 Servo drives . . . 20
  - 4.1.4 3D TOF camera . . . 21
  - 4.1.5 Custom made gripper . . . 22
  - 4.1.6 Master computer . . . 22
- 4.2 Hand-to-eye calibration . . . 23

4.3 AI Algorithm . . . 25
4.3.1 Datasets . . . 25
4.3.2 Planar grasp . . . 27
4.3.3 6DoF grasp . . . 30
4.4 Software Development . . . 33
4.4.1 Simulation environment setup . . . 33
4.4.2 Image Acquistion . . . 33
4.4.3 Robot Control . . . 34
4.4.4 Grasping Objects . . . 35
4.4.5 Code structure . . . 36

5 Results **39**
5.1 Planar Grasp . . . 39
5.1.1 Physical setup . . . 39
5.1.2 Eye-to-Hand Calibration . . . 40
5.1.3 Image Acquisition . . . 41
5.1.4 Neural Networks . . . 41
5.1.5 Simulation . . . 43
5.1.6 Real Application . . . 44
5.2 6DoF grasp . . . 45
5.3 Demonstration Video . . . 45

6 Discussions and recommendations **46**
6.1 AprilTag v.s. Chessboard . . . 46
6.2 Depth Image Quality . . . 46
6.3 Object segmentation . . . 47
6.4 Bin picking . . . 49

7 Conclusion **50**

Bibliography **51**

Appendix **55**

A Gripper Manual **56**
A.1 Wiring . . . 56
A.2 Control . . . 56

B Tensorboard Training Curves **58**
B.1 Planar Grasp . . . 58
B.2 6DoF Grasp . . . 63

C Digital Appendix **64**

List of Figures

1.1 Delta's industrial automation product line includes PLC motion controllers, servo systems (servo drives and servo motors), industrial robots, and HMIs, extracted from [1] . . . 2
1.2 Delta products. (left)Delta DRV Robot Arm. (right)Delta 3D ToF Camera 3

2.1 Bin picking based on deep learning category[2] . . . 5
2.2 Dex-Net 2.0 pipeline [3]. Using a dataset of 6.7 million synthetic point clouds, grasps, and associated robust grasp metrics computed with DexNet 1.0, the Grasp Quality Convolutional Neural Network (GQ-CNN) is trained offline to predict the robustness of candidate grasps from depth images. A depth camera provides the robot with a 3D point cloud that identifies several hundred potential grasps when an object is presented to it. . . . 6
2.3 A 5D rectangular grasp configuration[4] . . . 7
2.4 Grasp coordinate frame [5] . . . 8

3.1 A simple neural network, which is organized in layers consisting of a set of interconnected neurons. Networks can have tens or hundreds of hidden layers. [6] . . . 11
3.2 An example of convolution operation performed on an input image [7] . . . 12
3.3 An example of the max pooling operation using a 2×2 filter [7] . . . 13
3.4 An example of identical block. The skip connection "skips over" 2 layers [8] . . . 13
3.5 An example of convolutional block. There is a CONV2D layer in the shortcut path [8] . . . 13
3.6 Transfer learning [9]. The network is first trained on the source task (ImageNet classification, top row) using a large number of labelled images. In the next step, the parameters pre-trained in the internal layers of the network (C1-FC7) are transferred to the target tasks (Pascal VOC object or action classification, bottom row). . . . 14
3.7 A pinhole camera model where the image plane is located in front of the camera's origin. A non-inverted image is formed on the plane at $zi = f$ [10] . . . 16
3.8 A hand-eye calibration estimates the position of the camera relative to the robot's base H_{CAM}^{ROB} in eye-to-hand systems. The pose circle can be closed by calculating one pose from the other poses. In this case, the position of the object in relation to the robot. This can be determined by multiplying the pose of the camera relative to the robot with the pose of the object relative to the camera: $H_{OB,J}^{ROB} = H_{CAM}^{ROB} \cdot H_{OBJ}^{CAM}$ [11] . . . 17
3.9 Hand-to-eye calibration. Cameras are used to observe tool feature points mounted on the gripper in order to determine the tool's motion [12] . . . 18

4.1 An overview of our proposed RBP system. A number of grasping objects are placed in the Initiation state. A depth image from the Image Acquisition state will be used in the Grasping Prediction state. The proposed grasping location will be sent to the Robotic Manipulation state, where the final picking action will be performed. . 19

4.2 An illustration of the physical robot cell and the simulation environment. (a) Robot cell for real application. (b) A simulation of a robot cell in Pybullet 20

4.3 Technical specification of the robot arm extracted from [13]. (left)Articulated robot specification table. (right)DRV series motion directions 21

4.4 The AX-8 Windows-based PLC was used in the pratical work of this thesis [14] . . 21

4.5 ASDA-A2 servo drive with motor [15] 22

4.6 Diagram of the wiring between the robot, AX-8 motion controller and the custom-made gripper 23

4.7 Two examples of calibration pattern images taken by the 3D ToF camera in the amplitude mode 24

4.8 AprilTag detection using MATLAB calibration toolbox. (a) detected points shown in real images. (b) demonstration of MATLAB's detection function on a generated pattern 24

4.9 An illustration of the hand-to-eye calibration process. A total of 20 different images containing calibration patterns are imported into MATLAB. To get the final result of camera position and orientation, the output of the function along with the 20 different robot pose information is fed into OpenCV-Python function calibrationHandEye() 25

4.10 Sample of Cornell Grasp dataset [16] 26

4.11 A large variety of objects are included in the Jacquard dataset, each of which has multiple labeled grasps on realistic images. On the image, grasps are represented by 2D rectangles whose darker sides indicate the position of the jaws [17] 26

4.12 Food items included in the YCB object set. Back row, from left to right: a can of chips, a coffee can, a cracker box, a box of sugar, and a can of tomato soup. From left to right, middle row: mustard container, tuna can, chocolate pudding box, gelatin box, and potted meat can. An apple, a lemon, a pear, an orange, a banana, a peach, strawberries, and a plum are displayed in the front row [18] . . . 27

4.13 The GG-CNN architecture [19]. It is the last convolutional layer that generates the grasp and width images as well as the two angle images. The best grasp is separated from the quality distribution. 28

4.14 Proposed Generative Residual Convolutional Neural Network [20] 29

4.15 PointNetGPD algorithm flow [21]. Given raw RGB-D data from a sensor input, the depth map is first converted into a point cloud. Next, some candidate grasping poses are sampled based on geometric constraints. For each candidate, the point cloud inside the grasper is cropped and converted to the gripper's local coordinate system. Lastly, the candidate grasping inputs are fed into the grasping quality evaluation network to determine scores, and the candidate grasping poses with the highest scores are adopted. 30

4.16 PointNet-based network structure for grasp quality evaluation [21]. Based on the grasp pose and the original point cloud, the point cloud in the closed region of the grasper represents a grasp that is transformed into the grasper's local coordinate system and input into the network. As a result of multiple spatial transformations and feature extraction, the final global features are used to classify the input grasps according to their quality level. 31

4.17 Representation of a grasp in the local gripper coordinate system [21]. (a) a typical grasping pose; (b) the axes of the local coordinate system. The authors use the forward, parallel, and orthogonal directions of the gripper as the XYZ axes and the middle position of the bottom of the gripper as the origin, respectively. 31

4.18 (left) The grasps are estimated with respect to the center of mass of the object point cloud, X. The axes of the grasp coordinate frame are parallel to those of the camera. (right) An point cloud X is obtained by fitting a plane to a depth image. The Grasp Sampler Network uses the point cloud to propose different grasps. Based on the object point cloud and the proposed grasp, the evaluator network assesses the grasps. By using the gradient of the evaluator network, grasps are improved iteratively [5] 32
4.19 Each grasp is mapped to a point z in a latent space during training. Latent space distribution is minimized toward a normal distribution. Using the point cloud and latent values, the decoder reconstructs the 6D grasps, depicted here as gripper poses [5] 32
4.20 Camera parameters that can be adjusted using internal software. The integration time and intensity are defined by default as 1000 and 1.0 . Both of these parameters have a significant impact on the quality of the depth image 34
4.21 Different statuses of the axis group 35
4.22 An illustration of the general movement performed by the DRV robot axis groups . 36
4.23 Examples of grasping objects. (left) simulated objects from ShapeNetsem database in Pybullet. (right) real application grasping cubes provided by Delta 37

5.1 Schematic of the system. A motion controller and six servo drives are connected to the robot. A custom-made gripper connects the robot and controller via RS-485 communication. AX-8 and the master computer exchange data through OPC UA communication. Harvesters API is used on the master computer to control the 3D ToF camera using the GigE standard 39
5.2 Intrinsic paramters result from MATLAB. (left) camera centric (right) object centric 40
5.3 Example of amplitude and aligned depth images acquired with grasping objects (cubes) in the scene. 41
5.4 Accuracy curves and loss curves of planar grasp networks (a) GG-CNN(b) GR-ConvNet. The x-axis represents the epochs of the training process. The y-axis plots (from left to right) the IoU matrices (i.e. the grasp accuracy), the L2 loss function values of the training dataset and the L2 loss function values of the validation dataset 42
5.5 Training curves of VAE. The x-axis shows the steps taken during the training process, while the y-axis (from left to right, top to bottom) shows the reconstruction loss of the testing dataset (orientation and translation loss of the generated grasps with the ground truth grasps), confidence loss (confidence term that penalizes outputting zero confidence), KL-divergence loss, reconstruction loss, and total loss of the training dataset 43
5.6 Training curves of grasp evaluator. The x-axis shows the steps taken during the training process, while the y-axis (from left to right) indicates the accuracy of grasping the testing dataset, classification loss (cross-entropy loss), confidence loss (confidence term that penalizes outputting zero confidence) and total loss of the training dataset 43
5.7 Visualization of the grasping prediction module for real-world application. As shown in the image above, the depth image was captured and cropped, and a red dot indicates the predicted grasping center. Listed below are the grasp quality (successful grasp probability from 0 to 1), grasp angle in radian, and grasp width in pixels. . . 44
5.8 (a) Demonstration of generating predicted graphics from RealSense point clouds. (b) Simulation of grasping a mug with a gripper model in the Isaac Gym 45

6.1 Incorrect corner points detection on MATLAB. (a) Pose A (b) Pose B (approx. 180° from Pose A) 46
6.2 A comparison of depth images taken under different settings. (a) A smoother and less noisy depth image with higher integration time and intensity settings. (b) A depth image with more noise when the integration time and intensity are reduced . 47

6.3 Architecture of the proposed model includes a segmentation module [22]. The backbone network is shared by both branches for grasp detection and segmentation. A grasp refinement head uses both outputs (grasp candidates and semantic segmentation) to predict refined grasp candidates with increased accuracy 48
6.4 Results of the testing of the grasping module using the segmentation method [22]. The following images are explained from left to right: 1) the raw input image; 2) predicted semantic segmentation, where each color represents a specific class; and 3) the best possible grasp for each class in the scene (blue lines indicate parallel plates of the gripper, red lines indicate opening width). Each row represents a different example of input 48

B.1 GG-CNN training curve. The x-axis represents the epochs of the training process. The y-axis plots (from left to right) the IoU matrices (i.e. the grasp accuracy), the L2 loss function values of the training dataset and the L2 loss function values of the validation dataset 58
B.2 GG-CNN training dataset loss curves. The x-axis represents the epochs of the training process. The y-axis plots (from left to right) the angle *cos* loss, the probability loss, the angle *sin* loss, the width loss of the training dataset 58
B.3 GG-CNN validation dataset loss curves. The x-axis represents the epochs of the training process. The y-axis plots (from left to right) the angle *cos* loss, the probability loss, the angle *sin* loss, the width loss of the validation dataset 59
B.4 GG-CNN modified version testing dataset loss curve. The x-axis represents the steps of the training process. The y-axis plots (from left to right) the total loss, the angle *cos* loss, the probability loss, the angle *sin* loss and the width loss of the testing dataset 59
B.5 GG-CNN modified version accuracy curve. The x-axis represents the steps of the training process. The y-axis plots (from left to right) the prediction accuracy (between the predicted outputs and ground truth) and the probability accuracy of the testing dataset 59
B.6 GG-CNN modified version training dataset loss curve. The x-axis represents the steps of the training process. The y-axis plots (from left to right) the total loss, the angle *cos* loss, the probability loss, the angle *sin* loss and the width loss of the training dataset 60
B.7 GG-CNN modified version accuracy curve. The x-axis represents the steps of the training process. The y-axis plots (from left to right) the prediction accuracy (between the predicted outputs and ground truth) and the probability accuracy of the training dataset 60
B.8 GRConvNet training curve using Cornell dataset. The x-axis represents the epochs of the training process. The y-axis plots (from left to right) the IoU matrices (i.e. the grasp accuracy), the L2 loss function values of the training dataset and the L2 loss function values of the validation dataset 60
B.9 GRConvNet training dataset loss curves. The x-axis represents the epochs of the training process. The y-axis plots (from left to right) the angle *cos* loss, the probability loss, the angle *sin* loss, the width loss of the training dataset 61
B.10 GRConvNet validation dataset loss curves. The x-axis represents the epochs of the training process. The y-axis plots (from left to right) the angle *cos* loss, the probability loss, the angle *sin* loss, the width loss of the validation dataset 61
B.11 GRConvNet training curve using Jacquard dataset. The x-axis represents the epochs of the training process. The y-axis plots (from left to right) the IoU matrices (i.e. the grasp accuracy), the L2 loss function values of the training dataset and the L2 loss function values of the validation dataset 61
B.12 GRConvNet training dataset loss curves. The x-axis represents the epochs of the training process. The y-axis plots (from left to right) the angle *cos* loss, the probability loss, the angle *sin* loss, the width loss of the training dataset 61

B.13 GRConvNet validation dataset loss curves. The x-axis represents the epochs of the training process. The y-axis plots (from left to right) the angle *cos* loss, the probability loss, the angle *sin* loss, the width loss of the validation dataset 62
B.14 Training curves of VAE. The x-axis shows the steps taken during the training process, while the y-axis (from left to right, top to bottom) shows the reconstruction loss of the testing dataset (orientation and translation loss of the generated grasps with the ground truth grasps), confidence loss (confidence term that penalizes outputting zero confidence), KL-divergence loss, reconstruction loss, and total loss of the training dataset 63
B.15 Training curves of grasp evaluator. The x-axis shows the steps taken during the training process, while the y-axis (from left to right) indicates the accuracy of grasping the testing dataset, classification loss (cross-entropy loss), confidence loss (confidence term that penalizes outputting zero confidence) and total loss of the training dataset 63

List of Tables

4.1 Technical specifications for 3D ToF camera 22

5.1 Hyperparameters values 42
5.2 Pick success rate (%) in simulation 44
5.3 results of grasp success rates (%) among different categories of objects 45

Chapter 1

Introduction

1.1 Background

1.1.1 Project Background

In recent years, industrial automation technology has evolved significantly and is constantly being improved to increase productivity and reduce the need for direct human intervention. The goal of robotic research in this area is to develop a system that is more intelligent, flexible, and autonomous. This type of system would be capable of completely replacing human labor in the implementation of certain tasks, including those in the field of automated manufacturing. It is a major challenge in many automated assembly processes to locate or grasp components in the correct orientation from a bin in which they are randomly positioned and unsorted. The process is known as random bin picking (RBP), and the difficulties associated with it have long been recognized [23]. Most parts are found randomly located in boxes or bins, and objects to be processed on different production lines have a wide range in geometric shape, color, texture, and surface. As a result, it remains challenging and complex to obtain satisfactory perceptual abilities for a complete and optimal robot bin-picking system [24]. Objects must be able to be positioned randomly in an unstructured and poorly constrained occlusion in a heavily cluttered environment for the system to perform effectively. In the field of image processing and automated manufacturing, RBP has been the focus of research for many years owing to its necessity and high applicability. In spite of this, most research results have been somewhat limited due to simplistic hypotheses or insufficient robustness for industries with strict requirements for speed and stability [25].

There are several distinct functions involved in picking random bins by robots: creating an image of the objects and containers; isolating the object from the background image; determining the position of objects relative to the image sensor or robotic arm; generating a trajectory for moving the robot to grasp the objects; and finally gripping the part and transferring it to the required location [26]. As a result, several technologies are involved, which must be seamlessly integrated into a robotic RBP system: imaging and lighting, data processing, component handling, and robotic guidance [26]. The two most important technologies that support robotic RBP are three-dimensional (3D) vision and algorithms that interpret the images. The development of machine vision technology over the last few years has been very rapid due to an increase in computation power and many 3D imaging systems are now available commercially based on a variety of techniques, along with ever more sophisticated image processing algorithms for RBP applications [27]. Several 3D vision systems incorporate conventional two-dimensional(2D) cameras with laser line scanning projections for obtaining depth information, while others employ triangulation and time-of-flight (ToF) techniques.

Traditional RBP systems utilize task-specific algorithms that are specially tailored to meet the requirements of each individual situation. While this approach can be efficient, it has significant

limitations in terms of generalizability and unexpected situations, such as novel objects. However, the use of deep learning shows the potential to revolutionize the field of robotics and improve the efficiency of the picking processes. Unlike task-specific algorithms, it is capable of generalizing new objects and environments. Additionally, the deep learning approach is capable of adapting to changes in the environment, resulting in a more robust system. Furthermore, deep learning algorithms can learn from experience, allowing them to continuously improve.

1.1.2 Company Background

This project has been conducted out in the company called Delta Electronics [1] which was founded in 1971 and is headquartered in Taiwan. It is a global company leading in Power and Thermal management solutions. Its business verticals include Power Electronics, Automation, and Infrastructure. Delta has a sales presence in all 6 continents and manufacturing plants in Asia, Europe and Americas. Delta Group established its EMEA operation in 1995, setting up its regional headquarters in Hoofddorp, near Amsterdam. Delta EMEA offers a wide portfolio of products and solutions in EV Charging, Solar Inverters, Telecom Powers, Data Centers, Industrial Automation and Display [1].

The site in Helmond offers automation products and solutions with high performance and reliability, including drives, motion control systems, industrial control and communications, power quality improvement, human machine interfaces, sensors, meters and robot solutions. The center especially provides information monitoring and management systems such as SCADA (Supervisory Control And Data Acquisition) and industrial EMS (Electronics Manufacturing Services) for complete, smart manufacturing solutions. The company's basic information, including the logo, is shown in Figure 1.1.

Figure 1.1: Delta's industrial automation product line includes PLC motion controllers, servo systems (servo drives and servo motors), industrial robots, and HMIs, extracted from [1]

Delta Electronics (Netherlands) BV
Automotive Campus 260, 5708 JZ Helmond, The Netherlands
+31 40 800 3900

1.2 Problem Description

Delta CODESYS robotics solution for random bin picking tasks using a 3D ToF camera

This graduation project aims to provide a functional solution for a vision-based bin-picking system. By utilizing the information from a 3D ToF camera, the vision system should be able to locate and recognize objects randomly placed in a box. Following the coordinate transformation from the vision system, the robot arm would be able to pick up the required object. The 3D camera is capable of capturing high-resolution, real-time images of objects which are used as the inputs for the grasping strategy trained by a convolutional neural network. The company provides the robot arm and 3D ToF camera as shown in the Figures 1.2 below. CODESYS SoftMotion Robotics library would be used to control the movement of the robot arm during picking. As a result of this project, a simulation environment and a real-world application would be developed that can demonstrate the accuracy of bin-picking.

(a) (b)

Figure 1.2: Delta products. (left)Delta DRV Robot Arm. (right)Delta 3D ToF Camera

1.3 Research Objective

Our objective is to investigate the means by which algorithm can be trained to interact with the robot arm and objects, in particular, grasping and positioning them in intended positions while integrating a 3D ToF camera. The following steps must be taken in order to achieve this goal:

- Explore existing solutions:
  To realize this project, it is essential to have a solid understanding of the existing deep learning strategies for RBP. We need to understand both the core concepts and popular methods associated with deep learning for RBP. Furthermore, we must be able to compare our proposed work with current state-of-the-art research.

- Discover machine learning libraries and robot simulation environments:
  It is necessary to be familiar with the machine learning libraries that are related to deep learning methods. Additionally, we require an open source robot simulation environment in order to conduct our experiments. We should develop and modify these environments to meet the needs of our project.

- Create a functional algorithm and training framework:
  In order to achieve RBP, the algorithm developed should be capable of identifying and recognising objects, predicting the picking position, and mapping coordinates between the 3D ToF camera and the robot. It is also necessary to implement a framework that allows

a robot arm to interact with objects. A comprehensive description of the entire training process, from collecting training samples to conducting online training, should be included.

- Analyze and improve the algorithms and structures:
  The advantages and disadvantages of different learning algorithms and neural network structures must be compared in order to determine which is best. It is then necessary to improve both algorithms and structures in order to meet our needs and to accomplish our task.

1.4 Research Questions

The research questions form the backbone of the research and are formulated as follows:

How to develop a CODESYS-based robotics solution for RBP tasks using a 3D ToF camera and a Delta DRV robot arm?

The main question can be divided into several sub-questions, which are formulated as follows:

- What kind of method will be used for objects identifying and recognizing?
- What is the software structure of the system?
- Which programming language and development software will be used for the vision system and robot arm manipulating?
- What is the communication method between the robot arm and the vision system?
- What kind of experiments need to be done for performance evaluation?

