NTNU
Norwegian University of
Science and Technology

Development of an Automated Bin Picking System for Cluttered Environments

Sondre Aleksander Vadheim
Hans Bratland Østerdal

Master of Science in Mechanical Engineering

Submission date:	June 2018
Supervisor:	Olav Egeland, MTP
Co-supervisor:	Tingelstad Lars, MTP

Norwegian University of Science and Technology
Department of Mechanical and Industrial Engineering

NTNU

Fakultet for ingeniørvitenskap
Institutt for maskinteknikk og produksjon

Dato	Referanse
08.06.18	OEG

MASTEROPPGAVE 2018
Sondre A. Vadheim & Hans Bratland Østerdal

Tittel: Utvikling et Automatisert System for Plukking i Ustrukturerte Omgivelser

Title: Development of an Automated Bin Picking System for Cluttered Environments

Oppgavetekst:

Presise 3D-kameraer er nyttige redskaper for å innhente informasjon om objekters plassering og orientering i rommet, mens robotarmer tilbyr høy fleksibilitet og manøverbarhet. I denne oppgaven skal disse komponentene kombineres, for å produsere et autonomt gjenkjennings-og-plukk system som er i stand til å identifisere og plukke objekter fra lagerbokser. Systemet skal tilpasses AutoStore's nåværende løsninger, og skal testes ut instituttets Agilus-celle.

1. Beskriv kinematikken av en robotcelle bestående av en robot med fastmontert 3D-camera og en robot en med fastmontert gripeløsning.
2. Lage et system for å gjenkjenne spesifikke objekter i 2D, og estimer plassering i bildet, samt bildeprosessering av dybdebilde.
3. Lage et system som fastslår gripeposisjon av objekter relativt til robotene i rommet.
4. Lage en gripeløsning tilpasset AutoStore's helautomatiske lager-og plukkesystem.
5. Implementere et autonomt system som er i stand til å gjenkjenne og plukke opp objekter fra tilsvarende lagerbokser som benyttes av AutoStore.
6. Gjennomføre praktiske eksperimenter hvor resultatene blir vurdert og diskutert.

Oppgave utlevert: 15.01.18

Innlevering: 11.06.18

Utført ved Institutt for Maskinteknikk og Produksjon, NTNU.

Trondheim, 2018-06-08

Olav Egeland

Professor Olav Egeland
Faglærer og veileder

Preface

This is a master thesis in Production Technology at NTNU, under the study program Mechanical Engineering with Mechanical and Industrial Engineering as specialisation. The thesis was written during the spring semester of 2018, and was conducted with the cooperation of AutoStore AS. Guidance and problem description was carried out by our supervisors Olav Egeland, and Lars Tingelstad.

Trondheim, June 11, 2018

[illegible] Hans Bratland Østerdal

Sondre Vadheim Hans Bratland Østerdal

Acknowledgment

We would like to thank the following persons for their help during the development of this Master's thesis.

Our main supervisor **Professor Olav Egeland** for constructing a problem description that was both challenging and interesting, and for the guidance given during the period.

We also want to show our gratitude to **Lars Tingelstad** for guidance and help with the problem formulation, and for valuable feedback during the development of the thesis.

We would also like to mention the employees at the institute lab for producing the physical components for our gripper module, and Ph.D candidate Adam Leon Kleppe for helping us with the Agilus robot cell.

The help and guidance have been greatly appreciated, and have contributed to a steady work process.

Lastly, we would like to thank all the fellow students at the office for providing a social and positive work environment.

S.V & H.B.Ø

Abstract

The amount online purchases conducted by consumers show no sign of declining, and the growth in retail sales have rapidly increased over the last decade. This growth is driving the demand for automated packing and shipping systems, that can perform the highly repetitive work of locating and transporting products from inventory bins to shipping bins. This master thesis addresses this possibility thought the development of an automated bin picking system.

The system was developed at the *Agilus* robot cell at the *Department of Mechanical and Industrial Engineering* workshop during the spring semester of 2018. The proposed system utilises a combination of computer vision modules in 2D and 3D, robotics, and deep neural networks. Two neural networks have been trained for the task of performing object detection on specific objects, and evaluated in terms of performance. The kinematic relation between the camera and the robotic manipulators of the cell have been described, and a grasping approach based on a deep neural network has been implemented. A fully automated bin picking system has been realised in the robotic cell at the institute workshop.