1.5 Outline of contents

In this thesis, both theoretical and practical research is conducted that covers the available technologies that can be applied in an RBP system, including extensive literature and theory research. In this section, a brief overview of the thesis' structure is provided along with a summary:

- **Chapter 2** discusses robotic grasping challenges and other researchers' attempts to achieve rapid and reliable grasping.
- In **Chapter 3**, the concepts of neural networks, convolutions, coordinate frames, camera models, and inverse kinematics are introduced.
- In **Chapter 4**, we discuss the methods used for testing and evaluating practical experiments. After reviewing our network, we discuss improvements and extensions to existing network models.
- Presented in **Chapter 5** are the results based on Chapter 4
- **Chapter 6** presents personal thoughts and discussions related to the results in Chapter 5. The strengths and weaknesses of the practical work are discussed, as well as possible improvements.
- In **Chapter 7**, a summary of our findings is provided along with answers to the research questions.

Chapter 2

Literature Review

Vision-based robotic grasping can be classified according to a number of different criteria. As shown in Figure 2.1, it can be divided into several domains. Bin picking can be divided into two main topics, as inspired by the work of Kleeberger et.al [28] and Matthieu Grard [29]: gripper-oriented and object-oriented, which are discussed in turn in the following sections.

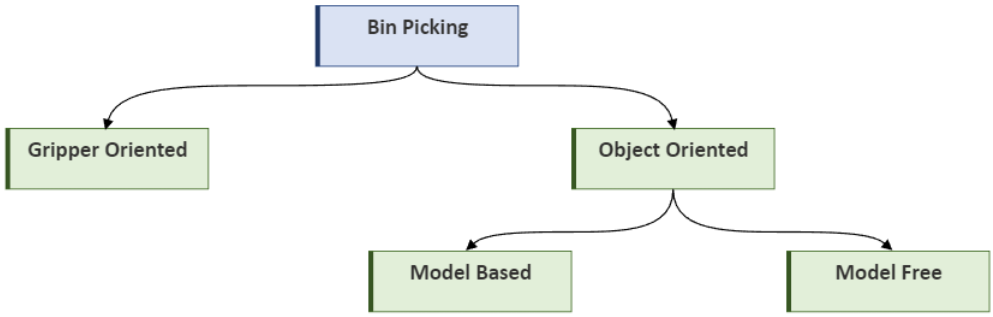


Figure 2.1: Bin picking based on deep learning category[2]

Gripper-oriented approach lacks the notion of instance, so it does not identify the various objects occupying the workspace, which is necessary for handling occlusions in dense stacks of instances. Object-oriented approach relies either on the notion of pose or on generic segmentation techniques. The approach can be divided into model-based, or analytical approaches [30], and model-free, or data-driven approaches.

2.1 Gripper oriented methods

In gripper-oriented methods, the robot end effector physics are taken into consideration in order to detect grasping opportunities. Unsupervised heuristic methods were used in early approaches in order to determine the optimal locations for parallel grippers or the most suitable planar area for vacuum cups using RGB and depth images. One of the first and most well-known heuristic procedures was introduced by Miller et al. [31]. For this application, Miller utilized heuristic rules

to generate and evaluate grasps for three-fingered hands, modeling an object as a set of shape primitives, including spheres, cylinders, cones and boxes.

In later stages of the research, deep convolutional networks (DCN) were employed to boost ranking of heuristic-based grasp candidates for more complex gripper-oriented methods. In order to gain a better understanding of how these methods work in bin picking environments, we will briefly describe some of these methods based on DCN.

There has been significant progress in developing methods with the best results and with greater depth since 2017. Dexterity Network (Dex-Net) 2.0 [3] is one of the most popular ones. It resulted in a Grasping Quality Convolutional Neural Network (GQ-CNN) model capable of predicting grasp success probabilities from depth images in a short amount of time. According to [3], grips are specified in terms of their planar position, angle, and depth relative to an RGB-D sensor. A pipeline can be identified in Figure 2.2 for this network.

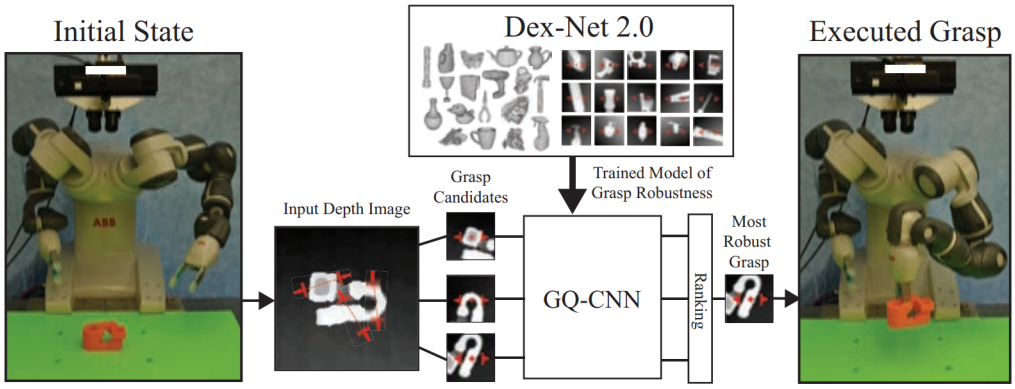


Figure 2.2: Dex-Net 2.0 pipeline [3]. Using a dataset of 6.7 million synthetic point clouds, grasps, and associated robust grasp metrics computed with DexNet 1.0, the Grasp Quality Convolutional Neural Network (GQ-CNN) is trained offline to predict the robustness of candidate grasps from depth images. A depth camera provides the robot with a 3D point cloud that identifies several hundred potential grasps when an object is presented to it.

2.1.1 Planar vs 6DoF

A different aspect of categorizing vision-based robotic grasping is brought to our attention here. To grasp an object, it is necessary to know the 6-dimensional pose of the robot end-effector in the camera coordinates. Our discussion of robot end-effectors in this thesis is limited to parallel grippers. As described in the grasping configuration, the gripper 6D pose is arranged to ensure that the object is successfully grasped. The grasping detection methods based on deep learning can be classified into planar grasp and 6 DoF grasp in accordance with the different grasping configurations.

A. Planar Grasp

In a 3DoF grasp, the target object is placed within a planar workspace, and the grasp is constrained in one direction perpendicular to the workspace. Therefore, it is also referred to as a 2D plane grasp, which involves a 2D in-plane location and a 1D rotation. There are a large number of methods using deep learning that treat oriented rectangles as grasp configurations. As a result, the capabilities of 2D planar grasp have been greatly expanded, and the range of objects to be

grasped has expanded from known objects to novel ones. There are two types of 2D planar grasping methods: structured grasping and pixel-level grasping.

i. Structured Grasping During the early stages of the research, the grasping configuration was based on points on scene images. In order to estimate the graspable point position in the cartesian coordinate system, Saxena et al. [32] proposed a regression learning method in an effort to find graspable points in the discrete 3D space. Although this approach only determines where to grasp, it does not determine the orientation of the gripper. As a solution to this limitation, an oriented rectangle is proposed as a representation of grasping configurations. Jiang et al. [**?**] proposed using a directed rectangle which contains three-dimensional position, three-dimensional orientation, and the gripper opening width, expressed as $G = (x, y, z, \alpha, \gamma, \phi, w)$, to estimate the 7D grasp. It is computationally expensive to present such information. In order to simplify the above-mentioned grasping configuration from 7D to 5D, Lenz et al. [16] used the rectangle with location, orientation, and size: $G = (x, y, \theta, h, w)$. Figure 2.3 illustrates the used rectangle representation. This was the first time that deep learning has been applied to robotic grasping. Two networks were proposed to be used as a two-step continues system, first removing candidate grasps that were impossible, and then re-evaluating the remaining grasps to find the top-ranked rectangles. Several subsequent studies have used this 5D rectangle grasp representation. With the 5D configuration, Redmon et al. [33] addressed the same problem as [16], but employed a different network architecture that performs a single-stage regression from RGB-D images to graspable bounding boxes.

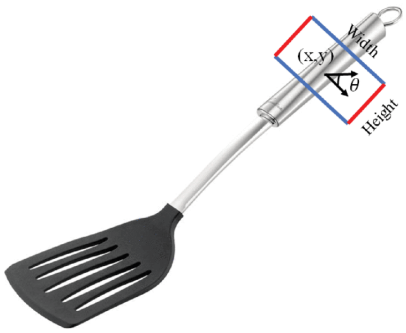


Figure 2.3: A 5D rectangular grasp configuration[4]

ii. Pixel-level grasping Pixel-level grasping configurations are designed to estimate the grasp quality for each pixel or to estimate pixel-by-pixel grasp affordances to evaluate the most probable grasping contacts. The Generative Grasping Convolutional Neural Network (GG-CNN) proposed by Morrison et al. [19] translates a depth image into a grasp map, which is composed of three pixel images: grasp quality, grasp angle, and grasp width. Based on these three pixel images, a grasp is determined for each pixel. Zeng et al. [34] also presented a framework for predicting grasp locations, orientations, and confidence scores at pixel-level according to grasp affordances. Parallel gripper grasps and suction cup grasps are pre-defined as grasp affordances. The authors used two fully convolutional networks in order to predict whether the object can be sucked or grasped from 16 different angles, and then derived grasp proposals with confidence scores.

B. 6DoF grasp

As shown in Figure 2.4, a 6DoF grasp requires an estimation of the gripper's 6D pose in camera coordinates to allow it to grasp objects at various angles. The 6D pose includes the 3D position and 3D orientation of the gripper. Analytical methods were initially used to analyze the geometric structure of the 3D data, and the points suitable for grasping were determined by force closure [29]. In [35], an overview of 3D object grasping algorithms was presented, where most of the algorithms dealt with complete shapes. Monocular object 6D pose estimation [36][37] is extensively researched as depth images become readily available. With the camera's intrinsic parameters, the depth image is easily lifted into a 3D point cloud, making depth image-based 6DoF grasps a hot area of research. The majority of 6DoF grasp methods aim at known objects where the grasps can be precomputed. Then 6DoF grasps can be estimated by sampling and ranking the grasp poses in the knowledge base. The problem of estimating grasp poses is transformed into estimating 6D poses for the target object, which we previously referred to as object-oriented methods. On the basis of prior knowledge about the shape of the objects, Deng et al. [38] predict the 6D poses of the objects, and then projects these predefined grasp poses onto the workspace.

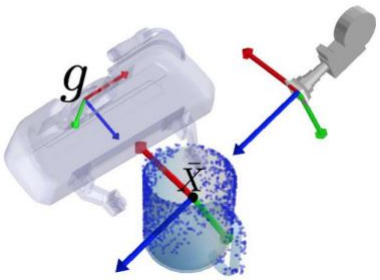


Figure 2.4: Grasp coordinate frame [5]

From [39], there has been a new research direction based on the partial point cloud that requires no prior knowledge of objects, and analyzes the input partial point cloud to estimate the 6DoF grasp poses. The majority of these methods recommend grasp candidates and provide estimates of their grasp quality. In [39] proposed GPD algorithm, grasp candidates are first sampled from a region of interest (ROI) determined by preprocessed viewpoint clouds, then encoded as stacked multi-channel images. A CNN algorithm can be used to evaluate the score of each candidate, and the grasps that will be executed are selected based on that score. PointNetGPD was proposed by Liang et al. [40] as a further expansion. Take raw point clouds as input instead of multi-view projection features. Following that, a geometric analysis based on PointNet should be conducted to evaluate the quality of the sampled candidate grasps. In general, this work outperforms GPD when the input point cloud is sparse.

2.2 Object oriented methods

Object-oriented bin-picking aims to provide affordable instances independent of the gripper model. From this perspective, the disparity between object-oriented and gripper-oriented approaches can be explained by the fact that object-oriented approaches are strongly related to occlusion perception, whereas gripper-oriented approaches are more focused on friction forces and gripper torques [29].

Object-oriented approaches can be divided into two categories: analytical or model-based approaches, and data-driven or model-free approaches [28]. In contrast, analytical methods assume an explicit model of the target, whereas data-driven methods are mainly based on image segmentation techniques, such as neural networks for segmenting images [29].

Analytical approaches (model-based): analyze the shape of the target object to determine an appropriate grasp position. These calculations are often based on certain information regarding the models, such as points of contact, Coulomb friction, and rigid body modeling [2]. In order to formulate the problem as a constrained optimization problem, geometric, kinematic, and dynamic formulations are required [41]. To conclude, this solution involves a model-based method, which, as its name suggests, relies on a model, such as a CAD model, in order to solve the problem.

Data-driven approaches (model-free): approaches that are based on machine learning [28], and have gained popularity in recent years. Using these methods, grasping candidates for unknown objects are sampled and ranked according to a certain metric [41]. Unlike analytic methods, they do not require a model, such as a CAD model or previously scanned model; instead, they require labels collected by humans or labeling processes, physical trial and error, heuristic methods, or a process based on human or robot demonstrations.

Chapter 3

Theoretical Background

3.1 Deep Learning

During the past few years, deep structured learning, or deep learning as it is commonly referred to, has emerged as one of the most promising fields of machine learning research. As a result of drastic improvements in chip processing capabilities, the considerable increase in data size used for training, and recent advances in machine learning, deep learning is becoming increasingly popular today. The techniques developed from deep learning research have had a profound impact on a wide range of signal and information processing work in recent years. This is especially true in the areas of image and object recognition, as well as speech recognition [42][43].

As a result of these developments, deep learning techniques are able to exploit complex, non-linear functions, to learn distributed and hierarchical representations of features, and to use both labeled and unlabeled data effectively [44].

3.2 Neural Network

There has been research into neural networks since the 1960s [45]. It was based on simulating the human brain and the objective was to find a principled approach to solving general learning problems. In a standard artificial neural network (ANN or NN), many simple, connected units are referred to as artificial neurons, which are loosely inspired by the neurons found in a biological brain. Signals can be transmitted from one artificial neuron to another through every connection between two neurons.

A typical implementation of an ANN involves sending real numbers between neurons, and calculating the output of each artificial neuron by summing the weights of its inputs. By implementing an activation function, signals are only sent from one neuron to the next if the threshold is crossed [46]. The goal of learning is to find weights that enable the ANN to display desired behavior, such as grasping an object in our case. Depending on the weight, the signal strength at a connection may be increased or decreased. In most cases, artificial neurons are arranged in layers. Input may be transformed in different ways by different layers. Upon receiving complex data inputs, input neurons receive weighted connections from previously active neurons, which activate other neurons. By triggering actions, output neurons may influence the environment. There may be a need for long chains of computational stages (layers) depending on the nature of the problem and the way the neurons are connected. Figure 3.1 shows the structure of a simple neuronal network.

An activation function defines the function applied to the input of a neuron in order to achieve a particular output. In modern neural networks, the default activation function is the Rectified

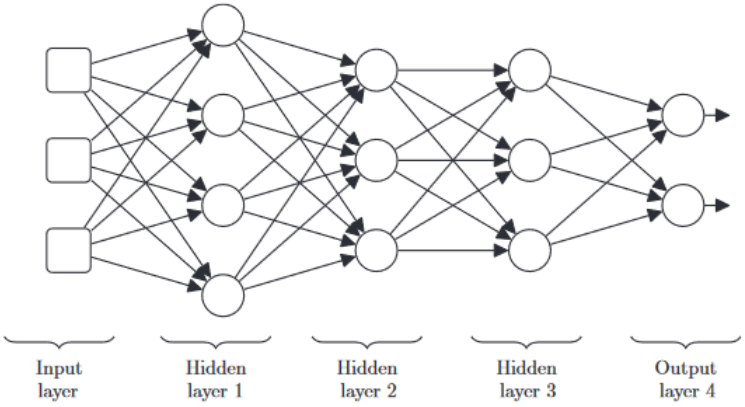


Figure 3.1: A simple neural network, which is organized in layers consisting of a set of interconnected neurons. Networks can have tens or hundreds of hidden layers. [6]

Linear Unit (ReLU) [44]. For any negative input x, this function returns 0; however, for any positive input x, this function returns that value. Therefore, it can be written as $f(x) = max(0, x)$.

3.3 Convolutional Neural Networks

Convolutional neural networks, also called CNNs, are specialized types of neural networks used to process data with known grid-like topologies. It is common to use CNNs in computer vision tasks related to classification. Srivastava et. al demonstrated the accuracy of the network structure in 2012 [43], when they won the computer vision competition ILSVRC-12 [47], by a significant margin.

In a CNN structure, there are typically one or more convolutional layers followed by a pooling layer and a fully connected layer. Through local connections and correlated weights, this network structure takes advantage of the 2D input structure, resulting in translation-invariant features.

3.3.1 Convolution Layer

Initially, a convolution neural network is set up with a convolution layer. An input image is projected with a given feature, resulting in a stack of processed, filtered images. Each pixel in the input images is mapped to the projected feature structure, and the rate of accuracy is computed on the new filtered image. In Figure 3.2, a colored input image is treated as a three-dimensional matrix of pixels, while a filter with a two-dimensional weight array is slid across the receiving field of the image to identify the presence or absence of a feature. In general, the filter is a $3x3$ matrix applied to the input image, and the dot product is calculated from the pixels and filter weights. Activation maps or feature maps are created by the output of each dot product. The activation map contains feature information extracted from an image using a specific filter. Pixel areas with high mapped values indicate that the given feature is present in that particular area. A stack of filtered images can be created by repeating this procedure for a variety of feature structures.

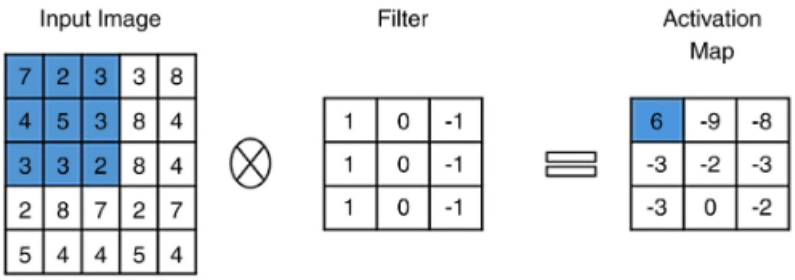


Figure 3.2: An example of convolution operation performed on an input image [7]

3.3.2 Pooling Layer

A CNN may include local or global pooling layers that combine the outputs of several neurons at a given layer into a single neuron at the next layer. Max-pooling as shown in Figure 3.3, for example, reports the maximum output within a neighborhood. All in all, pooling provides a representation that is approximately invariant to small translations of the input. The location of the eyes does not need to be precise when determining if an image contains a face; we are only concerned with the presence of eyes. As pooling summarizes responses over a whole neighborhood, fewer pooling units may be required than detector units. In this way, the network is made more efficient in terms of computation. In situations in which the number of parameters in the next layer is a function of its input size (e.g., when the layer is fully connected and based on matrix multiplication), the decrease in input size may result in a reduction in the memory requirements for storing the parameters [48]. By reducing the number of parameters, the search space for optimization was also reduced. In addition to preventing overfitting, fewer parameters facilitate a faster and more efficient training process.

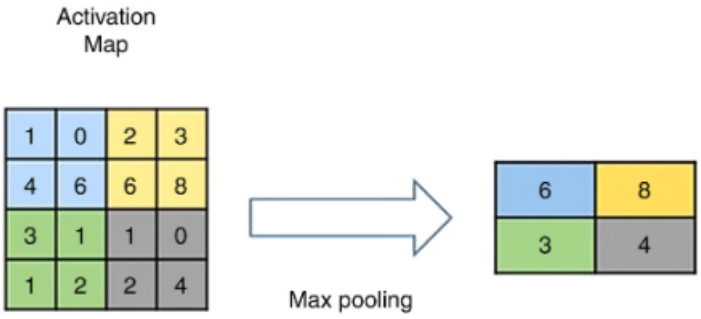


Figure 3.3: An example of the max pooling operation using a 2 × 2 filter [7]

3.3.3 Residual Blocks

A residual block consists of a stack of layers arranged in such a way that the output of one layer is added to another layer deeper within the block. The non-linearity is then applied after adding it to the output of the corresponding layer in the main path. It is called a shortcut or a skip-connection because it bypasses the main connection. Deeper networks begin to converge, exposing a degradation problem: with increasing network depth, accuracy becomes saturated and rapidly degrades [49]. As a result of the skipping connection/residual connection, adding new layers guarantees that the model's performance will not decrease, but might increase slightly [50].

In general, there are two types of blocks, depending on whether the input and output dimensions are the same or different.

- Identical residual block (shown in Figure 3.4)

  In an identical residual block, the shortcut path and main path outputs have the same dimensions. As a result of padding the input to each convolutional layer in the main path, the output and input dimensions remain unchanged.

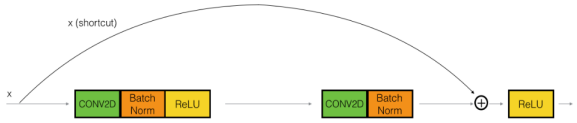


Figure 3.4: An example of identical block. The skip connection "skips over" 2 layers [8]

- Convolutional residual block (shown in Figure 3.5)

  This type of residual block uses a convolutional layer to resize the output of the shortcut path to match the dimensions of the main path. Additionally, the layer can control the dimension of the output volume by using different filter sizes, including 1x1, padding, and strides.

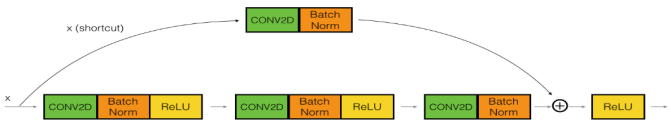


Figure 3.5: An example of convolutional block. There is a CONV2D layer in the shortcut path [8]

3.4 Transfer Learning

In order to adjust weights and biases between layers during training, deep neural networks require a substantial amount of labeled data. In many applications, it is not feasible to access an extensive amount of labeled data during training since the number of parameters may be in the range of hundreds to millions [9]. A transfer training method entails reusing a pre-trained network from a larger dataset in order to reduce the dependence between weights and biases [51]. The ImageNet [43] dataset is one of several public datasets available today that are sufficient in size for training large neural networks. In neural networks, general features are learned early in the network structure, while specific features for classification are learned in the last layers, so weights and biases learned from these datasets can be applied to new target objects. Only the last layers of the deep learning model are trained during transfer learning in order to achieve classification. Thus, deep learning can be applied in situations where there is a limited amount of labeled data available. In Figure 3.6, the concept of transfer learning is demonstrated, where the classifier of the target task is trained after the parameters from the source task have been transferred.

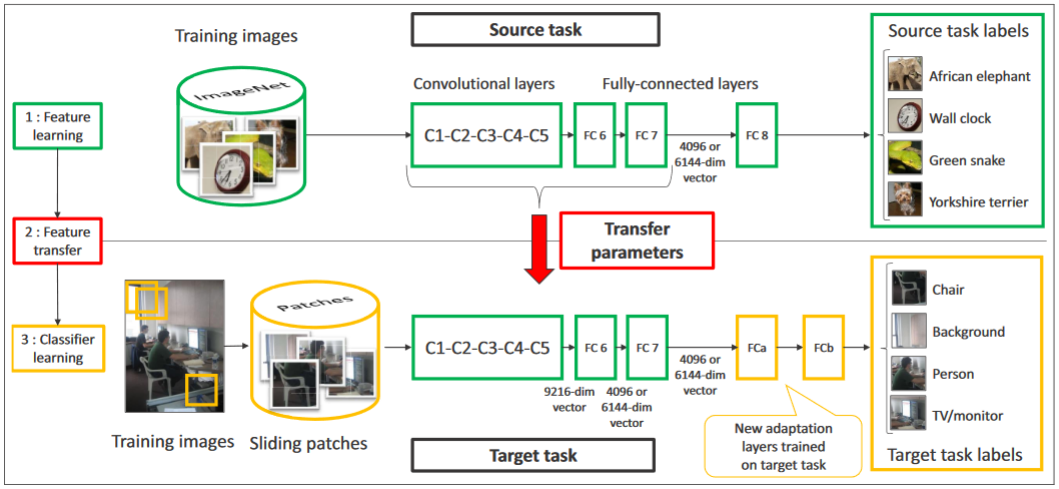


Figure 3.6: Transfer learning [9]. The network is first trained on the source task (ImageNet classification, top row) using a large number of labelled images. In the next step, the parameters pre-trained in the internal layers of the network (C1-FC7) are transferred to the target tasks (Pascal VOC object or action classification, bottom row).

3.5 Robotic Fundamentals

3.5.1 Coordinate Frame and Notation

A rotation matrix is a transformation matrix that produces a rotated vector such that the coordinate axes remain constant. Every rotation matrix has the following form in two dimensions:

$$R = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

This rotates the point in the xy-cartesian plane counterclockwise through an angle θ about its origin.

Below are the three basic rotation matrices in three dimensions:

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \mathbf{R}_y = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \mathbf{R}_z = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where α ,β and γ are the angles rotated about the x,y and z axis respectively. To align the reference frame's axes with the body frame's axes, rotation matrices describe how to rotate about an axis in space. Consider two coordinate systems C (camera coordinate) and W (world coordinate) that differ by a rotation. The rotation matrix can be used to transform a point from one coordinate system to another. For example, if we have the coordinates of a point in coordinate system C, we can find the equivalent set of coordinates in coordinate system W using the rotation matrix:

$$^{W}\mathrm{P} = \mathbf{R}_{\mathbf{C}}^{\mathbf{W}}{}^{C}\mathrm{P}$$

It is important to note that these rotation matrices are orthonormal, which means that:

$$\mathbf{R}_{\mathbf{C}}^{\mathbf{W}} = \mathbf{R}_{\mathbf{W}}^{\mathbf{C}}{}^{-1} = \mathbf{R}_{\mathbf{W}}^{\mathbf{C}}{}^{T}$$

Translation vectors describe the relative displacement between two frames in the same way that rotation matrices describe relative orientation between two frames. A 6 DoF pose consists of position (3DoF) and orientation (3DoF). For example, the transformation of a position vector P^B to P^A can be expressed as follows:

$$^{A}\mathrm{P} = \mathbf{R}_{\mathbf{B}}^{\mathbf{A}}{}^{B}\mathrm{P} + \mathbf{t}_B^A$$

Where $\mathbf{R}_{\mathbf{B}}^{\mathbf{A}}$ is the rotation matrix and $\mathbf{t}_B^A$ is the translation vector. Alternatively, we can use the homogeneous transformation matrix:

$$\mathbf{T}_B^A = \begin{pmatrix} \mathbf{R}_B^A & \mathbf{t}_B^A \\ \mathbf{0}^{1x3} & 1 \end{pmatrix}$$

$$^{A}\mathrm{P} = {}_{B}^{A}\mathrm{T}^{B}P$$

3.5.2 Inverse Kinematics

Inverse Kinematics is the opposite of Forward Kinematics. It involves transforming a given position of the end-effector in space into the joint space values needed to attain the desired position [52]. When it comes to bin picking, the inverse kinematics problem is generally more relevant than the forward kinematics problem. Using this method, the planner can plan the trajectory of the end effector during motion planning. A set of joint values corresponds to one position of the end-effector frame in forward kinematics. Inverse kinematics, on the other hand, might allow for multiple joint values for any given end effector frame. Consequently, the inverse kinematics problem is more complex since the equations to be solved are generally nonlinear.

There are two common approaches to solving inverse kinematics problems: analytical and numerical. Analytical approaches calculate joint angles by utilizing a mathematical formula based on the position of the end-effector. According to the joint parameters and end-effector poses, inverse kinematics can find all possible joint angles analytically based on the lengths of the linkages, the starting position, and the rotation constraints. Since the kinematic equations are nonlinear and scalability is limited for redundant robot configurations, analytical inverse kinematics is typically applied to robots with a low degree of freedom. The numerical solution can be used to approximate robot configurations that meet specified goals and constraints. Each joint angle is calculated iteratively using algorithms for optimization, such as gradient-based methods. In comparison with analytic inverse kinematics solvers, numerical solvers are more general, but require a number of steps in order to arrive at a solution to the nonlinearity.

3.6 Camera

Cameras can be used by robots to perceive the environment. As a result, navigation and manipulation of objects in an unknown and dynamic environment are enabled. Figure 3.7 illustrates a pinhole camera model that can be used to project and map a 3D world coordinate point onto a 2D image plane.

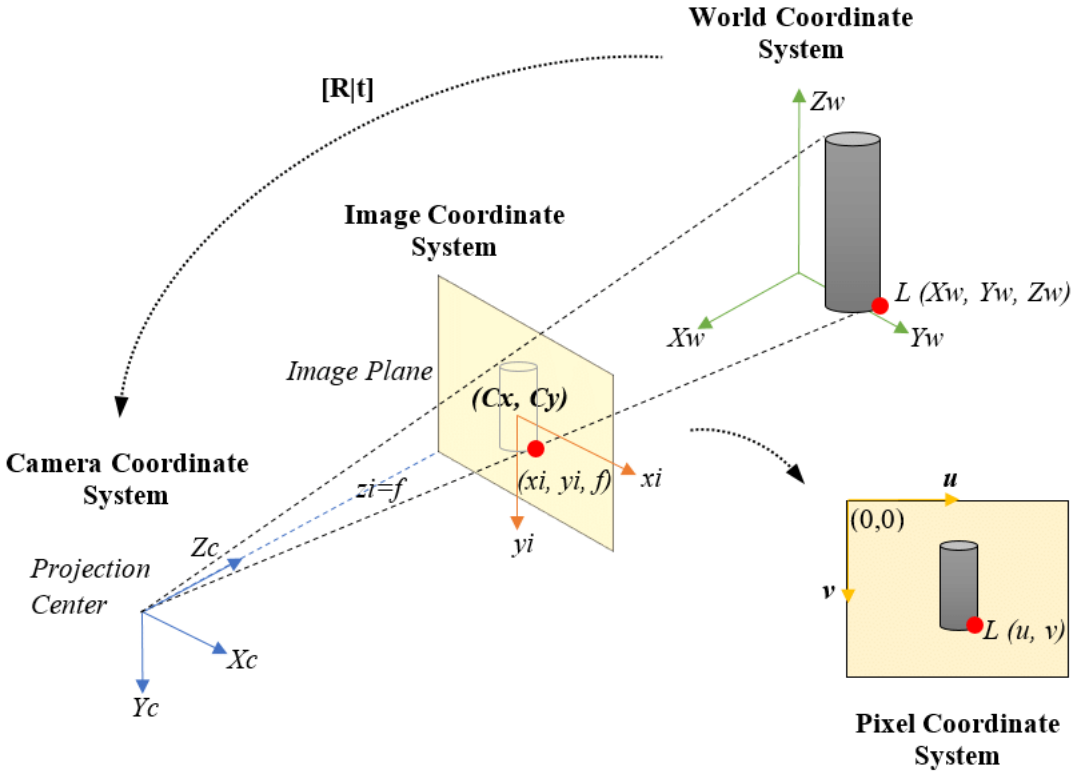


Figure 3.7: A pinhole camera model where the image plane is located in front of the camera's origin. A non-inverted image is formed on the plane at $zi = f$ [10]

In order to capture 3D images and store them in 2D formats, cameras play an instrumental role. Our project does not require a detailed explanation of the physics of the pinhole camera model since explaining the mathematics behind it will suffice. The 3D world is projected onto a plane (the projection plane) in front of the camera center in this model. The focal length is the distance between the camera center and the projection plane. Using the following formula, any point in the 3D world can be projected onto the projection plane of the camera:

$$m' = A[R \mid t]M'$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Here, (X, Y, Z) indicate the coordinates of the point in the 3D world coordinate frame and (u, v) indicate the coordinates of the projection point in pixels. Camera matrix A is defined as a matrix of intrinsic parameters (internal to the camera setup, they determine the positioning and

orientation of the camera in relation to the world frame), while matrix $[R \mid t]$ is defined as a matrix of extrinsic parameters (external to the camera and may vary with respect to the world frame, allowing a mapping between camera coordinates and pixel coordinates in the image frame). (c_x, c_y) represents the center of the image (in pixels) and is sometimes referred to as the principal point. The focal lengths (f_x, f_y) are also expressed in pixel units.

3.6.1 Hand Eye Calibration

Hand-eye calibration is the process of calculating the relative 3D position and orientation between the camera and the robot arm in an eye-to-hand configuration, in which the camera is stationary within the robot's environment. It involves calculating the relative rotation and translation (homogeneous transformation) between two coordinate frames, one centered at the camera lens center, and the other at the robot arm center. Figure 3.8 illustrates the relationships between the various components of a vision-guided robot. It is necessary to obtain all the relationships of an object based on the robot base frame in order to identify that object completely. From the kinematic model of the robot, it is possible to determine the relationship between the robot base and the end-effector, while from the calibration of the camera, it is possible to determine the relationship between the camera and the environment. It is therefore necessary to compute a relationship between the camera and the robot hand.

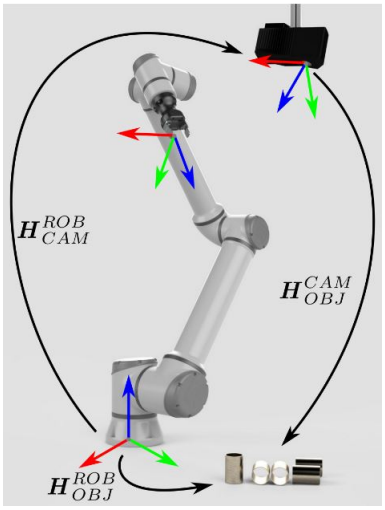


Figure 3.8: A hand-eye calibration estimates the position of the camera relative to the robot's base H^{ROB}_{CAM} in eye-to-hand systems. The pose circle can be closed by calculating one pose from the other poses. In this case, the position of the object in relation to the robot. This can be determined by multiplying the pose of the camera relative to the robot with the pose of the object relative to the camera: $H^{ROB}_{OB,J} = H^{ROB}_{CAM} \cdot H^{CAM}_{OBJ}$ [11]

An illustration of the problem can be found in Figure 3.9, where B represents the transformation between the coordinate frame of the hand and the base of the robot. The transformation between the camera coordinate frame and the base coordinate frame is represented by Z, while the transformation between the camera and tool coordinate frame is represented by A. Lastly, X represents the transformation between the hand and the tool frame.

The problem can be formulated as $AX = ZB$ assuming that the transformation B is known and the transformation A can be calculated by calibration software. In this case, X and Z are unknown. AX represents a transformation from the robotic hand to the camera via the calibration object (tool), while ZB represents a transformation from the robotic hand to the camera via the

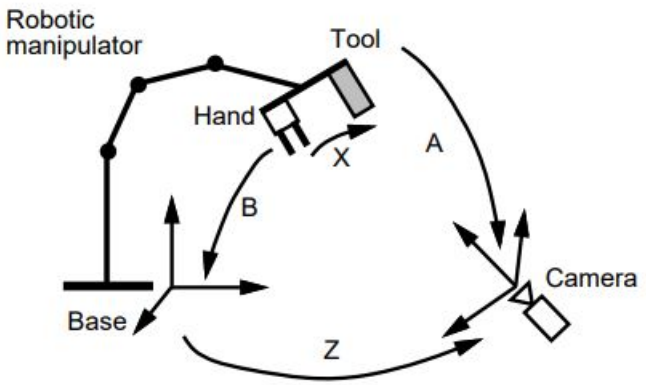


Figure 3.9: Hand-to-eye calibration. Cameras are used to observe tool feature points mounted on the gripper in order to determine the tool's motion [12]

robot base. $AX = ZB$ can be solved by passing a series of matrices A_i and B_i that contain the relevant transformations for image i, and solving X and Z as follows:

$$\begin{aligned}
\mathbf{A}_i\mathbf{X} &= \mathbf{ZB_i} \rightarrow \mathbf{Z} = \mathbf{A}_i\mathbf{X}\mathbf{B}_i^{-1} \\
\mathbf{A}_{i+1}\mathbf{X} &= \mathbf{Z}\mathbf{B}_{i+1} \\
\mathbf{A}_{i+1}\mathbf{X} &= \mathbf{A}_i\mathbf{X}\mathbf{B}_i^{-1}\mathbf{B}_{i+1} \\
\mathbf{A}_i^{-1}\mathbf{A}_{i+1}\mathbf{X} &= \mathbf{X}\mathbf{B}_i^{-1}\mathbf{B}_{i+1}.
\end{aligned}$$

As this set of equations corresponds to solving $AX = XB$, where $A = \mathbf{A}_i^{-1}\mathbf{A}_{i+1}$ and $B = \mathbf{X}\mathbf{B}_i^{-1}\mathbf{B}_{i+1}$, this is called a Sylvester equation and it can be solved numerically or methodologically with respect to X [53].

Chapter 4

Method

We will present our work and findings in this chapter while pursuing an RBP vision-guided system, as well as some of the choices we made while conducting our research. As shown in Figure 4.1, a simplified pipeline for the proposed system is depicted. In the Start state, hardware and software are initialized, before a depth image is acquired in the Image Acquisition state. In the Grasp Prediction state, a deep neural network is used to locate the most suitable grasping pose based on the location of the object in the depth image. In the Robot Manipulation state, a robot performs the picking operation based on the grasping location. After completing the pick, the series of actions may be repeated for a new object, or the system may exit.

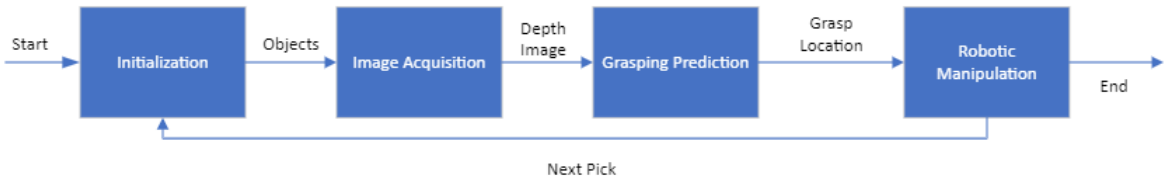


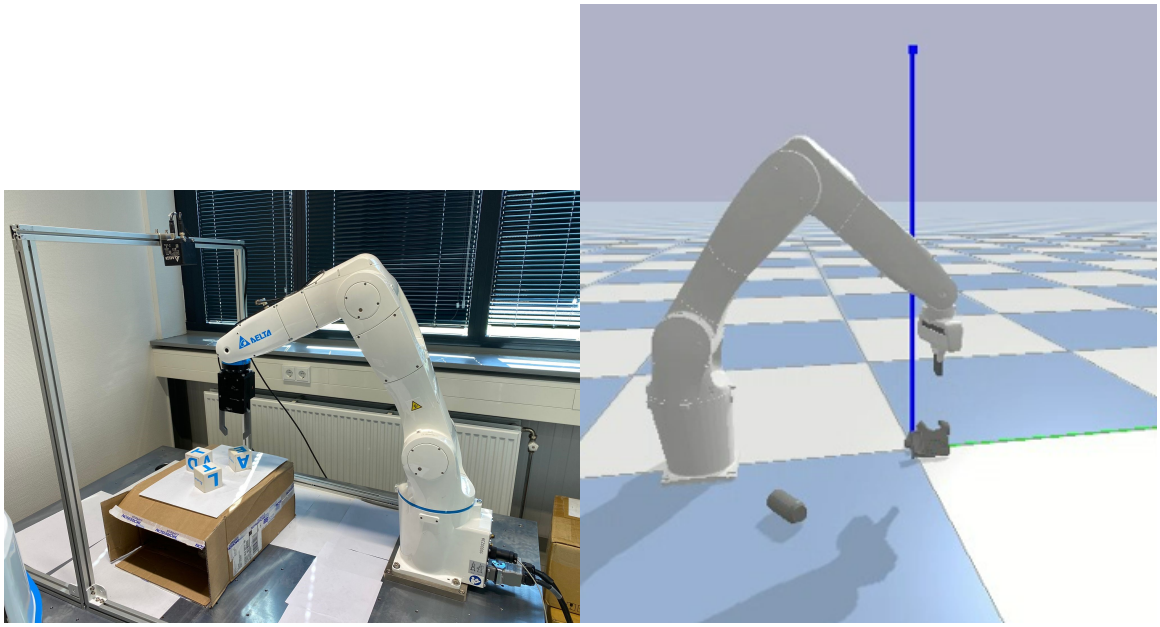
Figure 4.1: An overview of our proposed RBP system. A number of grasping objects are placed in the Initiation state. A depth image from the Image Acquisition state will be used in the Grasping Prediction state. The proposed grasping location will be sent to the Robotic Manipulation state, where the final picking action will be performed.

4.1 Tools and Equipment

In the FAE lab at Delta Electronics, practical work was conducted in a small robot cell which consisted of the following hardware:

- DRV90L, six axis robot arm
- 3D ToF camera
- AX-8 motion controller
- ASDA-A2 servo drives
- master computer
- custom made gripper module

Figure 4.2 displays the physical setup of the robot cell, as well as a simulated model in Pybullet.



(a) (b)

Figure 4.2: An illustration of the physical robot cell and the simulation environment. (a) Robot cell for real application. (b) A simulation of a robot cell in Pybullet

4.1.1 Robot arm

In this graduation project, a Delta DRV90L is used. DRV series robots are composed of six axes, J1 to J6; their motion direction is determined by the joint coordinates. The (+) and (-) shown in Figure 4.3 represent the actual direction of motion for each axis. On the robot arm, a signal connector is used to connect the parallel gripper. The parallel gripper wires are connected to pins 1-12 of the 24Pos circular connector behind the base of the robot and to the 12Pos signal circuit from the robot's J4 axis wrist.The wiring diagram can be found in Section 4.1.5. A technical description of the robot arm and its axes of rotation can be found in Figure 4.3, as well as an illustration of the robot and its axes of rotation.

4.1.2 AX-8 motion controller

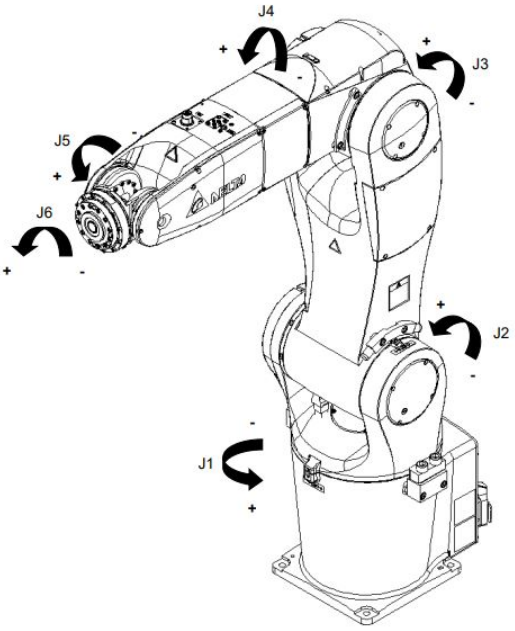
The AX-8 Series utilizes the CODESYS motion control platform to enable simple programming for complex motion control settings. The device supports EtherCAT communications for high speed motion control, as well as OPC UA for connecting other equipment or software. The robot arm in this thesis is controlled by an AX-8 Windows-based PLC. It is connected to six servo drives that correspond to the six axes on the robot arm to achieve motion control. Motion planning is performed using the CODESYS robotics library, which will be discussed later in this chapter. A diagram of the controller can be found in Figure 4.4.