Each module of the final system is presented with the associated results. The individual modules are combined to one single system, that is capable of performing autonomous bin picking operations. The final results demonstrate that the system is able to perform the intended task with a relatively high success rate. The results from the individual modules are also presented, and a discussion is presented based on the obtained results. The results obtained from our experiments during practical work are discussed in context to the possibility of implementing a similar system in a real storage setting, where new products are added to the inventory list continuously. Finally, strengths and weaknesses of the proposed system are discussed, along with potential improvements for future work.

Sammendrag

Andelen nettbaserte salg blant forbrukere har økt kraftig det siste tiåret, og denne tendensen ser også ut til å vedvare i tiden fremover. Denne veksten driver etterspørselen etter automatiserte pakke- og forsendelsessystemer, som kan utføre det høyt repeterende arbeidet med å finne og transportere produkter fra lagerbeholdere til fraktbeholdere. Denne masteroppgaven adresserer denne muligheten gjennom utviklingen av et automatisert plukkesystem.

Systemet ble utviklet på *Agilus*-robotcellen ved *Institutt for maskinteknikk og produksjon* verkstedet i løpet av vårsemesteret 2018. Det foreslåtte systemet benytter en kombinasjon av datasynmoduler i 2D og 3D, robotikk, og dype nevrale nettverk. To nevrale nettverk har blitt trent for oppgaven med å utføre objektgjenkjennelse på bestemte objekter, og er blitt evaluert når det gjelder ytelse. Det kinematiske forholdet mellom det brukte kameraet og robotmanipulatorene i cellen er blitt beskrevet, og en gripleøsning basert på et dypt nevralt nettverk er blitt implementert. Et fullt automatisert plukkesystem har blitt realisert i robotcellen ved instituttverkstedet.

Hver modul i det endelige systemet presenteres med tilhørende resultater. De enkelte modulene er kombinert i ett enkelt system, som kan utføre autonome plukke operasjoner. De endelige resultatene viser at systemet er i stand til å utføre den tiltenkte oppgaven med en relativt høy suksessrate. Resultatene fra de enkelte modulene presenteres også, og en diskusjon presenteres basert på de oppnådde resultatene. Resultatene fra våre eksperimenter fra det praktisk arbeidet diskuteres i sammenheng med muligheten for å implementere et lignende system i en reell lagersituasjon, der nye produkter legges til produkt sortimentet kontinuerlig. Avsluttningsvis er styrker og svakheter i det foreslåtte systemet diskuteres, sammen med potensielle forbedringer for fremtidig arbeid.

Contents

List of Figures

List of Tables

List of Abbreviations

API	Application Programming Interface
ASRS	Automated Storage and Retrieval System
CAD	Computer-Aided Design
CNN	Convolutional Neural Network
Dex Net	Dexterity Network
Faster R-CNN	Faster Regional Convolutional Neural Network
GCP	Google Cloud Platform
GQ-CNN	Grasp Quality- Convolutional Neural Network
HDR	High Dynamic Range
RGB-D Image	Red, Green, Blue and Depth Image.
RoI	Region of Interest
ROS	Robotic Operating System
RPN	Region Proposal Network
SSD	Single Shot multibox Detector