4.1.3 Servo drives

Delta introduced the high-performance motion control ASDA-A2 series servo motors and servo drives in 2009 in order to satisfy the demanding requirements of motion control applications in

Model		DRV70L7D/E Series	DRV90L7D/E Series
Number of axes		6	
Arm length		710 mm/27.55 in	900 mm/35.43 in
Rated / maximum payload		1 kg / 7 kg	1 kg / 7 kg
Operation range (°)	J1	±170	
	J2	+125 / -95	+125 / -95
	J3	+58 / -198	
	J4	±190	
	J5	±120	
	J6	±360	
Maximum speed (° / sec)	J1	450	370
	J2	340	260
	J3	510	410
	J4	550	550
	J5	550	550
	J6	820	820
Maximum Synthetic speed		11,000 mm/sec	10,060 mm/sec
Maximum allowable load inertia (kg*m²)	J4	0.47	
	J5	0.47	
	J6	0.15	
Load torque (Nm)	J4	16.6	
	J5	16.6	
	J6	9.4	
Repeatability		±0.02 mm	±0.03 mm
Standard cycle time*1		0.32 sec	0.35 sec
Installation method		counter, hanging, wall mount*2	
The robot arm has the followings built-in		One 12 Pos connector for customers to connect sensors and 2 ø6 straight air pipes (one provided for the built-in 3 sets of solenoid valves and one for expansion)	
IP Level		IP40 for DRV70/90L7D series; IP65 for DRV70/90L7E series	
Environmental specifications	Temperature	0ºC to +40ºC	
	Humidity	20 to 85%RH	
Shock		0.5G	
Robot arm weight		36 kg	38 kg
Controller type		DCV-2J00-CA	
Phase / Voltage		220V~+PE · 15A · 50/60Hz	
Rated power		3.3 kW	

(a)



(b)

Figure 4.3: Technical specification of the robot arm extracted from [13]. (left)Articulated robot specification table. (right)DRV series motion directions

Figure 4.4: The AX-8 Windows-based PLC was used in the pratical work of this thesis [14]

industrial automation as well as the needs of machine designers and system integrators for high-precision positioning control [14]. For precise motion control, we used six ASDA-A2 servo drives corresponding to encoders built into the robot arm. An illustration of the servo drive can be seen in Figure 4.5.

4.1.4 3D TOF camera

In our system, we used a 3D ToF camera DMV-TI3000, which is a high-performance 3D camera for industrial and robotic applications. Table 4.1 contains the technical specifications. The camera in our system was mounted on top of the robot arm. In addition to the internal software, the camera is capable of obtaining depth and amplitude images through any universal software that supports the GigE standard. For the purposes of this thesis, we are using Harvesters API to acquire the depth images through a Python file on the master computer.

Figure 4.5: ASDA-A2 servo drive with motor [15]

Model	DMV-TI3000GSM
Sensing Technology	iToF(indirect Time of Flight)
Resolution(pixel)	640x480
Sensor	Sony IMX556 TOF image sensor
Pixel size (um)	10
Frame rate (fps)	60
FOV	67 x 51
Recommend Working Range (m)	6

Table 4.1: Technical specifications for 3D ToF camera

4.1.5 Custom made gripper

We have made a custom gripper that is a parallel gripper using RS-485 for communication. It is manufactured by a third-party company using 3D printing and customized PCB boards. According to Figure 4.6 below, the gripper is powered by a 24V power supply and is connected to AX-8 via two RS-485 signal wires. Appendix A contains the specific operation manual. CODESYS was used to control the gripper via MODBUS commands.

4.1.6 Master computer

Master computers were mainly used to run image acquisition and neural networks for grasp prediction, as well as receiving and transmitting signals to other components of the system. The computer was a Linux-based computer with 32 GB RAM, and an Intel Core i7-9700 processor running at 3.00GHz x8. The graphic card was NVIDIA Quadro RTX 4000.

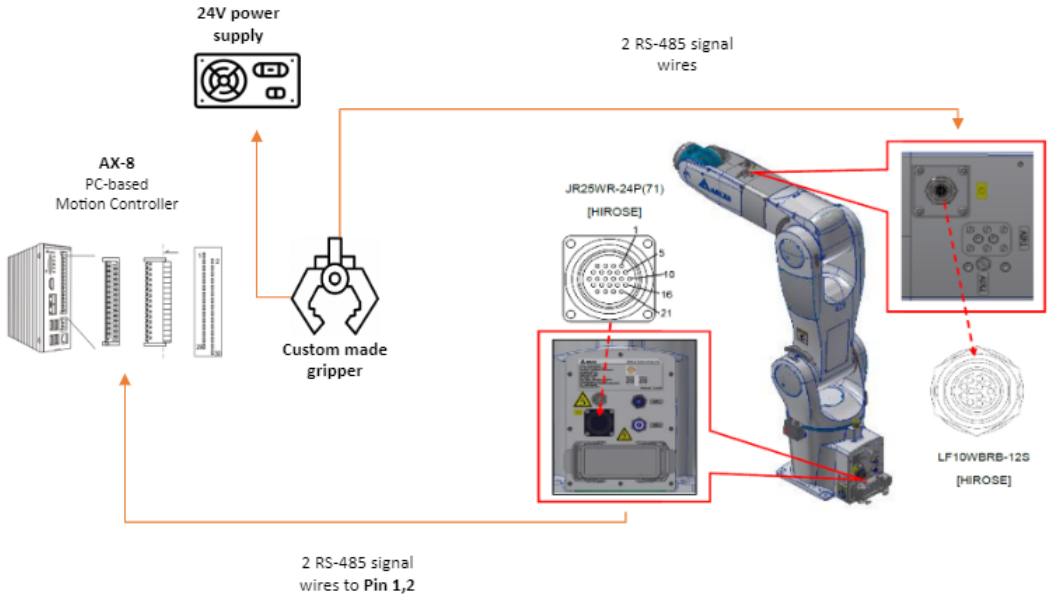


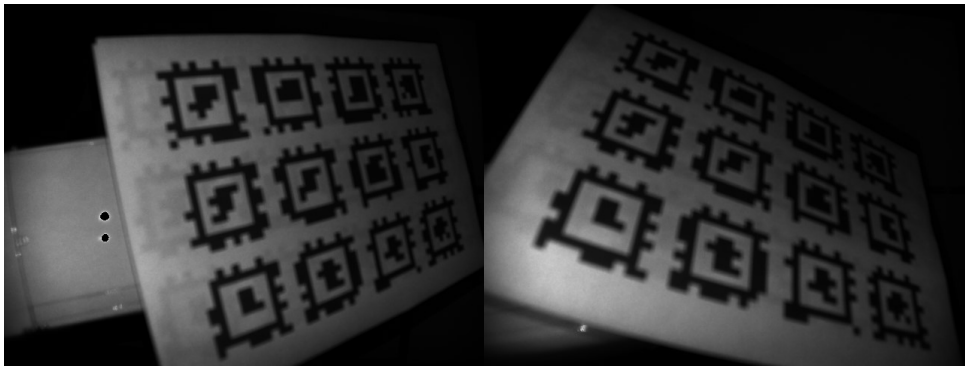
Figure 4.6: Diagram of the wiring between the robot, AX-8 motion controller and the custom-made gripper

4.2 Hand-to-eye calibration

It is necessary to perform hand-eye calibration since a 3D camera system is used for bin picking. In order to increase the accuracy of the calibration, the robot is controlled to move to 20 different locations while an image of a standard calibration pattern is taken, along with the robot's location. Figure 4.7 illustrates two examples of the images.
Extrinsic parameters indicate the camera's position and orientation in the 3D environment. Mathematically, the position is defined by a 3x1 translation vector and the orientation by a 3x3 rotation matrix. The calibration was performed using the MATLAB calibration toolbox with AprilTag calibration pattern.

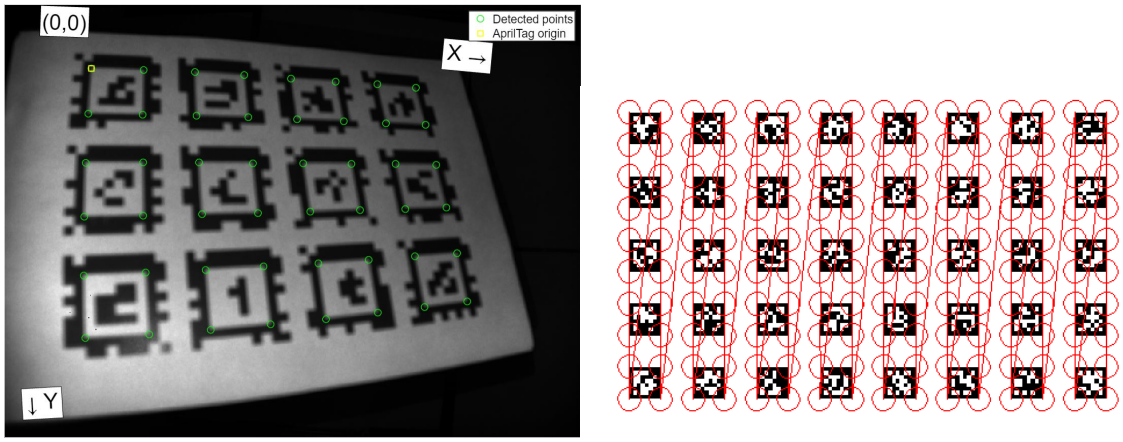
AprilTag is a visual identification system that can be detected and recognized by a camera in order to provide special information regarding certain tags. An identifier associated with the tag provides information regarding the tag's position and orientation. The AprilTag barcode can be detected even if the image is captured at a low resolution, and it can also be detected at a longer range than other 2D barcodes, such as QR codes [54]. In the captured image, the tag can be identified by detecting a four-sided region known as quads. When an AprilTag is scanned by a camera, it provides different information that is used during the hand-to-eye calibration process. When an AprilTag is decoded, the center and corners of the tag can be calculated, as well as the family and ID. An example of how AprilTag is read by MATLAB **helperDetectAprilTagCorners** function and how data is generated is shown in Figure 4.8.

It is necessary to provide the calibrator function with known $T\_gripper2base$ and $T\_target2cam$ transformations in order to compute the relative transformation between the camera and the base. In this particular case, $T\_gripper2base$ represents the dynamic transformation from the robot base to the end effector frame. An AprilTag marker and **estworldpose** function are used to calculate the $T\_target2cam$ transformation, which represents the dynamic transformation between the



(a) (b)

Figure 4.7: Two examples of calibration pattern images taken by the 3D ToF camera in the amplitude mode



(a) (b)

Figure 4.8: AprilTag detection using MATLAB calibration toolbox. (a) detected points shown in real images. (b) demonstration of MATLAB's detection function on a generated pattern

camera and AprilTag frames. Figure 4.9 illustrates the calibration process in a simple schematic. Using the two known dynamic transformations acquired in different poses, this calibration calculates the desired static transformation.

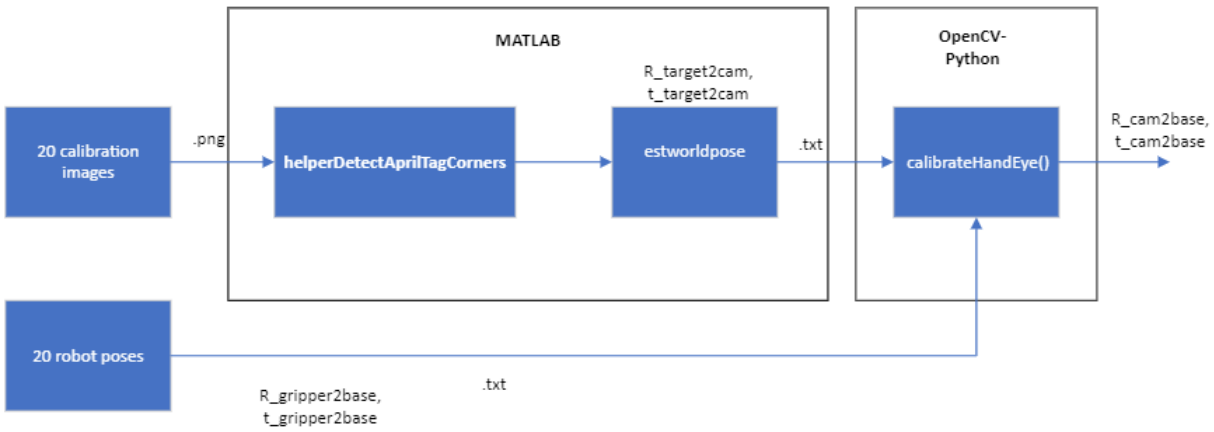


Figure 4.9: An illustration of the hand-to-eye calibration process. A total of 20 different images containing calibration patterns are imported into MATLAB. To get the final result of camera position and orientation, the output of the function along with the 20 different robot pose information is fed into OpenCV-Python function calibrationHandEye()

4.3 AI Algorithm

Traditionally, RBP systems are customized to meet the needs of each situation, as discussed in Section 1.1.1. The purpose of this thesis is to explore the possibility of applying deep learning to Delta's existing products with more generalized solutions. We concluded, as discussed in the previous two chapters, that we would base our planar grasp algorithm on GG-CNN, our 6DoF grasp algorithm on PointNetGPD and o6DoF graspnet. In the grasp prediction module, Python was used to implement a decentralized neural network approach based on the PyTorch framework. In the following sections, we explain the algorithm and dataset used for training the neural networks, as well as how we improved their performance.

4.3.1 Datasets

Cornell Grasping Dataset

Cornell Grasping Dataset (CGD) from [16] is a popular grasp dataset that has been used for most transfer learning approaches in robotic grasping [55][33][56]. CGD contains grasp rectangle information for 240 different types of objects and contains 885 RGB-D images, 885 point clouds, and 51100 human-labeled positive and 2909 negative grasp rectangles. Figure 4.10 illustrates a sample set of images. There have been a number of research studies concerning the CGD in the recent past, which suggests that it contains a reasonable amount of examples of generalized grasps [33].

Jacquard Dataset

The Jacquard dataset [17] contains 54485 different scenes featuring 11619 distinct objects with a total of 4967454 grasps annotations (1181330 unique locations). Each scene includes a rendered RGB image, a segmentation mask, two depth images, and grasp annotations. An example of annotated images from the Jacquard dataset can be found in Figure 4.11 . Since this dataset

Figure 4.10: Sample of Cornell Grasp dataset [16]

includes a variety of grasping positions on different objects, it can be viewed as an extension of CGD.

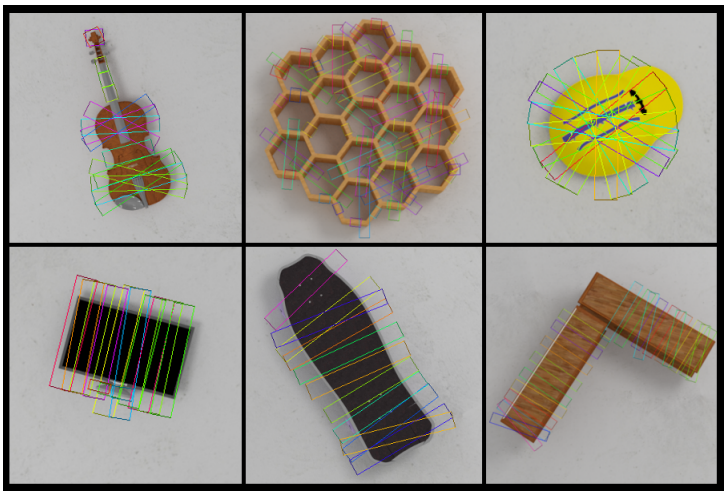


Figure 4.11: A large variety of objects are included in the Jacquard dataset, each of which has multiple labeled grasps on realistic images. On the image, grasps are represented by 2D rectangles whose darker sides indicate the position of the jaws [17]

Yale-CMU-Berkeley object and model set

Yale-CMU-Berkeley (YCB) object and model set [18] [57] contains mesh models, RGB, RGB-D, and point cloud images of over 80 objects. The data are collected using two state-of-the-art systems: UC Berkley's scanning rig and Google's scanner. This scanning rig data includes meshes generated using Poisson reconstruction as well as meshes generated using volumetric range image integration, textured versions of both meshes, Kinbody files that can be used in OpenRAVE with the meshes, 600 high-resolution RGB images, 600 RGB-D images, and 600 point clouds for each object. As part of the Google scanner data, 3 meshes are provided, each with a different resolution (16k, 64k, and 512k polygons), as well as textured versions of the meshes and Kinbody files for use with OpenRAVE. The following is a sample set of images in Figure 4.12. It is important to note that in this project, we selected the 512k laser scan option.

Figure 4.12: Food items included in the YCB object set. Back row, from left to right: a can of chips, a coffee can, a cracker box, a box of sugar, and a can of tomato soup. From left to right, middle row: mustard container, tuna can, chocolate pudding box, gelatin box, and potted meat can. An apple, a lemon, a pear, an orange, a banana, a peach, strawberries, and a plum are displayed in the front row [18]

4.3.2 Planar grasp

GG-CNN

The Generative Grasping Convolution Neural Network (GG-CNN) [19] is a fully convolutional neural network capable of generating an antipodal grasp pose and quality measure for each pixel in an input depth image. It is fast enough to allow for closed-loop control of grasping in dynamic environments. There are four separate outputs from this model, each of which is based on a depth-only image (no color information):

- Grasp Quality: normalized between [0,1], it represents the probability of successful grasp of the pixel
- Angle: the angle is represented as points in the unit circle by two images, one for each components, with value between [-1,1]
- Width: also given in normalized units and remapped to [0,150] pixels.

The output of the network is post-processed in order to facilitate the generation of grasps. The angle is computed from the two components as $\frac{1}{2}arctan\frac{y}{x}$, so that the values fall within the range $[-\frac{\pi}{2}, \frac{\pi}{2}]$. Using the smoothed grasp quality distribution, the best local maxima are found to determine the angles and widths of the best grasp. It is then possible to transform the image-space grasp representation into world-space coordinates by knowing the intrinsic and extrinsic parameters of the camera, as well as the depth from the input image.

Grasp point definition

GG-CNN defines $\mathbf{g} = (\mathbf{p}, \phi, w, q)$ as a grasp that is executed perpendicular to the x-y plane. This is determined by the gripper's pose, namely its center position $\mathbf{p} = (x, y, z)$ in Cartesian coordinates, its rotation ϕ around the z axis, and its width w. The quality measure q represents the

probability that the grasp will be successful.

An image of depth $I = R^{H \times W}$ with height H and width W, acquired from a camera whose intrinsic parameters are known. Image I illustrates a grasp as $\tilde{g} = (s, \tilde{\phi}, \tilde{w}, q)$, where $s = (u, v)$ represents the center point in image coordinates (pixels), ϕ represents rotation in the camera's reference frame, and w represents width in image coordinates. By applying a sequence of known transformations, a grasp in image space $\tilde{g}$ can be converted to a grasp in world coordinate g.

Architecture

As shown in Figure 4.13, GG-CNN is a fully convolutional topology. There are 62,420 parameters in the GG-CNN, making it significantly smaller and faster to compute than the CNNs used for grasp candidate classification in other works, which can contain hundreds of thousands [49][58] or millions [59][3][60] of parameters. A total of 3 transposed-convolutional layers are applied after the initial 3 convolutional layers in order to achieve a dense prediction for each pixel in the original input image.

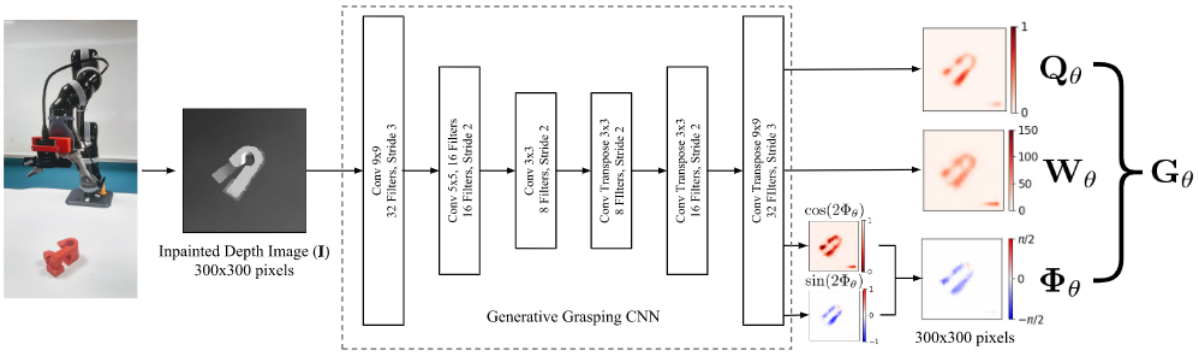


Figure 4.13: The GG-CNN architecture [19]. It is the last convolutional layer that generates the grasp and width images as well as the two angle images. The best grasp is separated from the quality distribution.

The grasp detection pipeline consists of three stages: image processing, evaluation of the GG-CNN, and computation of the grasp pose. In order to suit the input of the GG-CNN, the depth image is first cropped to a square and scaled to 300 x 300 pixels. In order to correct invalid depth values, we use OpenCV inpainting function [61]. To enable to produce the grasp map G_θ, the GG-CNN is evaluated on the processed depth image. We filter Q_θ using a Gaussian kernel and remove outliers to achieve convergence on more robust grasp regions. Lastly, the best grasp pose in the image space $\tilde{g}^*_\theta$ is determined by identifying the maximum pixel $\mathbf{s}^*$ in $\mathbf{Q}_\theta$, and the rotation and width are determined by $\Phi_\theta|_{\mathbf{s}^*}$ and $\mathbf{W}_\theta|_{\mathbf{s}^*}$, respectively. By using sequence of transformations, we are able to calculate the grasp in Cartesian coordinates.

Evaluation

In the CGD, antipodal grasps are represented as rectangles aligned to the position and rotation of a gripper. The center third of each grasping rectangle is used as an image mask to convert from the rectangle representation to our image-based representation G. To train our network, we only consider positive labeled grasps and assume any other area is not a valid grasp.

In order to evaluate the performance of the network, the GG-CNN uses the rectangle metric proposed by Jiang [62]. Two conditions must be met for a grasp to satisfy the proposed rectangle metric:

- An intersection over union (IoU) score of more than 25% exists between the ground truth grasp rectangle and the predicted grasp rectangle
- An offset of less than 30 degrees exists between the predicted grasp rectangle and the ground truth rectangle

Network modification

In Section 3.3.3, we discussed residual blocks, and we want to develop a net model using GG-CNNs combined with residual blocks to increase the accuracy of the network. Figure 4.14 illustrates the proposed GR-ConvNet [20] model, which is a generative architecture that generates three pixel-wise grasps from an n-channel input image. In order to generate the four output images, the n-channel image is passed through three convolutional layers, followed by five residual layers and three convolution transpose layers. Convolutional layers extract features from the input image. Five residual layers are applied to the output of the convolutional layer. As we know, accuracy increases as layers are added. This is not true, however, when you exceed a certain number of layers, resulting in vanishing gradients and dimensionality errors, which cause saturation and degradation of accuracy. Through using skip connections, residual layers can help us learn identifier functions more effectively. The image size is reduced to 56x56 after passing through convolutional and residual layers, making it difficult to interpret. Similarly to GG-CNN, we up-sample the image using a convolution transpose operation after convolution to make it easier to interpret and retain spatial features. Thus, the output image is the same size as the input image.

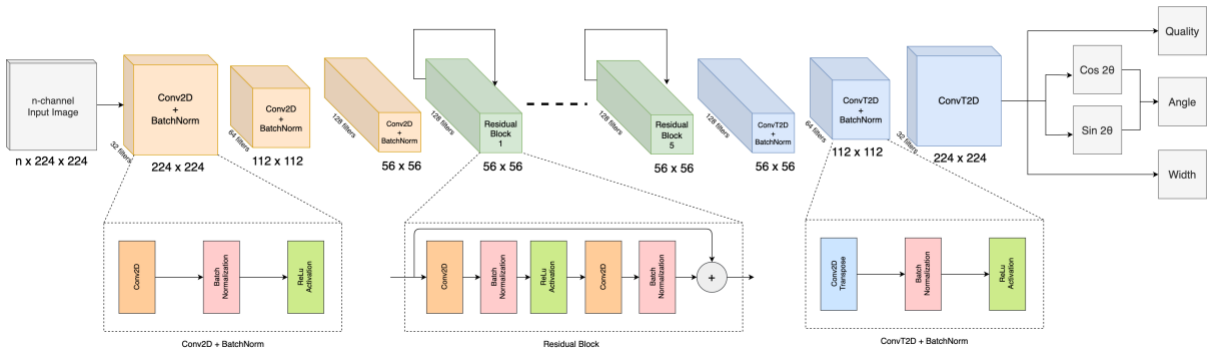


Figure 4.14: Proposed Generative Residual Convolutional Neural Network [20]

In the last section, we mentioned that GG-CNN uses rectangle metrics for evaluation. In order to calculate this metric, a grasp rectangle representation is required, but our model predicts an image-based grasp representation. The value corresponding to each pixel in the output image needs to be mapped to its equivalent rectangle representation in order to convert from image-based grasp representation to rectangle representation. As a result, we propose another method of evaluation that does not require conversion and requires the following two conditions:

- A difference of less than 30 degrees exists between the predicted grasp angle and the labelled grasp angle
- The grasp center point is less than five pixels from the ground truth point

For the convolutional layers, the useful information in the image is concentrated in the middle (where the objects are), which leads to sparse and unbalanced classes. Instead of normal L1,L2 loss, we introduce focal loss as a modified loss function. In [63], it is demonstrated that focal loss is more effective when the classes are imbalanced. Predicting the grasp region can be regarded as a binary classification problem. The loss function is followed by a binary cross-entropy function, which is defined as:

$$L = -\frac{1}{N}\sum_{n=0}^{N}\left[y_q^n \cdot \log\left(p_q^n\right) + \left(1 - y_q^n\right) \cdot \log\left(1 - p_g^n\right)\right]$$

N represents the size of the output feature maps, p_q^n represents the probability predicted at the n position, and y_q^n represents the corresponding label.

4.3.3 6DoF grasp

PointNetGPD

PointNetGPD [21]is an end-to-end grasp evaluation model that addresses the challenging issue of identifying robot grasp configurations directly from point clouds. It is capable of capturing the complex geometric structure of the contact area between the gripper and the object even when the point cloud is sparse. The algorithm flow is shown in Figure 4.15.

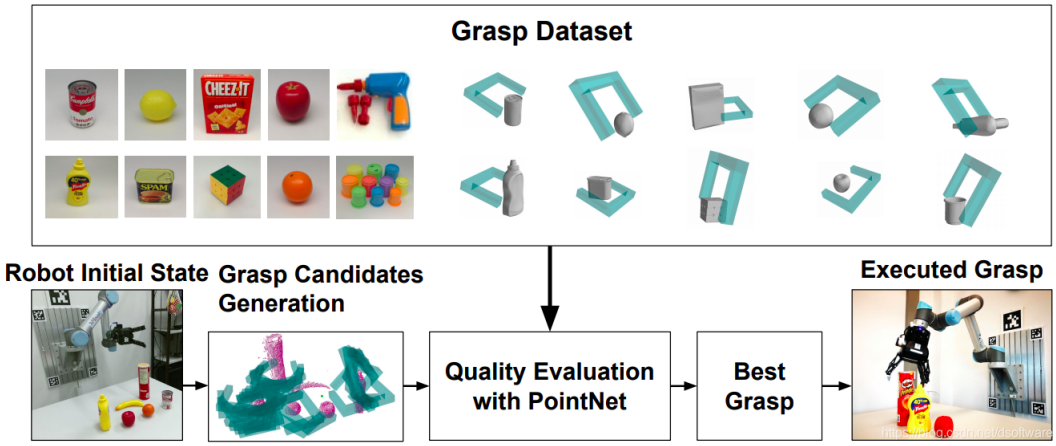


Figure 4.15: PointNetGPD algorithm flow [21]. Given raw RGB-D data from a sensor input, the depth map is first converted into a point cloud. Next, some candidate grasping poses are sampled based on geometric constraints. For each candidate, the point cloud inside the grasper is cropped and converted to the gripper's local coordinate system. Lastly, the candidate grasping inputs are fed into the grasping quality evaluation network to determine scores, and the candidate grasping poses with the highest scores are adopted.

On the basis of the YCB dataset, the authors generated a dataset with 350k real point clouds, parallel gripper grasping poses, and resolved grasping quality scores. As opposed to the Dex-Net[3] dataset, this dataset provides a more detailed score for each gripper pose. The authors calculate force-closure [64] and friction-independent GWS (Grasp Wrench Space) [65] as the score for the gripping pose, given a 6D grasp pose and a CAD model of the object. Figure 4.16 illustrates the grasp quality evaluation network. In this network, input grasps are represented as point clouds within the closed region of the gripper, without using the entire point cloud as input, which can improve the efficiency of learning and inference. Figure 4.17 illustrates how the point clouds are transformed into a uniform gripper local coordinate system, which eliminates the ambiguity of

grasp locations resulting from different experimental setups. The point clouds are then fed into the network in order to estimate the grasp quality hierarchy. Compared with other CNN-based networks used for grasp estimation, this network is lightweight and has only approximately 1.6 million parameters.

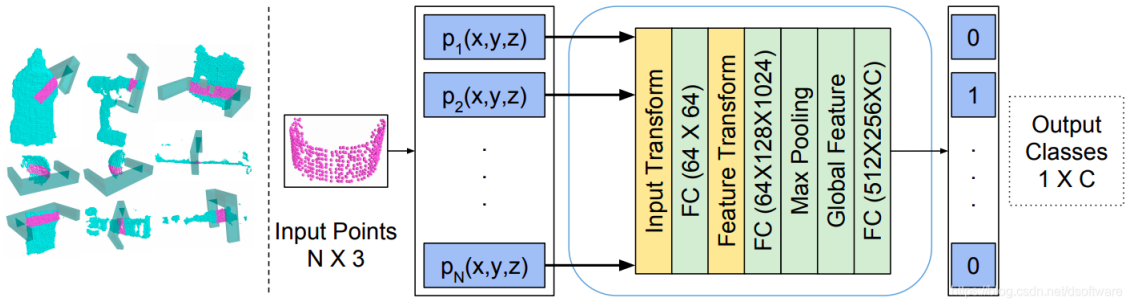


Figure 4.16: PointNet-based network structure for grasp quality evaluation [21]. Based on the grasp pose and the original point cloud, the point cloud in the closed region of the grasper represents a grasp that is transformed into the grasper's local coordinate system and input into the network. As a result of multiple spatial transformations and feature extraction, the final global features are used to classify the input grasps according to their quality level.

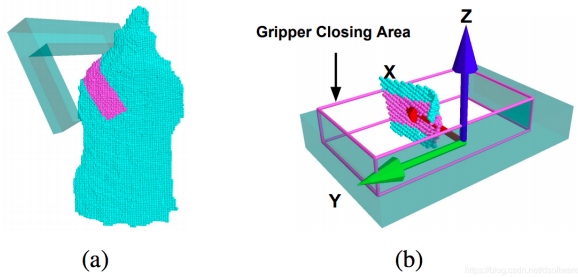


Figure 4.17: Representation of a grasp in the local gripper coordinate system [21]. (a) a typical grasping pose; (b) the axes of the local coordinate system. The authors use the forward, parallel, and orthogonal directions of the gripper as the XYZ axes and the middle position of the bottom of the gripper as the origin, respectively.

6DoF GraspNet

NVIDIA has developed another 6DoF grasp neural network structure, called 6DoF graspnet [5]. It is presented in this paper that a set of grasps are sampled using a variational autoencoder, and the sampled grasps are then evaluated and fine-tuned for refinement through the use of a grasp evaluator model. A 3D point cloud observed by the depth camera is taken as input by both the grasp sampler and grasp refine network. Figure 4.18 illustrates the overall network structure.

The grasp sampling network consists of a variational autoencoder, or VAE. In the input, X is the target point cloud from each viewpoint sampled from the original target 3D point cloud, and g is the grasping pose, which is the R (rotation) and T (translation) of the gripper in the target coordinate system. In order to obtain a similar g to the input, the encoder Q of the VAE encodes the input into the hidden layer space, so that it meets the unit Gaussian distribution; then, after decoding the hidden layer variable z, $\hat{g}$ similar to the input is obtained. It is the purpose of the

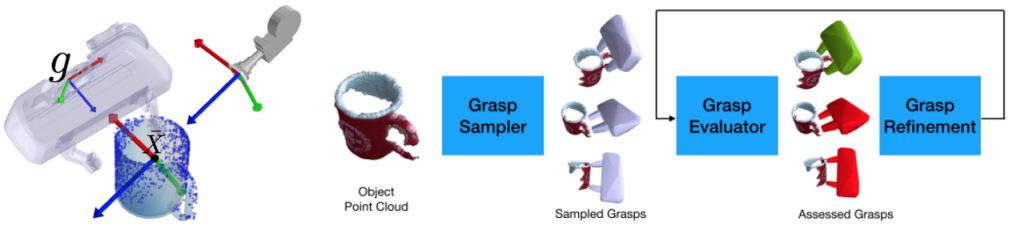


Figure 4.18: (left) The grasps are estimated with respect to the center of mass of the object point cloud, X. The axes of the grasp coordinate frame are parallel to those of the camera. (right) An point cloud X is obtained by fitting a plane to a depth image. The Grasp Sampler Network uses the point cloud to propose different grasps. Based on the object point cloud and the proposed grasp, the evaluator network assesses the grasps. By using the gradient of the evaluator network, grasps are improved iteratively [5]

VAE training process to make z obey as closely as possible the unit Gaussian distribution described above, so when testing, the encoder is removed and a random sample of the unit Gaussian distribution replaces the hidden layer variable z that is encoded in order to obtain the input point cloud X, resulting in the reconstructed grasp $\hat{g}$ that corresponds to the network's requirements. The process is illustrated in Figure 4.19.

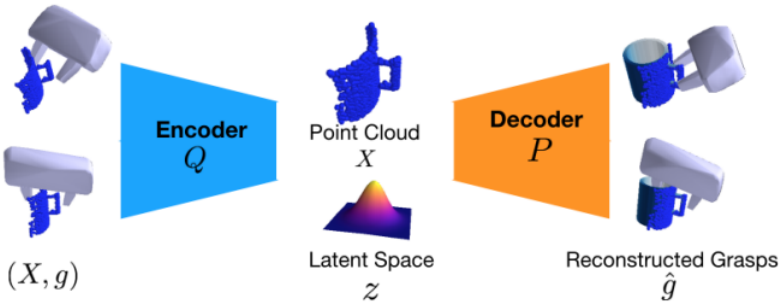


Figure 4.19: Each grasp is mapped to a point z in a latent space during training. Latent space distribution is minimized toward a normal distribution. Using the point cloud and latent values, the decoder reconstructs the 6D grasps, depicted here as gripper poses [5]

Since the grasp generated in the previous step must appear correct to the network (because z obeys the unit Gaussian distribution, then the g reconstructed by sampling from the unit Gaussian distribution must be the correct grasp), a judgment is actually required to determine whether the generated grasp appears plausible. As a result, the authors in [5] add a grasp pose evaluation network after the grasp sampling network. This evaluation network is essentially a binary network, where the input is the synthetic rendered point clouds $X \cup X_g$ of the target and gripper, and the output is the success rate s. The grasp evaluation network is optimized using cross-entropy loss:

$$\mathcal{L}_{\text{evaluator}} = -(\text{y}\log(\text{s}) + (1-\text{y})\log(1-\text{s}))$$

Following the evaluation, some examples of successful and unsuccessful grasps were obtained, but how can the success rate be improved? What can be done to improve the accuracy of the estimated grasps $\hat{g}$? An ingenious approach called iterative grasp pose refinement has been devised

by the authors [5] in order to achieve this goal. As a larger s in the evaluation network indicates a greater likelihood of success, making all these s as large as possible and converging to 1 will also improve the grasps. In order to compute the refinement transformation that results in the greatest improvement in success probability, the derivative of success with respect to the grasp transformation should be taken: $\partial S/\partial g$.

Combination

We have discussed above that PointNetGPD uses traditional methods to sample grasps, such as force-closure and GWS. For the purpose of generating and evaluating grasps, they randomly sample the grasps from the given mesh model and each time they must calculate scores, which is time consuming to prepare this dataset for grasp prediction networks. In contrast, in 6DoF graspnet, all sampling and evaluating is performed by VAE. Our proposal is to combine the two networks and select the advantages of each network to achieve a more efficient network for 6DoF objects grasping. The point cloud method from PointNetGPD can continue to be used for the input of the grasp prediction network.

4.4 Software Development

In this project, we wrote the majority of the software for the system in Python, but some parts were written in MATLAB and CODESYS. Programming in Python was used to control the grasping prediction module. A Python API was used to control the 3D camera, while MATLAB was used to calibrate the camera. The robot manipulation module on the AX-8 controller was implemented using CODESYS SoftMotion+CNC robotics library.

4.4.1 Simulation environment setup

Pybullet

In this project we used Pybullet [66] as one of the simulation environments. The Pybullet Python module is a fast and easy-to-use tool for robotics simulation and machine learning. An articulated body can be loaded from files in URDF, SDF, MJCF, and other formats. These files contain information about the structure of a robot, the connections between its various components, etc. This project uses the URDF model provided by the company's research and development department.

In addition to forward dynamic simulation and inverse dynamic computation, Pybullet also supports collision detection and ray intersection queries. It is not only designed for grasping, but also for general simulations. Since Pybullet does not accurately model friction, grasping objects based on physics can be quite challenging. In contrast, the simulation can detect collisions fairly easily, so if the gripper collides with an object during grasping, we can attach the object to the end effector to simulate a grasp rather than relying on physics.

IsaacGym

Isaac Gym [67] provides a high performance learning platform for training robotics policies directly on GPUs. The physics simulation and neural network policy training are both executed on GPUs, and data is passed directly from physics buffers to PyTorch tensors without ever passing through any CPU bottlenecks. In view of the complexity of the model and the efficiency of the simulation environment, Isaac gym will be used to simulate and test 6 DoF grasp.

4.4.2 Image Acquistion

A Python program was used to control the 3D ToF camera and acquire the depth images. An initial connection was established between the computer and the camera, and the pre-calibrated

camera settings were defined. Figure 4.20 shows the specific camera settings for our setup. The 3D ToF camera records the captured data as an array containing all the information it has obtained. As a result, it is possible to post-process the image in order to obtain the desired depth and amplitude images.

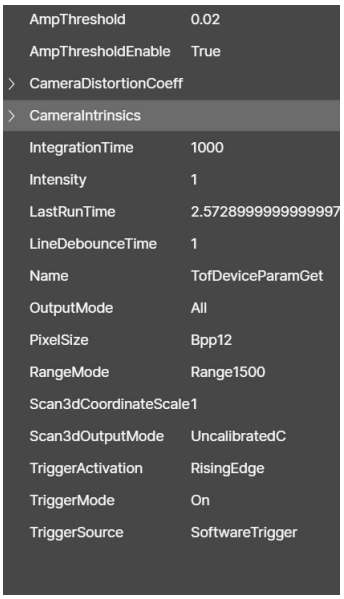


Figure 4.20: Camera parameters that can be adjusted using internal software. The integration time and intensity are defined by default as 1000 and 1.0 . Both of these parameters have a significant impact on the quality of the depth image

4.4.3 Robot Control

As part of Delta Electronics' requirements, CODESYS was chosen as the main control software for the robot arm. CODESYS is an integrated development environment designed to develop controller applications in accordance with the international industrial standard IEC 61131-3. Delta provides software called Designer-AX that integrates the CODESYS-based controller with additional customized features. CODESYS SoftMotion CNC+Robotics library provides function blocks for motion planning, inverse kinematics, trajectory planning and supports OPC UA communication with Python. Here are some explanations of the function blocks we use in this thesis.

- SMC_GroupPower: perform a power on command for each axis of the robot arm
- MC_GroupEnable: The axis group has several statuses, as shown in Figure 4.21. With this command, we are able to move the axis group from GroupDisabled to GroupStandby
- MC_MoveDirectAbsolute: execute a point to point movement with a non-linear trajectory
- MC_GroupReadActualPosition: povides information regarding the position and orientation of the end-effector
- SMC_GroupSetTool: in robotics, a tool is a configurable offset between the Tool Coordinate System and the Flange Coordinate System. We need to set the Z axis offset for the gripper installed on the end-effector in our case

Target position of movement function blocks are just variables that need to be filled in with the coordinate values in the following format:

$$pos1 : SMC\_POS\_REF := (c := (X := 100, Y := 100, Z := 100; A := 30, B := 40, C := 50))$$

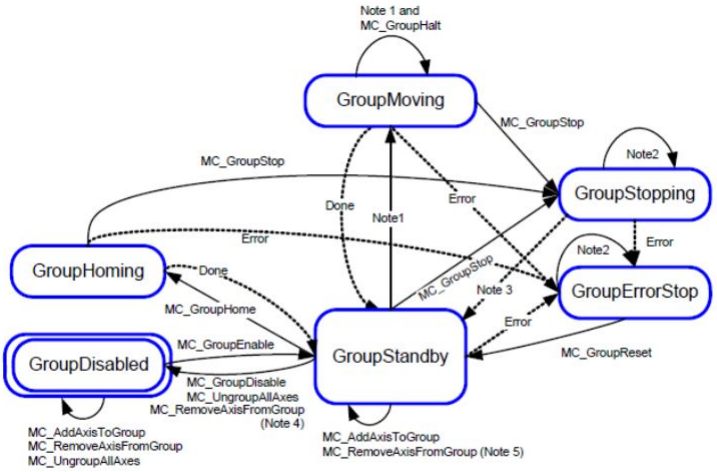


Figure 4.21: Different statuses of the axis group

where X,Y,Z are the Cartesian coordinates, and A,B,C are Euler angles in the order ZY'Z".

Our proposed system consists of several static and dynamic manipulator poses given by the grasping prediction module during the RBP task. A pipeline of the different robot poses during a picking operation can be seen in Figure 4.22, where the poses are as follows:

- Default Start Position
  This is the robot's starting position. Upon initialization, the robot will always start at this position.
- Grasp Standby
  As soon as CODESYS receives the predicted grasp from the master computer through OPC UA, the robot moves above to the target position and is ready to perform the grasping operation.
- Grasp Location
  Through MODBUS, the gripper received the done signal from the previous step and executed the grasp command.
- Move to New Object
  The robot arm moves the picked object to a new location. This series of operations may be repeated for a new object, or the system may terminate the process.