Chapter 1

Introduction

1.1 Background

The amount online purchases conducted by consumers show no sign of declining, and the growth in retail sales have rapidly increased over the last decade. Global e-commerce sales are estimated to experience an increased growth of 265.11% in the year 2021 compared to 2014, with sales reaching 4.88 trillion US dollars worldwide [1]. According to a review of the Norwegian e-commerce behaviour for 2016 by *PostNord*, as much as 65% of the Norwegian population conducted online purchases every month, with an estimated national annual spending of 5.1 billion euros [2]. This was a national growth of 26% compared to the previous year in the percentage of the population who shopped online every month, and a 0.7 billion euro increase in the total national amount spent on online purchases. This growth is driving the demand for automated packing and shipping systems, that can perform the highly repetitive work of locating and transporting products from inventory to shipping bins. These operations are ideal for automation, but presents challenges related to object detection and grasping due to irregular item poses within the storage bin, large assortments, different sizes and weights, and different material and geometrical properties of the items. Human workers are typically still needed in the pick-and-placing operation from the storage container to the shipping container, which is not particularly desirable in tight labor markets. However, recent development in the field of *deep learning* has greatly improved robots ability to perform reliable classification of objects through computer vision, as well as determining robust grasping locations for transportation. This opens new opportunities for developers and companies that are capable producing reliable and robust automated bin picking systems for costumers, where the pick-and-place operation is central. The opportunities are also great for customers who successfully are able to implement an autonomous system, as it has the potential of reducing cost related to manual labor while increasing the productivity of the operation.

The traditional pick-and-place operations that utilises computer vision and robotics have mainly been related to industrial applications in the production industry. The picking operation is most often categorised either as a *structured* or a *semi structured* picking operation, where the position and orientation of the objects are fully or partly known. Pick-and-place operations related to inventory handling in e-commerce warehouses are referred to as *bin picking*, and is usually categorised as a *random* picking operation, where the objects are positioned at random locations and orientations within a storage bin. This bin picking operation has proven difficult to automate as it is hard to generalise patterns for successful object detection and grasping, due to the large variety and randomness within the bin. Several companies are developing and offering autonomous bin picking solutions that are based on the technology and possibilities introduced by deep learning [3], [4], [5].

1.2 Problem Formulation

Our collaborator, AutoStore AS, delivers automated storage and retrieval system (ASRS) solutions to companies, saving inventory space and reducing the amount of manual labor in the process. Optimisation of space is achieved by utilising mobile robots that bring inventory bins to the shipping area, where the products are manually localised and transferred to a shipping bin. This thesis will study the possibility of replacing the manual pick-and-place operation with an automated bin picking solution capable of recognising and picking specific objects. This will be done by examining the theory of the components that can be used in an autonomous bin picking system, and to produce a physical system based on the theory presented. The scope of this problem formulation is concretised in the following list of objectives:

- **Robot Kinematics**
  Describe the kinematics of a robotic system consisting of two robots, where one robot is equipped with a gripper and one robot is quipped with a 3D camera.
- **Computer Vision**
  Describe and develop a system capable of identifying and localising specified objects in 2D, and to process 3D images to extract relevant data.
- **Grasping**
  Develop a system capable to determining a suited grasping point given a 3D image of a scene, relative to the robotic manipulators.
- **Gripper**
  Develop a gripper solution compliant with AutoStore's automated storage and retrieval system.
- **Physical Implementation of the System**
  Implement a complete autonomous system capable of recognising and grasping objects from a storage bin that is equal in size to the bins that are used at AutoStore.
- **Practical Experiments**
  Conduct practical experiments in order to evaluate the system, and to present a discussion based on the results.

1.3 Thesis Structure

The objective of this thesis is both theoretical and practical, and includes an extensive literature and theory research that covers the available technologies that can be applied in an automated bin picking system. The practical objective is based on this theory, and is used to create an automated bin picking system. The structure of this thesis can be summed up in the following list:

- **Chapter 1. Introduction**
  The background and motivation behind this thesis is presented, along with the problem formulation.
- **Chapter 2. Background and Theory**
  The background and theory behind a final proposed bin picking system is presented. The chapter is divided into a computer vision section, and a robotics section.
- **Chapter 3. Method**
  The methods used for testing and evaluation of the practical experiments are presented.
- **Chapter 4. Results**
  The results based on the Method chapter are presented.
- **Chapter 5. Discussion**
  Discussions and personal thoughts based on the findings in the Results chapter are presented. Weaknesses and strengths from the practical work are discussed, and possible improvements are proposed.
- **Chapter 6. Concluding Remarks and Future Work**
  The work of this thesis is concluded, and suggestions to further improve the system are presented for future work.
- **Appendix**
  Relevant appendixes for the thesis are listed.