4.4.4 Grasping Objects

For simulation, we use object models from ShapeNetSem. ShapeNetSem contains a series of 3D object databases that have been extensively annotated. Consequently, the scaling of the objects is not homogeneous and in some cases is unknown. A set of 40 objects was randomly subsampled and manually rescaled using Blender [68] to fit the gripper and robot arm's dimensions in order to ensure that all the objects could be grasped by our simulated gripper. Figure 4.23 illustrates the objects that we wanted to grasp and pick during the simulation. It was chosen to represent the wide range of product attributes that may exist in an actual warehouse by selecting objects with a wide variety in terms of form, material, physical properties, and mass. For the real application,

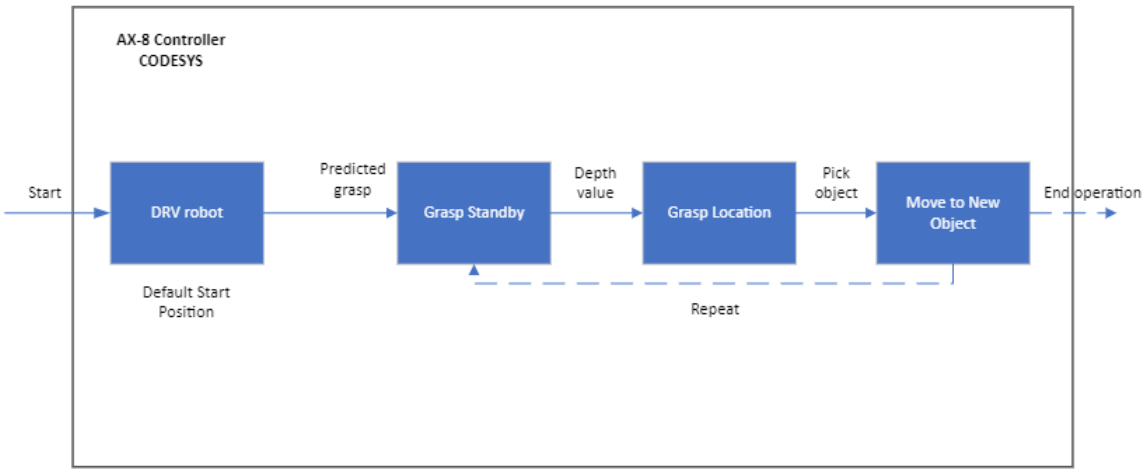


Figure 4.22: An illustration of the general movement performed by the DRV robot axis groups

we only utilized a simple cube since it is common in industry to use repeated shapes and due to time constraints, we only demonstrated the grasping capability. Figure 4.23 illustrates the cube.

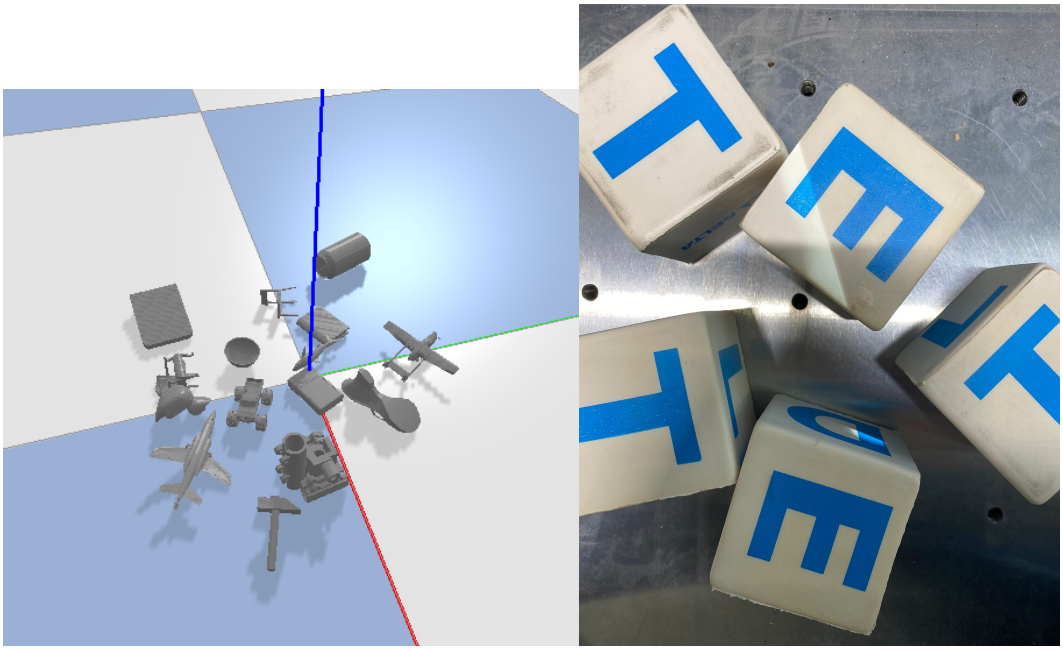
4.4.5 Code structure

The entire code consists of two main parts that are responsible for achieving both simulation and real-world application for planar grasps, as well as simulation for grasps with six degrees of freedom.

For the simulation of planar grasps, neural network models were implemented. A Pybullet simulation environment was set up in accordance with the URDF models of the robot arm and the objects. The neural networks were trained and evaluated using the original GG-CNN as well as the improved version GRConvNet. The real application includes the communication set up with different hardware and the PLC program for controlling the robots throughout the whole RBP process. There is also a Jupyter notebook file included for demonstration purposes.

As with 6DoF grasp, neural network models and visualization tool output (i.e. Tensorboard) were incorporated along with a simulation environment for the Isaac Gym. We provide a demonstration of 6DoF grasp on a real point cloud as well as a simulation of the grasping scene.

```
Graduation_project_Jie/
    Planar grasp/
        Application/
            hardware/
                camera.py
                eye_to_hand_cali.py
                Apriltag.m
            PLC_project/
                robotGraspWithVision.project
            demo.ipynb
            opcua.py
        Simulation/
            delta_drv90l/
                meshes/
                    collision
```



(a) (b)

Figure 4.23: Examples of grasping objects. (left) simulated objects from ShapeNetsem database in Pybullet. (right) real application grasping cubes provided by Delta

```
                visual
            urdf/
                drv90l.urdf
                gripper.urdf
        dataset/
            cornell
            jacquard
        ggcnn/
            models/
            output/
            utils/
                data
                dataset_processing
                visualisation
            eval_ggcnn.py
            train_ggcnn.py
        grConvnet/
            models/
            output/
            utils/
                evaluate.py
                train_network.py
        objects_model/
        graspLebel/
            generate_target_ggcnn.py
            showlabel.py
        sim_grasp.py
        simEnv.py
6DoF grasp/
    ycb_dataset/
    output/
    models/
    isaac_sim/
        assets/
```

```
        results/
        sim_one_object_all_grasps.py
    demo.py
    train.py
```

Listing 4.1: code structure

Chapter 5

Results

In this following chapter, we will present the results of our proposed system. Due to time constraints, we were able to achieve planar grasp with both simulations and real applications, but only with simulation for 6DoF grasps.

5.1 Planar Grasp

5.1.1 Physical setup

The physical setup consists of a number of components that communicate via input and output ports. Figure 5.1 illustrates a simple schematic representation of the system, in which the master PLC controls the 3D camera and robot.

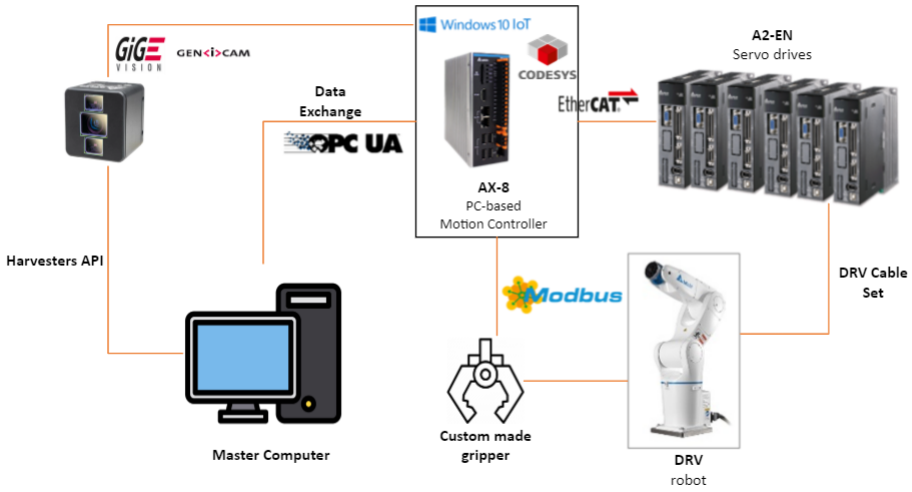


Figure 5.1: Schematic of the system. A motion controller and six servo drives are connected to the robot. A custom-made gripper connects the robot and controller via RS-485 communication. AX-8 and the master computer exchange data through OPC UA communication. Harvesters API is used on the master computer to control the 3D ToF camera using the GigE standard

5.1.2 Eye-to-Hand Calibration

The eye-to-hand calibration was conducted in accordance with Section 3.6.1. A total of 20 images were taken along with the recorded robot position at each iteration. As a result of the MATLAB calibration toolbox, intrinsic and extrinsic parameters of the camera were calculated, as well as distortion coefficients.

Intrinsic Parameters

In the camera matrix K, the intrinsic parameters were contained:

$$\mathbf{K} = \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}$$

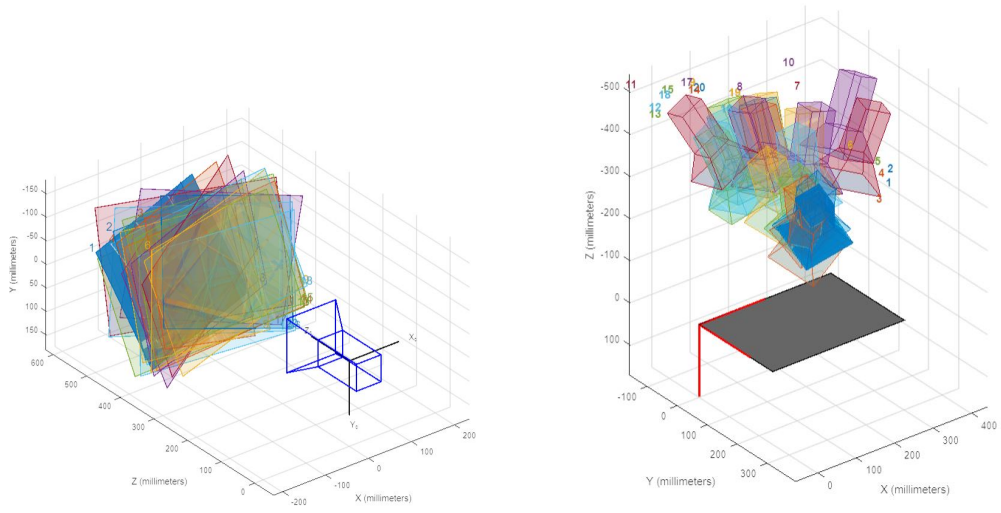
A camera's focal length is given by f, its axis skew by s, and its optical center by (x_0, y_0). It is important to note that these values are only a function of the camera, which means they are independent of the positioning and orientation of the camera in space. However, due to differences in mounting and production variation of individual components, they may vary among assumed identical camera models. According to the calibration, the resulting camera matrix is as follows:

$$\mathbf{K} = \begin{pmatrix} 527.627 & 0.000 & 339.664 \\ 0.000 & 527.527 & 238.394 \\ 0.000 & 0.000 & 1.000 \end{pmatrix}$$

In addition, camera distortion coefficients were estimated to be:

$$\mathbf{K}_d = \begin{pmatrix} -0.248 & 0.184 & -0.163 & 0 & 0 \end{pmatrix}.$$

Figure 5.2 shows the results of the MATLAB calibration toolbox analysis. This figure illustrates the relationship between camera and calibration pattern, with respect to camera or object.



(a) (b)

Figure 5.2: Intrinsic paramters result from MATLAB. (left) camera centric (right) object centric

Extrinsic Parameters

In the extrinsic camera parameters, the location and orientation of the camera are described. According to the calibration, the homogeneous transformation matrix between the robot base and the mounted camera T_C^B is as follows:

$$\mathbf{T}_C^B = \begin{pmatrix} 0.99842424 & 0.04931398 & -0.02678007 & 760.89255123 \\ 0.0493939 & -0.99877666 & 0.00233091 & 124.59082248 \\ -0.02663236 & -0.00365001 & -0.99963863 & 863.50226556 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

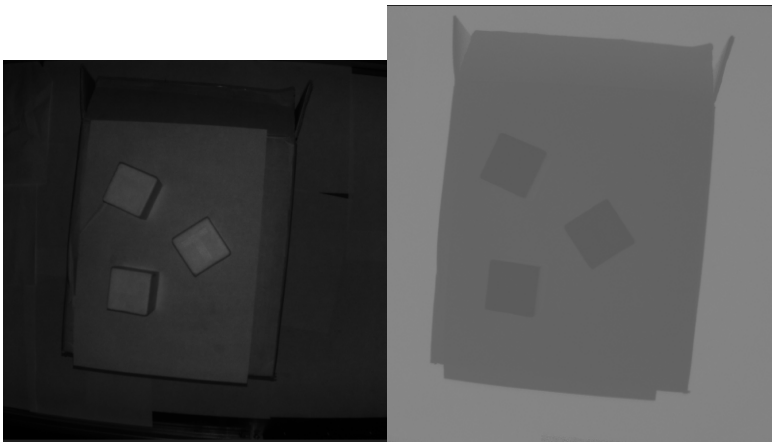
In order to estimate the rigid transformation T_G^W between the world frame W and the grasping location G, the transformation matrix T_C^B was used:

$$\mathbf{T}_G^W = \mathbf{T}_B^H \mathbf{T}_C^B \mathbf{T}_G^C$$

where T_B^H is the transformation from the the hand/end-effector to robot base. T_G^C is the transformation from the camera to the grasp position.

5.1.3 Image Acquisition

In accordance with the proposed image processing pipeline from Section 4.4.2, we were able to capture high quality depth images of the objects using the 3D camera. This was essential for the picking operation, as the grasping module determines the object location based on this depth image. Figure 5.3 illustrates an example captured depth image of grasping scene.



(a) (b)

Figure 5.3: Example of amplitude and aligned depth images acquired with grasping objects (cubes) in the scene.

5.1.4 Neural Networks

Planar grasp

In order to interpret the results of the GG-CNN network during training and evaluation, the visualization tool Tensorboard [69] was used. TensorBoard was accessed using the following command:

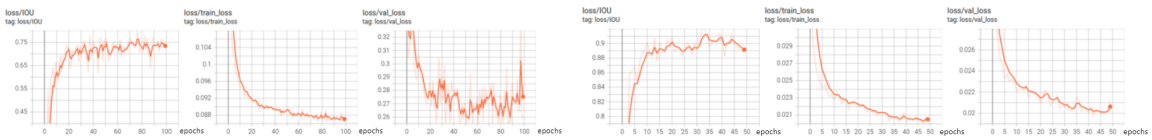
```
$ tensorboard --logdir='/your/path/here'
```

In this case, TensorFlow creates a local host server on which a visual representation is displayed. With TensorBoard, we were able to monitor the progress of the networks as they were trained using graphs, diagrams, and histogram plots. To detect overfitting during training, the accuracy of the evaluation was continuously monitored. In order to evaluate the performance and behavior of the network during training, different configurations with different hyperparameters were tested. Based on [19] [20] and results we observed, we found that the hyperparameters listed in Table 5.1 produced the best results. Figure 5.4 illustrates the validation precision for both networks, as well as the corresponding total loss graph for the same training period.

Hyperparameter	Value
Training epochs	50
Batch size	8
Batches per epoch	1000
Validation batches	250
Optmizer for the training	Adam

Table 5.1: Hyperparameters values

Validation precision is calculated in accordance with Section 4.3.2. Based on the validation accuracy curve (IOU), GG-CNN and GRConvNet exhibit rapid increases in accuracy before stabilizing at 75% and 90%, respectively. At a given point, we cannot exclude the possibility of overfitting in the networks, as we do not observe a sudden decrease in the score. Due to the similarity between the training and validation sets, this can be attributed to the limited variation between them. Based on the results of the total loss graphs, we are able to conclude that our trained networks yielded sufficient classification for the training set since the squared error loss (L2 loss function) are relatively low. Upon viewing the validation graph, the networks converged towards their highest validation accuracy, so we stopped training and saved the frozen graphs at the highest peak. More detailed graphs of our trained models can be found in Appendix B.



(a) (b)

Figure 5.4: Accuracy curves and loss curves of planar grasp networks (a) GG-CNN(b) GR-ConvNet. The x-axis represents the epochs of the training process. The y-axis plots (from left to right) the IoU matrices (i.e. the grasp accuracy), the L2 loss function values of the training dataset and the L2 loss function values of the validation dataset

6DoF grasp

As described in Section 4.3.3, the 6DoF grasp is comprised of two neural networks, a grasp generator and a grasp evaluator. Both of them are based on the PointNet++ architecture. Figure 5.5 and Figure 5.6 illustrate the training curves. As can be seen, the accuracy of the evaluator model remains stable around 80%. Because of time constraints and large computation times, we did not perform hyperparameter tuning and used the default settings provided in the papers. We will discuss the grasping results in the following section, and more training curves can be found in Appendix B.

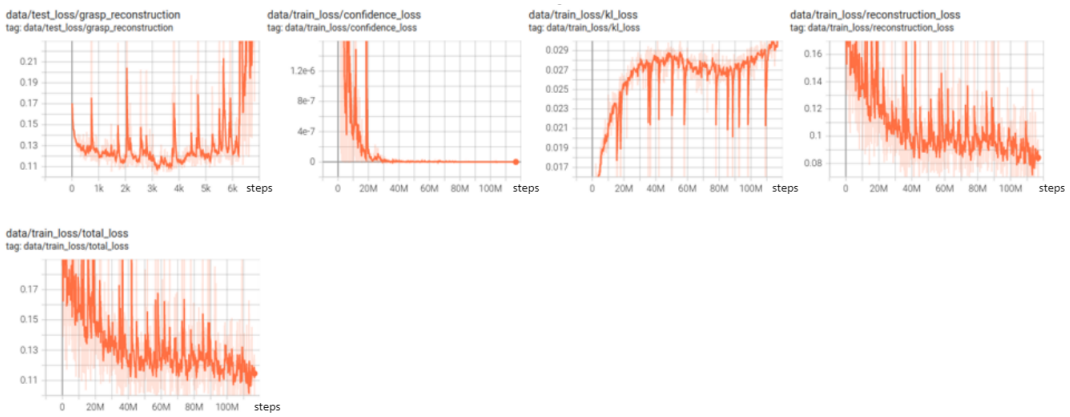


Figure 5.5: Training curves of VAE. The x-axis shows the steps taken during the training process, while the y-axis (from left to right, top to bottom) shows the reconstruction loss of the testing dataset (orientation and translation loss of the generated grasps with the ground truth grasps), confidence loss (confidence term that penalizes outputting zero confidence), KL-divergence loss, reconstruction loss, and total loss of the training dataset

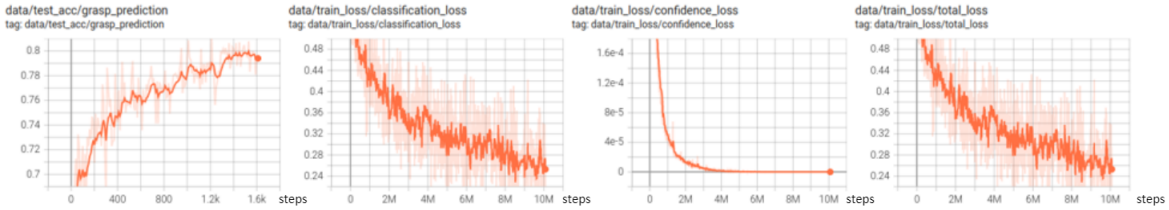


Figure 5.6: Training curves of grasp evaluator. The x-axis shows the steps taken during the training process, while the y-axis (from left to right) indicates the accuracy of grasping the testing dataset, classification loss (cross-entropy loss), confidence loss (confidence term that penalizes outputting zero confidence) and total loss of the training dataset

5.1.5 Simulation

Based on our previous discussion in Section 4.4, we developed a simulation environment in Pybullet to evaluate planar robotic grasping. The simulation environment consists of the DRV robot arm equipped with a gripper and a simulated depth camera that observes the robot's workspace from above. A robot is set to a predefined homing pose and randomly selected objects are placed in arbitrary poses within the robot's workspace. In all experiments, the robot knows in advance how to place the object, while the neural network model must predict the best grasping pose for the given scene and send it to the robot to grasp, pick up, and place the object. Whenever an object is higher than a threshold at the end of a pick and place mission, it is recorded as a success, and the object will be automatically removed so that the process can be repeated.

Our neural networks were evaluated under two different scenarios: isolated and cluttered. In the isolated object scenario, only one randomly selected object is placed. For the cluttered scenarios, 5 objects are randomly placed on the floor in order to simulate a cluttered pile of objects.

To assess the performance of the models, we measured the pick success rate, which is the ratio of successful grasps to attempts. For each experiment, we conducted 50 grasp attempts and reported the pick success rate. In Table 5.2, the results for different models have been summarized for objects that were tested in isolation and in clutter. As shown in the graphs, the GRConvNet and improved GGCNN models perform significantly better than the original GGCNN model in both

isolated and clutter scenarios.

Approach	Training Dataset	Isolated	Cluttered
GGCNN	Cornell	79.0	74.5
GGCNN improved	Cornell	83.5	80.0
GR-ConvNet	Cornell	93.0	90.0
GR-ConvNet	Jacquard	92.5	95

Table 5.2: Pick success rate (%) in simulation

5.1.6 Real Application

Real-world applications were conducted on the Delta Electronics DRV robot. For grasping the test objects, a parallel gripper was custom-made. Due to the time constraints, this application was only used to demonstrate Delta's product's potential on RBP. The success rate of grasping with the real application could not be tested out using different shapes of objects. Demonstrations were conducted using only simple cubes. Based on its outstanding performance proven previously in Section 5.1.5, we chose GRConvNet as the neural network. A visualization demo is shown in Figure 5.7. Also the predicted grasping center for the testing cube and the output of the grasp mapping are shown.

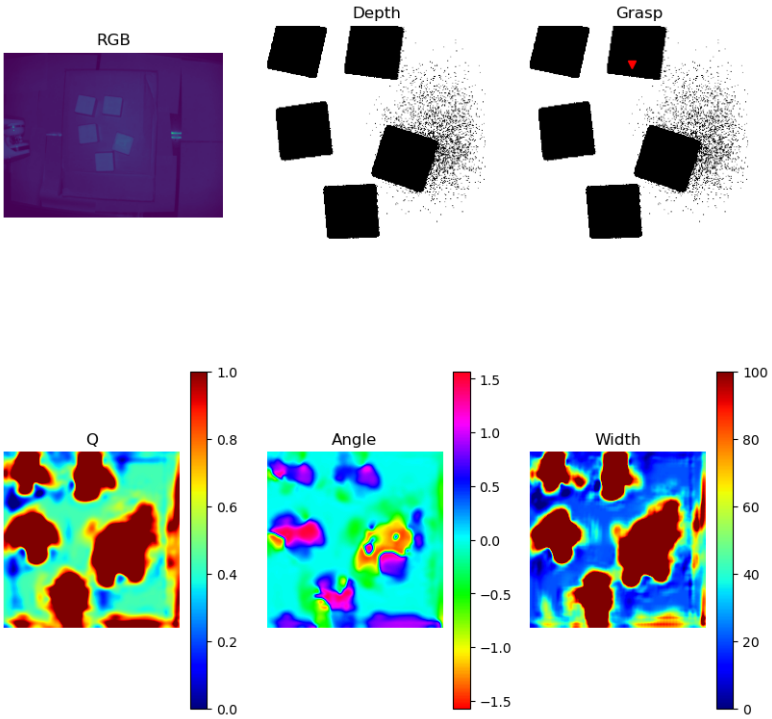


Figure 5.7: Visualization of the grasping prediction module for real-world application. As shown in the image above, the depth image was captured and cropped, and a red dot indicates the predicted grasping center. Listed below are the grasp quality (successful grasp probability from 0 to 1), grasp angle in radian, and grasp width in pixels.

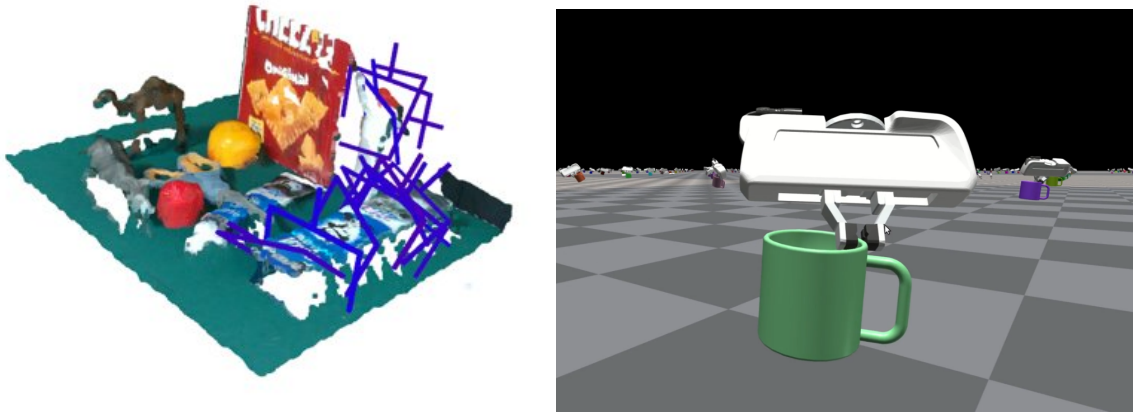
5.2 6DoF grasp

The simulation for 6DoF grasp was conducted in IssacGym due to its outstanding performance on GPUs. A simulation environment has been developed for the grasping scene, similar to the planar grasping scene. This simulation environment only includes the gripper model and several category object models, as listed below in Table 5.3, due to the complexity and time commitment of the 6DoF grasp model. In the simulation environment, the mentioned combination network was tested with three categories: mug, bottle, and bowl. According to the grasp scores calculated by the evaluation networks, the final grasp poses are selected. The new combination network achieves average 77% success rate and requires less computing time.

Approach	Mug	Bottle	Bowl	Average
6DoF grasp	76.3	78.0	76.7	77.0

Table 5.3: results of grasp success rates (%) among different categories of objects

Figure 5.8 shows a demonstration of predicted grasps using a real-world point cloud and a simulation testing scene in Isaac Gym.



(a) (b)

Figure 5.8: (a) Demonstration of generating predicted graphics from RealSense point clouds. (b) Simulation of grasping a mug with a gripper model in the Isaac Gym

5.3 Demonstration Video

A demonstration video was produced to illustrate the potential of planar grasp on Delta's products. Additionally, simulations of planar grasp and 6DOF grasp are demonstrated. You can find the video in the digital appendix.

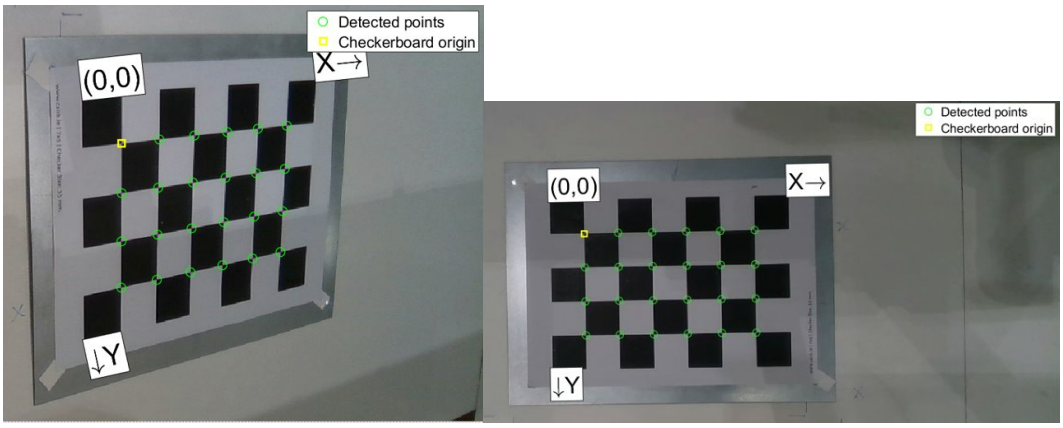
Chapter 6

Discussions and recommendations

A discussion of the results obtained from our practical work will be presented in the following chapter. A review of the system's strengths and weaknesses will be presented, as well as suggestions for improvements.

6.1 AprilTag v.s. Chessboard

Most camera hand-to-eye calibrations use a chessboard pattern as the calibration pattern. Initially, we chose the chessboard pattern as well, but the calibration results were always far from the real value. As we discovered, the MATLAB calibration toolbox does not recognize pattern changes when they are rotated 180 degrees. This issue is explained in Figure 6.1. In order to avoid this issue, we decided to use AprilTag as the calibration pattern since it contains unique patterns.



(a) (b)

Figure 6.1: Incorrect corner points detection on MATLAB. (a) Pose A (b) Pose B (approx. 180° from Pose A)

6.2 Depth Image Quality

In order to locate the assumed best grasp location, the neural network requires a high quality depth image. Feature enhancement and noise removal operations are performed on the initial depth image before it is fed into the neural network. Despite the improvement in the processed image, it proved difficult to obtain a high quality depth image for all surfaces, textures, and angles of the objects. Using more smoothing filters can partially resolve this problem, but at the expense

of reduced texture and edge details in the depth image. As shown in Figure 6.2, camera settings affect depth image noise and quality.

(a) (b)

Figure 6.2: A comparison of depth images taken under different settings. (a) A smoother and less noisy depth image with higher integration time and intensity settings. (b) A depth image with more noise when the integration time and intensity are reduced

6.3 Object segmentation

In general, the neural network performed well on semi-structured bin picking operations in which the objects are clearly separated. Stacking two or more objects side by side resulted in a higher failure rate. As a result, the neural network often evaluated the side-by-side objects as one solid object. Thus, the final grasp location was located near the center of the combined objects, which in reality might have been near the edge of the combined objects. Due to the fact that the network is only trained on single object depth images, this type of failure may occur.

As a solution to this issue, we propose a separate object segmentation module in the bin picking pipeline that may be able to resolve the issue where several objects are evaluated as one. The Figure 6.3 illustrates the results of a proposed object segmentation module that could be implemented in our system in order to increase success rates. Based on the work presented in [22], the module applies a Feature Pyramid Network (FPN) to achieve segmentation. Figure 6.4 illustrates the results of testing the grasping module using this segmentation method.

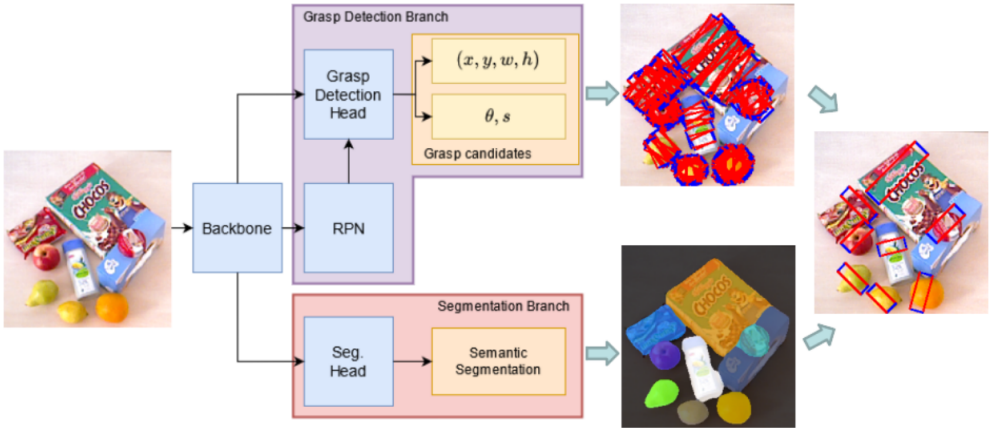


Figure 6.3: Architecture of the proposed model includes a segmentation module [22]. The backbone network is shared by both branches for grasp detection and segmentation. A grasp refinement head uses both outputs (grasp candidates and semantic segmentation) to predict refined grasp candidates with increased accuracy

Figure 6.4: Results of the testing of the grasping module using the segmentation method [22]. The following images are explained from left to right: 1) the raw input image; 2) predicted semantic segmentation, where each color represents a specific class; and 3) the best possible grasp for each class in the scene (blue lines indicate parallel plates of the gripper, red lines indicate opening width). Each row represents a different example of input

6.4 Bin picking

Due to limited time and the complexity of 6DoF grasp, we were not able to integrate the algorithm with a real-world application for this graduation project. Planar grasp should also be tested with more experiments, for example, using different categories of objects. In addition, internal software that is being developed could accelerate the process, since it will support automatic calibration and direct extraction of point clouds. Consequently, it would be possible to implement the 6DoF grasp on real applications and to integrate the planar grasp system into an automated system.

Chapter 7

Conclusion

The purpose of this thesis was to develop a functional solution using a 3D ToF camera for RBP tasks. This thesis aimed to answer the following research question:

How to develop a CODESYS-based robotics solution for RBP tasks using a 3D ToF camera and a Delta DRV robot arm?

To answer this main question, we explored deep learning based RBP algorithms and integrated them with existing Delta products. By conducting extensive literature research, we were able to gain a comprehensive understanding of the different technological approaches that might be used to accomplish the task at hand. Through our practical work, we have successfully met the objectives stated in Section 1.3 of the problem formulation. The kinematics of the robot were described, followed by an estimation of the eye-to-hand coordination between the camera and the robots. To ensure the success of grasping, a gripper was designed and manufactured, planar and 6DoF grasping neural networks were trained, and simulations and real-world experiments were conducted. Due to time constraints, real-world applications are only possible with planar grasps.

We selected deep neural networks for the purpose of detecting object grasping, as these methods have shown significant improvements in performance since their introduction. It became evident from the results that our grasp prediction module was capable of picking a wide variety of objects in a simulation environment. Additionally, the results demonstrate the robustness of deep learning approaches in general.

Individual modules were combined with Python scripts and CODESYS PLC programming to create one autonomous bin picking system, which was evaluated in terms of robustness and feasibility for implementation in a realistic setting. The camera and master computers were connected using Harversters API, and the master computer is connected to the PLC that controls the servo drives on the robot arm using OPC UA. Simulation results indicate that the proposed bin picking system achieved a success rate of 95% when picking unspecified objects using a planar grasp algorithm and 77% when using a 6DoF grasp algorithm. There was a problem distinguishing and separating objects that were stacked side by side, as the network often evaluated these objects as a single object. In order to resolve this issue, an object segmentation module could be introduced into the pipeline.

In general, this thesis explores the possibility of integrating Delta's products with intelligent vision-guided RBP. As a result of the practical work and experiments conducted in this master's thesis, a bin picking system has been developed by combining deep learning, computer vision, and robotics. Each module of the system has been thoroughly explained, and weaknesses and strengths have been identified.

Bibliography

[1] W. C. Chang and C. H. Wu, "Eye-in-hand vision-based robotic bin-picking with active laser projection," *International Journal of Advanced Manufacturing Technology*, vol. 85, pp. 2873–2885, 8 2016. 1

[2] T. T. Le and C. Y. Lin, "Bin-picking for planar objects based on a deep learning network: A case study of usb packs," *Sensors (Switzerland)*, vol. 19, no. 16, 2019. 1

[3] C. Martinez, R. Boca, B. Zhang, H. Chen, and S. Nidamarthi, "Automated bin picking system for randomly located industrial parts," in *IEEE Conference on Technologies for Practical Robot Applications, TePRA*, vol. 2015-August, IEEE Computer Society, 8 2015. 1

[4] R. Bogue Consultant, "Random Bin Picking: Has Its Time Finally Come?," *Assembl. Autom.*, vol. 34, pp. 217–221, 7 2014. 1

[5] G. Sansoni, M. Trebeschi, and F. Docchio, "State-of-the-art and applications of 3D imaging sensors in industry, cultural heritage, medicine, and criminal investigation," 1 2009. 1

[6] "About Delta - Delta Profile - Delta." 2

[7] K. Kleeberger, R. Bormann, W. Kraus, and M. F. Huber, "A Survey on Learning-Based Robotic Grasping," *Current Robotics Reports*, vol. 1, pp. 239–249, 12 2020. 5, 9

[8] M. Grard, *Generic instance segmentation for object-oriented bin-picking.* PhD thesis, Lyon, 2019. 5, 8, 9

[9] A. Cordeiro, L. F. Rocha, C. Costa, P. Costa, and M. F. Silva, "Bin Picking Approaches Based on Deep Learning Techniques: A State-of-the-Art Survey," in *2022 IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2022*, pp. 110–117, Institute of Electrical and Electronics Engineers Inc., 2022. 5, 9

[10] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-driven grasp synthesis-A survey," *IEEE Transactions on Robotics*, vol. 30, no. 2, pp. 289–309, 2014. 5

[11] A. T. Miller, S. Knoop, H. I. Christensen, and P. K. Allen, "Automatic grasp planning using shape primitives," in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1824–1829, 2003. 5

[12] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. Aparicio Ojea, and K. Goldberg, "Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics," tech. rep. 6, 28, 30

[13] A. Saxena, J. Driemeyer, and A. Y. Ng, "Robotic grasping of novel objects using vision," in *International Journal of Robotics Research*, vol. 27, pp. 157–173, 2 2008. 7

[14] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *International Journal of Robotics Research*, vol. 34, pp. 705–724, 4 2015. 7, 25, 26

[15] J. Redmon and A. Angelova, "Real-time grasp detection using convolutional neural networks," in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2015-June, pp. 1316–1322, Institute of Electrical and Electronics Engineers Inc., 6 2015. 7, 25

[16] L. Wang, Z. Zhang, J. Su, and Q. Gu, "Robotic Autonomous Grasping Technique: A Survey," in *Proceedings of 2021 5th Asian Conference on Artificial Intelligence Technology, ACAIT 2021*, pp. 287–295, Institute of Electrical and Electronics Engineers Inc., 2021. 7

[17] D. Morrison, P. Corke, and J. Leitner, "Closing the Loop for Robotic Grasping: A Real-time, Generative Grasp Synthesis Approach," tech. rep. 7, 27, 28, 42

[18] A. Zeng, S. Song, K. T. Yu, E. Donlon, F. R. Hogan, M. Bauza, D. Ma, O. Taylor, M. Liu, E. Romo, N. Fazeli, F. Alet, N. C. Dafle, R. Holladay, I. Morena, P. Qu Nair, D. Green, I. Taylor, W. Liu, T. Funkhouser, and A. Rodriguez, "Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching," in *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3750–3757, Institute of Electrical and Electronics Engineers Inc., 9 2018. 7

[19] G. Du, K. Wang, S. Lian, and K. Zhao, "Vision-based Robotic Grasping From Object Localization, Object Pose Estimation to Grasp Estimation for Parallel Grippers: A Review," 5 2019. 8

[20] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab, "SSD-6D: Making RGB-Based 3D Detection and 6D Pose Estimation Great Again," in *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-October, pp. 1530–1538, Institute of Electrical and Electronics Engineers Inc., 12 2017. 8

[21] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes," *arXiv preprint arXiv:1711.00199*, 2017. 8

[22] X. Deng, Y. Xiang, A. Mousavian, C. Eppner, T. Bretl, and D. Fox, *Self-supervised 6D Object Pose Estimation for Robot Manipulation; Self-supervised 6D Object Pose Estimation for Robot Manipulation*. 2020. 8

[23] A. Mousavian, C. Eppner, and D. Fox, "6-DOF GraspNet: Variational Grasp Generation for Object Manipulation," 5 2019. 8, 31, 32, 33

[24] A. Ten Pas, M. Gualtieri, K. Saenko, and R. Platt, "Grasp pose detection in point clouds," *The International Journal of Robotics Research*, vol. 36, no. 13-14, pp. 1455–1473, 2017. 8

[25] Institute of Electrical and Electronics Engineers, *2019 International Conference on Robotics and Automation (ICRA)*. 8

[26] Y. Li, Q. Lei, C. Cheng, G. Zhang, W. Wang, and Z. Xu, "A review: machine learning on robotic grasping," in *Eleventh International Conference on Machine Vision (ICMV 2018)* (A. Verikas, D. P. Nikolaev, P. Radeva, and J. Zhou, eds.), vol. 11041, p. 110412U, SPIE, 2019. 9

[27] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3642–3649, 2012. 10

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," tech. rep. 10, 12, 14

[29] L. Deng and D. Yu, "Deep learning: Methods and applications," 2013. 10, 11

[30] C. M. BERNERS-LEE, "Cybernetics and Forecasting," *Nature*, vol. 219, no. 5150, pp. 202–203, 1968. 10

[31] J. Schmidhuber, "Deep Learning in neural networks: An overview," 1 2015. 10

[32] J. D. Kelleher, *Deep Learning*. MIT Press Essential Knowledge Series, Cambridge, Massachusetts: The MIT Press, 2019. 11

[33] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision*, vol. 115, pp. 211–252, 12 2015. 12

[34] S. Mostafa and F. X. Wu, "Diagnosis of autism spectrum disorder with convolutional autoencoder and structural MRI images," in *Neural Engineering Techniques for Autism Spectrum Disorder: Volume 1: Imaging and Signal Analysis*, pp. 23–38, Elsevier, 1 2021. 12, 13

[35] D. Scherer, A. Müller, and S. Behnke, "Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition," tech. rep. 12

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," 12 2015. 13, 28

[37] R. K. Srivastava, K. Greff, and J. Urgen Schmidhuber, "Training Very Deep Networks," tech. rep. 13

[38] P. Oommen, "ResNets — Residual Blocks & Deep Residual Learning," 12 2020. 13

[39] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1717–1724, IEEE Computer Society, 9 2014. 14

[40] S. J. Pan and Q. Yang, "A survey on transfer learning," 2010. 14

[41] J. J. Craig, P. Prentice, and P. P. Hall, "Introduction to Robotics Mechanics and Control Third Edition," tech. rep., 2005. 15

[42] L. Fernandez, V. Avila, and L. Gonçalves, "A Generic Approach for Error Estimation of Depth Data from (Stereo and RGB-D) 3D Sensors," 2017. 16

[43] "Hand-Eye Calibration Problem — ZIVID KNOWLEDGE BASE documentation." 17

[44] F. Dornaika and R. Horaud, "Simultaneous Robot-World and Hand-Eye Calibration," *IEEE Transac-tions on Robotics and Automation*, vol. 14, no. 4, pp. 617–622, 1998. 18

[45] "Konstantinos Daniilidis Hand-Eye Calibration Using Dual Quaternions," tech. rep. 18

[46] "Delta Articulated Robot System User Manual," tech. rep. 21

[47] "Products - CODESYS Motion Solution - Delta." 21

[48] "Products - Servo Systems - AC Servo Motors and Drives - Delta." 22

[49] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3400–3407, 2011. 23

[50] J. Watson, J. Hughes, and F. Iida, "Real-world, real-time robotic grasping with convolutional neural networks," in *Towards Autonomous Robotic Systems: 18th Annual Conference, TAROS 2017, Guildford, UK, July 19–21, 2017, Proceedings 18*, pp. 617–626, Springer, 2017. 25

[51] Z. Ju, C. Yang, and H. Ma, "Kinematics modeling and experimental verification of baxter robot," in *Proceedings of the 33rd Chinese control conference*, pp. 8518–8523, IEEE, 2014. 25

[52] A. Depierre, E. Dellandréa, and L. Chen, "Jacquard: A Large Scale Dataset for Robotic Grasp Detection," 3 2018. 25, 26

[53] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Benchmarking in Manipulation Research: Using the Yale-CMU-Berkeley Object and Model Set," *IEEE Robotics and Automation Magazine*, vol. 22, pp. 36–52, 9 2015. 26, 27

[54] S. Kalkan, U. Saranlı, T. Orta Dogu Teknik Universitesi (Ankara, Institute of Electrical and Electronics Engineers, and IEEE Robotics and Automation Society, *Proceedings of the 17th International Conference on Advanced Robotics (ICAR) : 27-31 July, 2015, Istanbul, Turkey.* 26