Chapter 2

Background and Theory

Creating an autonomous bin picking system requires several individual components, working and communicating together in order to fulfill the intended operation. This chapter will cover the theory behind the necessary components in a potential bin picking system. The chapter is separated in a computer vision section, and a robotics section. The computer vision section covers aspects of 2D and 3D image transformations, and object detection using deep neural networks. The robotics section covers the kinematic aspects of robot movement, the ROS communication system, and grasping.

2.1 Computer Vision

Computer vision is the field of study in which computers gain a higher level of understanding of their surroundings, though the use of digital sensors. Processing of the acquired data is often required, in order to enhance or remove certain features that are of importance for the computer when it interprets the information. The data may contain 2D or 3D information of the scene, and can be represented in an RGB image, depth image, or a point cloud. Two central terms used within computer vision are *object detection* and *object segmentation*, which we define as follows:

- **Object Detection**
  Object detection is the process of locating and classifying objects within an image, and to output class probabilities along with the location of the objects within the input image.
- **Object Segmentation**
  Object segmentation is the process of locating and extracting specific regions and boundaries that may be of interest in an image.

Since computer vision was introduction in the 1950s, there have been large advancements in the field of object detection and object classification, and is today applied in numerous applications in our society

2.1.1 2D: Perspective Transformation

Capturing 2D images involves transforming a 3-dimensional scene onto a 2-dimensional plane. This transformation is referred to as a *perspective projection*, where information regarding the depth of the scene is lost. Several camera models have been proposed, but the *central-projection model* is often used for the purpose of computer vision [6]. The model is illustrated in Figure 2.1, where the rays converge to the origin of the camera frame $\{C\}$.

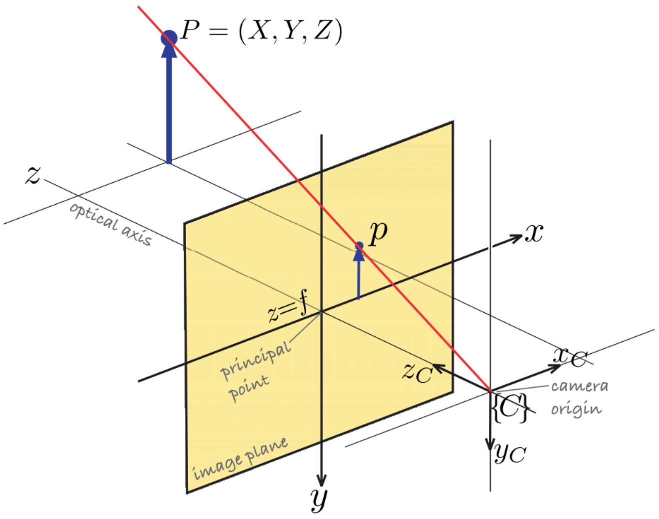


Figure 2.1: Illustration of the central projection model, where the image plane is located in front of the camera's origin. The plane is located at $z = f$, on which a non-inverted image is formed. [6]

The image is projected onto the image plane which is located at $z = f$, where f is the focal length. The image is non inverted, opposed to images obtained from a traditional pinhole model. The world coordinates $\mathbf{P} = (X, Y, Z)$ is projected to the image plane $\mathbf{p} = (x, y)$ by

$$x = f\frac{X}{Z}, \quad y = f\frac{Y}{Z}$$

which is the perspective projection. As a result of this transformation from 3-dimensional space to 2-dimensional space, parallel lines in the real world are projected to straight lines that intersect at a vanishing point. The transformation is not one-to-one, and it is therefore not possible to uniquely determine (X, Y, Z)

given (x, y). The image plane point can be expressed in homogeneous form $\tilde{\mathbf{p}} = (x', y', z')$ where

$$x' = \frac{fX}{z'}, \quad y' = \frac{fY}{z'}, \quad z' = Z$$

or in matrix form

$$\tilde{\mathbf{p}} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

where the non-homogeneous image plane coordinates are

$$x = \frac{x'}{z'}, \quad y = \frac{y'}{z'}.$$

Normalised image plane coordinates are achieved when $f = 1$. The world coordinates can also be written on homogeneous form ${}^{C}\tilde{\mathbf{P}} = (X, Y, Z, 1)^T$, resulting in a linear representation of the perspective projection:

$$\tilde{\mathbf{p}} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} {}^{C}\tilde{\mathbf{P}}.$$

The perspective projection makes it possible to establish the relationship between the pixel coordinates in an image and the image coordinates in the image plane. The pixel coordinates can be expressed as

$$u = \frac{x}{\rho_w} + u_0, \quad v = \frac{y}{\rho_h} + v_0 \tag{2.1}$$

where ρ_w and ρ_h are the width and height of each pixel respectively, and (u_0, v_0) is the principal point where the optical axis intersects the image plane. The pixel coordinates from Equation 2.1 can be converted into homogeneous coordinates $\mathbf{S} = (u', v', w')$, given the homogeneous coordinates of the image plane:

$$\mathbf{S} = \begin{pmatrix} 1/\rho_w & 0 & u_0 \\ 0 & 1/\rho_h & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}.$$

2.1.2 3D: Structured Light

Structured light is a method of acquiring 3D information in an image, that was first presented in computer vision literature in the 1990s. The operating principle is to project a known pattern onto a scene of interest, and thereafter measure the distortion in the pattern to calculate the depth information for each pixel in the image. Figure 2.2 illustrates the general concept, where a spatially varying 3D structured illumination is generated by a special projector modulated by a spatial light modulator.

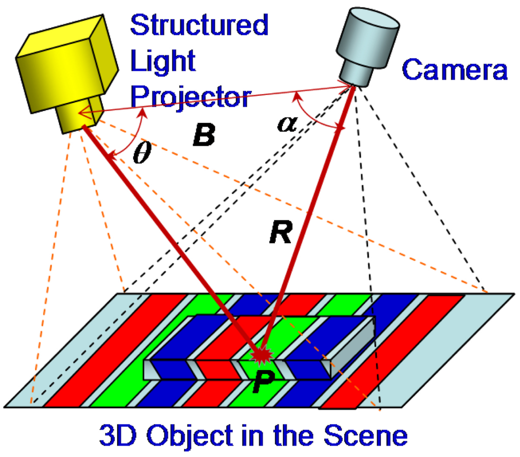


Figure 2.2: Illustration of the principle of structured light. Trigonometry can be used to calculate the distance between the camera and a point in the scene. [7]

Trigonometry can be applied in order to calculate the distance between the imaging sensor, the structured light projector, and the object surface point [8]. The principle of the triangulation can be expressed as

$$R = B\frac{\sin\theta}{\sin(\alpha+\theta)}$$

where R is the distance between between camera and a point P in the scene, B is the fixed distance between the camera and the projector, and θ and α are the relative angles between the light beam to P and the projector and camera. An essential aspect of this triangulation based 3D imaging, is the technique used to differentiate a single projection light spot from the acquired image during a 2D projection pattern. There exists several techniques for this purpose, which differs in the achievable resolution and accuracy of a system. One popular technique is the *phase shift* technique, due to its high speed and accuracy.

Phase Shift

Phase shifting involves projecting several sinusoidal fringe patterns over the scene, and measure the pixel intensity for each projection at each pixel point. This method has several advantages, such as high resolution, high speed 3D measurements, and is less sensitive to surface reflectivity and variations [9].

The intensities for each pixel (x, y) of three separate projected fringe patterns can be denoted as

$$\begin{aligned}
I_1(x,y) &= I_0(x,y) + I_{\text{mod}}(x,y)\cos(\phi(x,y) - \theta)\\
I_2(x,y) &= I_0(x,y) + I_{\text{mod}}(x,y)\cos(\phi(x,y))\\
I_3(x,y) &= I_0(x,y) + I_{\text{mod}}(x,y)\cos(\phi(x,y) + \theta)
\end{aligned}$$

where $I_1(x,y)$, $I_2(x,y)$ and $I_3(x,y)$ are the intensities of the three fringe patterns, $I_0(x,y)$ is the DC component, $I_{\text{mod}}(x,y)$ is the modulation signal amplitude, $\phi(x,y)$ and θ is the phase and the constant phase shift angle, respectively.