[55] B. Siciliano and O. Khatib, "Springer Proceedings in Advanced Robotics 1," tech. rep. 28

[56] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50K tries and 700 robot hours," in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2016-June, pp. 3406–3413, Institute of Electrical and Electronics Engineers Inc., 6 2016. 28

[57] E. Johns, S. Leutenegger, and A. J. Davison, "Deep learning a grasp function for grasping under gripper pose uncertainty," in *IEEE International Conference on Intelligent Robots and Systems*, vol. 2016-November, pp. 4461–4468, Institute of Electrical and Electronics Engineers Inc., 11 2016. 28

[58] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000. 28

[59] Y. Jiang, S. Moseson, and A. Saxena, "Efficient grasping from rgbd images: Learning using a new rectangle representation," in *2011 IEEE International conference on robotics and automation*, pp. 3304–3311, IEEE, 2011. 29

[60] S. Kumra, S. Joshi, and F. Sahin, "Antipodal Robotic Grasping using Generative Residual Convolutional Neural Network," 9 2019. 29, 42

[61] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," 8 2017. 30

[62] H. Liang, X. Ma, S. Li, M. Görner, S. Tang, B. Fang, F. Sun, and J. Zhang, "PointNetGPD: Detecting Grasp Configurations from Point Sets," 9 2018. 30, 31

[63] V.-D. Nguyen, "Constructing force-closure grasps," in *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, vol. 3, pp. 1368–1373, Institute of Electrical and Electronics Engineers. 30

[64] S. Qiu and M. R. Kermani, "A new approach for grasp quality calculation using continuous boundary formulation of grasp wrench space," *Mechanism and Machine Theory*, vol. 168, p. 104524, 2022. 30

[65] "PyBullet Quickstart Guide." 33

[66] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning," 2021. 33

[67] B. Foundation, "blender.org - Home of the Blender project - Free and Open 3D Creation Software." 35

[68] Martín˜Abadi, Ashish˜Agarwal, Paul˜Barham, Eugene˜Brevdo, Zhifeng˜Chen, Craig˜Citro, Greg˜S.˜Corrado, Andy˜Davis, Jeffrey˜Dean, Matthieu˜Devin, Sanjay˜Ghemawat, Ian˜Goodfellow, Andrew˜Harp, Geoffrey˜Irving, Michael˜Isard, Y. Jia, Rafal˜Jozefowicz,

Lukasz˜Kaiser, Manjunath˜Kudlur, Josh˜Levenberg, Dandelion˜Mané, Rajat˜Monga, Sherry˜Moore, Derek˜Murray, Chris˜Olah, Mike˜Schuster, Jonathon˜Shlens, Benoit˜Steiner, Ilya˜Sutskever, Kunal˜Talwar, Paul˜Tucker, Vincent˜Vanhoucke, Vijay˜Vasudevan, Fernanda˜Viégas, Oriol˜Vinyals, Pete˜Warden, Martin˜Wattenberg, Martin˜Wicke, Yuan˜Yu, and Xiaoqiang˜Zheng, " TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015. 41

[69] S. Ainetter and F. Fraundorfer, "End-to-end Trainable Deep Neural Network for Robotic Grasp Detection and Semantic Segmentation from RGB," 7 2021. 47, 48

Appendix A

Gripper Manual

A.1 Wiring

The gripper has the for pin cable:

- Red: Power input
- Black: Ground
- Yellow: RS-485 A Signal
- Green: RS-485 B Signal

Note that some devices invert the signal lines or have different name convention, this was what matched with the adaptor we used. The gripper won't be responsive but nothing will be damaged, in that event swap the signal wire lines.

A.2 Control

The gripper exposes 4 modbus registers over the signal wires and has the device address of 5 The register map is as following:

- Register 0: Control Register (Input)
  - The register accepts values: 0, 1 and 2, any other value will we corrected to value 0
  - Value 0 sets the device to idle, the value is also set automatically by the system when it reaches the target pose
  - Value 1 sets the device to travel to position open
  - Value 2 sets the device to travel to position close
- Register 1: Status Register (Output)
  - Execute only read operations on it
  - The value returned is the status code and maps as follows: 1 - Starting, 2 - Idle, 3 - Moving to close position, 4 - Moving to open position
  - The device accepts the commands only when in Idle state, when entering this state it also sets the command register to 0
- Register 2: Target Open Pose (Input)
  - Holds the target pose for the open command, value is an unsigned integer representing degrees

  - On startup the value is setup to the default of 90
  - The value can be changed when the device is idle

- Register 3: Target CLose Pose (Input)
  - Holds the target pose for the open command, value is an unsigned integer representing degrees
  - On startup the value is setup to the default of 80
  - The Value can be changed when the device is idle

Appendix B

Tensorboard Training Curves

B.1 Planar Grasp

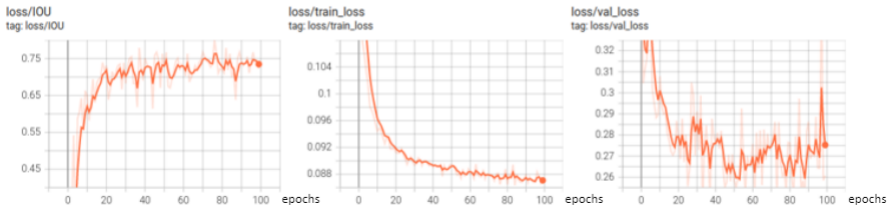


Figure B.1: GG-CNN training curve. The x-axis represents the epochs of the training process. The y-axis plots (from left to right) the IoU matrices (i.e. the grasp accuracy), the L2 loss function values of the training dataset and the L2 loss function values of the validation dataset

Figure B.2: GG-CNN training dataset loss curves. The x-axis represents the epochs of the training process. The y-axis plots (from left to right) the angle *cos* loss, the probability loss, the angle *sin* loss, the width loss of the training dataset

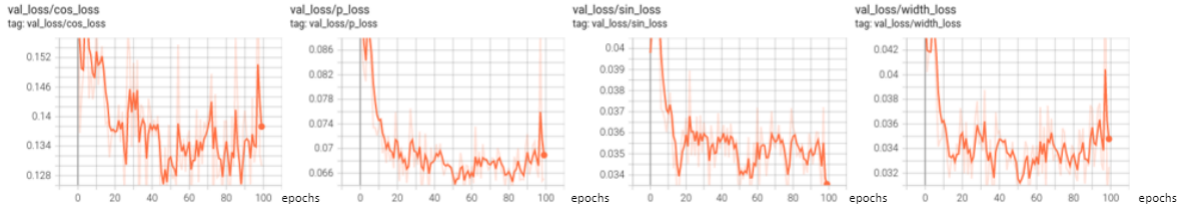


Figure B.3: GG-CNN validation dataset loss curves. The x-axis represents the epochs of the training process. The y-axis plots (from left to right) the angle *cos* loss, the probability loss, the angle *sin* loss, the width loss of the validation dataset

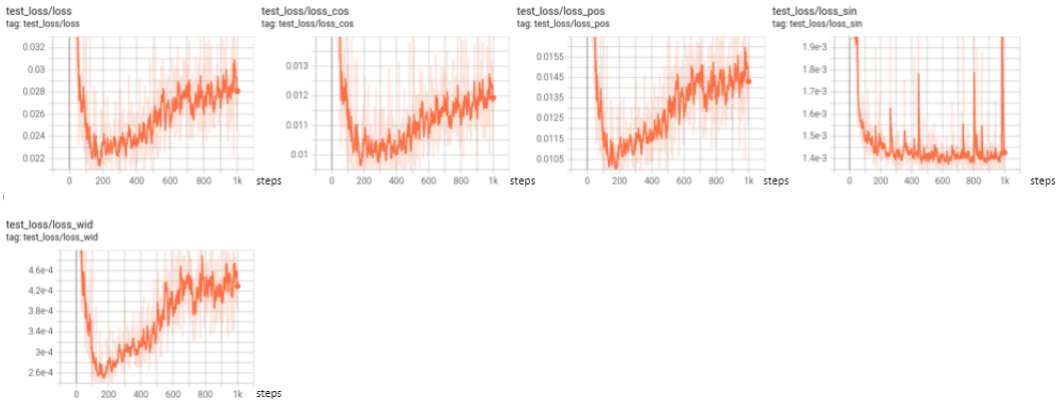


Figure B.4: GG-CNN modified version testing dataset loss curve. The x-axis represents the steps of the training process. The y-axis plots (from left to right) the total loss, the angle *cos* loss, the probability loss, the angle *sin* loss and the width loss of the testing dataset

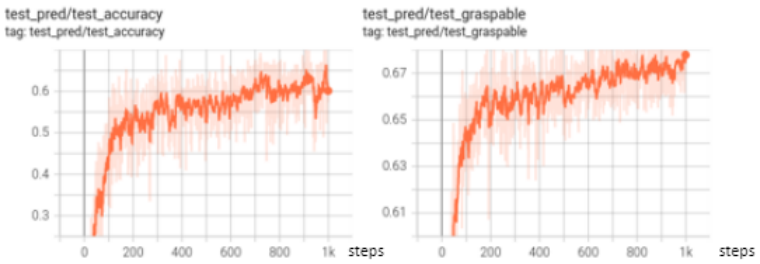


Figure B.5: GG-CNN modified version accuracy curve. The x-axis represents the steps of the training process. The y-axis plots (from left to right) the prediction accuracy (between the predicted outputs and ground truth) and the probability accuracy of the testing dataset

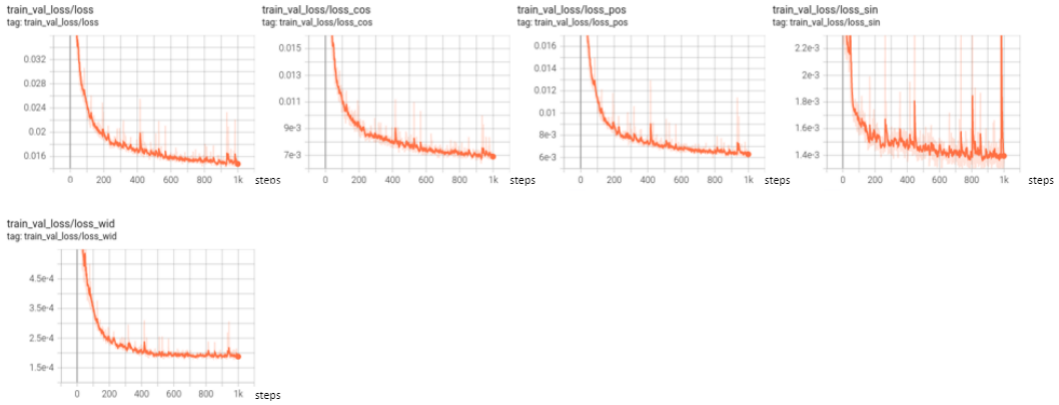


Figure B.6: GG-CNN modified version training dataset loss curve. The x-axis represents the steps of the training process. The y-axis plots (from left to right) the total loss, the angle *cos* loss, the probability loss, the angle *sin* loss and the width loss of the training dataset

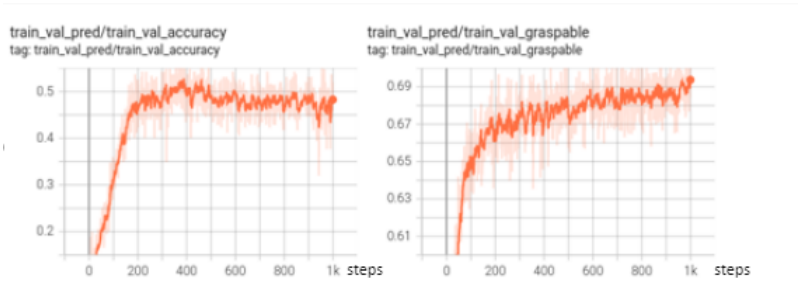


Figure B.7: GG-CNN modified version accuracy curve. The x-axis represents the steps of the training process. The y-axis plots (from left to right) the prediction accuracy (between the predicted outputs and ground truth) and the probability accuracy of the training dataset

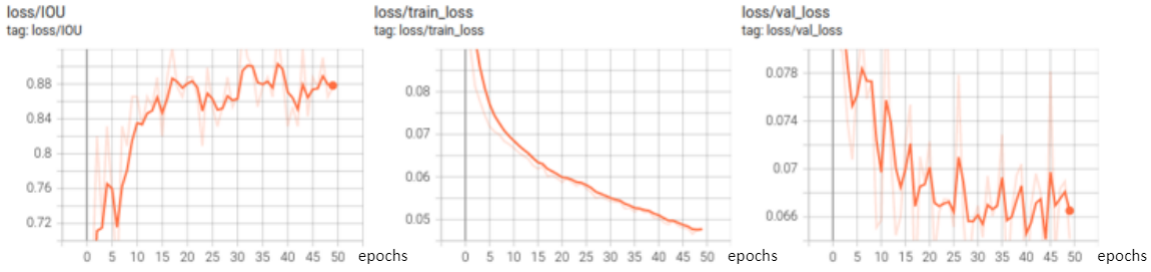


Figure B.8: GRConvNet training curve using Cornell dataset. The x-axis represents the epochs of the training process. The y-axis plots (from left to right) the IoU matrices (i.e. the grasp accuracy), the L2 loss function values of the training dataset and the L2 loss function values of the validation dataset

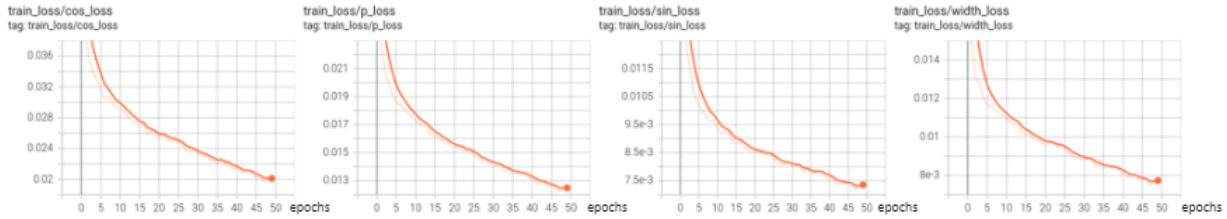


Figure B.9: GRConvNet training dataset loss curves. The x-axis represents the epochs of the training process. The y-axis plots (from left to right) the angle *cos* loss, the probability loss, the angle *sin* loss, the width loss of the training dataset

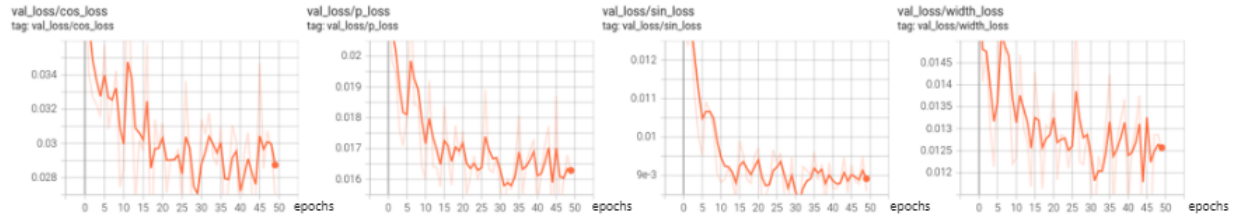


Figure B.10: GRConvNet validation dataset loss curves. The x-axis represents the epochs of the training process. The y-axis plots (from left to right) the angle *cos* loss, the probability loss, the angle *sin* loss, the width loss of the validation dataset

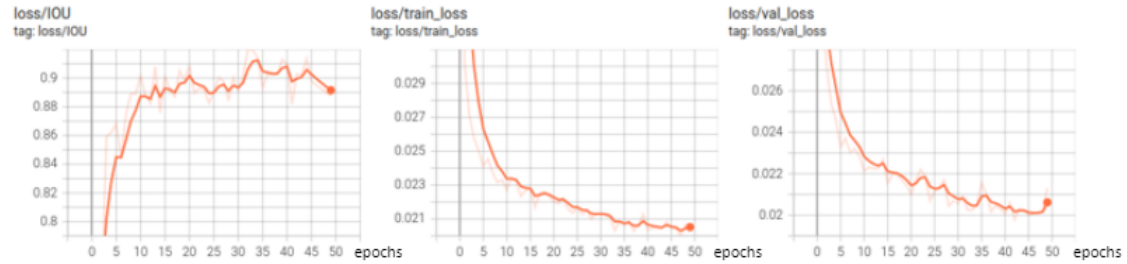


Figure B.11: GRConvNet training curve using Jacquard dataset. The x-axis represents the epochs of the training process. The y-axis plots (from left to right) the IoU matrices (i.e. the grasp accuracy), the L2 loss function values of the training dataset and the L2 loss function values of the validation dataset

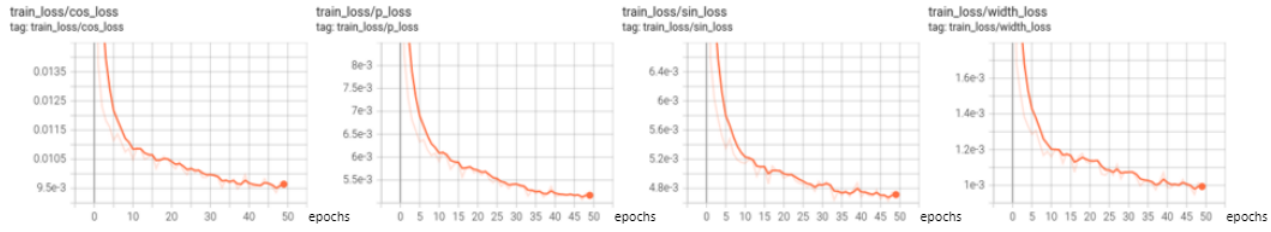


Figure B.12: GRConvNet training dataset loss curves. The x-axis represents the epochs of the training process. The y-axis plots (from left to right) the angle *cos* loss, the probability loss, the angle *sin* loss, the width loss of the training dataset

Figure B.13: GRConvNet validation dataset loss curves. The x-axis represents the epochs of the training process. The y-axis plots (from left to right) the angle *cos* loss, the probability loss, the angle *sin* loss, the width loss of the validation dataset

B.2 6DoF Grasp

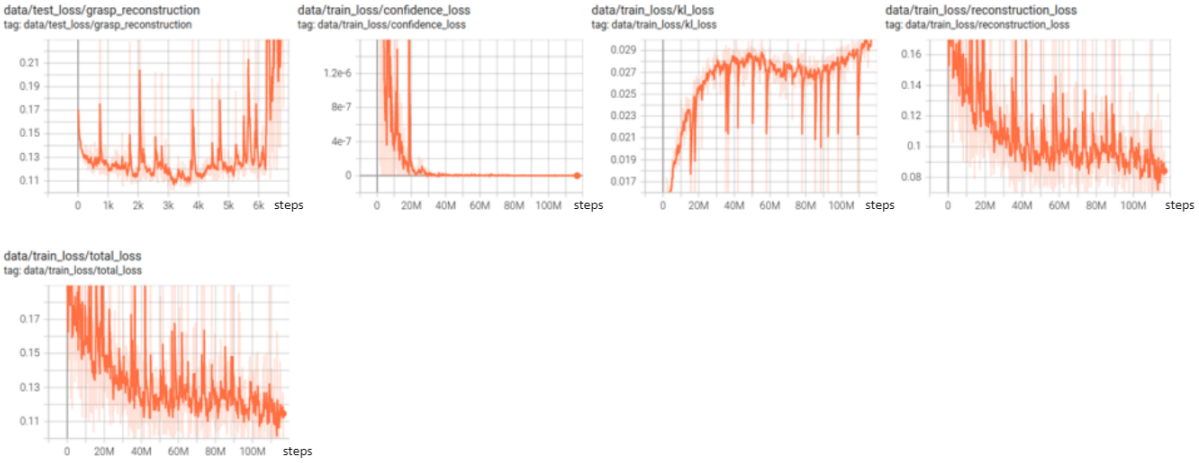


Figure B.14: Training curves of VAE. The x-axis shows the steps taken during the training process, while the y-axis (from left to right, top to bottom) shows the reconstruction loss of the testing dataset (orientation and translation loss of the generated grasps with the ground truth grasps), confidence loss (confidence term that penalizes outputting zero confidence), KL-divergence loss, reconstruction loss, and total loss of the training dataset

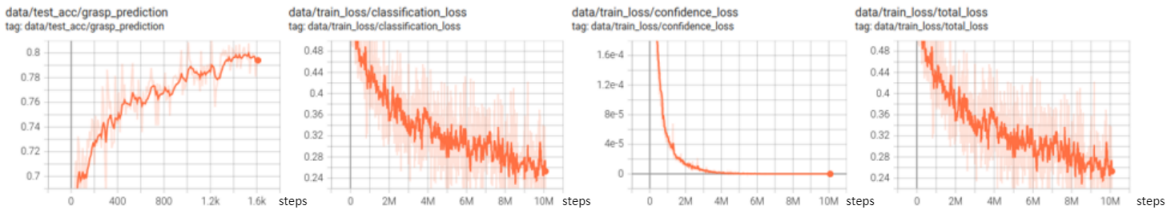


Figure B.15: Training curves of grasp evaluator. The x-axis shows the steps taken during the training process, while the y-axis (from left to right) indicates the accuracy of grasping the testing dataset, classification loss (cross-entropy loss), confidence loss (confidence term that penalizes outputting zero confidence) and total loss of the training dataset

Appendix C

Digital Appendix

There was a digital appendix attached to the thesis that contained the demonstration videos:

- DemoVideo.mp4: demonstration video with RBP tasks on real application
- Simulation.mp4: video on planar and 6DoF grasp simulations