In order to obtain absolute phase pixel by pixel, a temporal phase unwrapping algorithm is applied. Instead of comparing a pixel's intensity to its neighbours, the algorithm compares information from other phase values at the same camera pixel. The phase information $\phi(x,y)$ can be retrieved, or unwrapped, from the intensities in the three fringe patterns:

$$\phi' = \arctan\left(\sqrt{3}\frac{I_1(x,y) - I_3(x,y)}{2I_2(x,y) - I_1(x,y) - I_3(x,y)}\right).$$

Discontinuities may appear in the arc tangent function at intervals of 2π, and can be removed by adding or subtracting multiples of 2π on the $\phi'(x,y)$ value:

$$\phi(x,y) = \phi'(x,y) \pm 2k\pi$$

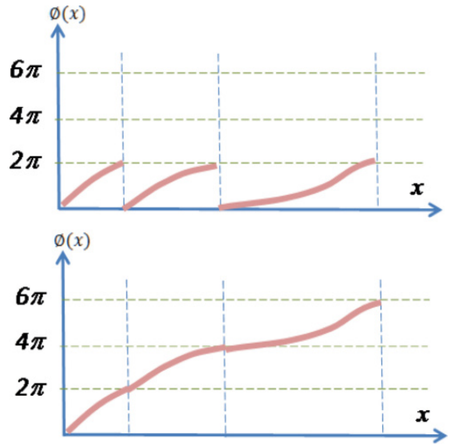
where k is an integer that represents the projection period. The 3D coordinates (x,y,z) can be obtained on basis of the difference between the measured phase $\phi(x,y)$, and the phase value from a reference plane. The result is visualised in Figure 2.3, along with the phase unwrapping process. The depth Z can be calculated as

$$\frac{Z}{L-Z} = \frac{d}{B}, \quad \longrightarrow \quad Z = \frac{L-Z}{B}d$$

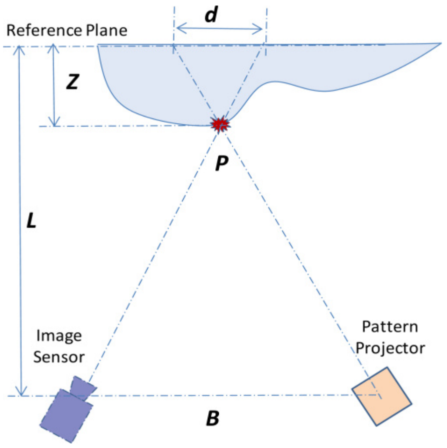
which simplifies to

$$Z \approx \frac{L}{B}d \propto \frac{L}{B}(\phi - \phi_0)$$

where L is the distance from the camera to the reference place, and B is the distance between the camera and the projector. d is the distance between the light rays from the projector and the camera through the point P on the interference plane.



(a) Phase unwrapping process.



(b) Structured light calculation of depth in a scene.

Figure 2.3: Illustration of the phase unwrapping process, and a scene captured by a camera and a pattern projector. [7]

2.1.3 Processing Depth Images

An acquired depth image may often experience some degree of deviations and noise within the measurements. This might be due to numerous reasons, such as random noise, surface properties of objects in the scene, or movement. Post processing images may reduce the amount of uncertainty in an image, and help accentuate features in the image that might be of importance.

2.1.3.1 HDR: High Dynamic Range

High dynamic range imaging, or HDR imaging, is a technique in which several separate images are combined to increase the overall dynamic range of luminosity in the image [10]. This is usually achieved by capturing a set of images of a scene, and adjusting the iris (aperture) of the camera between each frame. The aperture determines the opening of the lens, and therefore the amount of light being absorbed at the camera sensor. HDR imaging can be applied to generate both RGB images and depth images. If a depth image is acquired with the structured light technique, the degree in which a surface reflects or absorbs the emitted light beams will influence the quality of the reconstructed depth image, as the light beams might be under- or overexposed in the image. The HDR technique allows the camera to capture RGB images of the projected light beams on dark areas at a high iris setting, and bright areas when at a low iris setting. The final HDR image combines the relevant data captured in the RGB images to create an improved depth image, as can be seen in Figure 2.4.

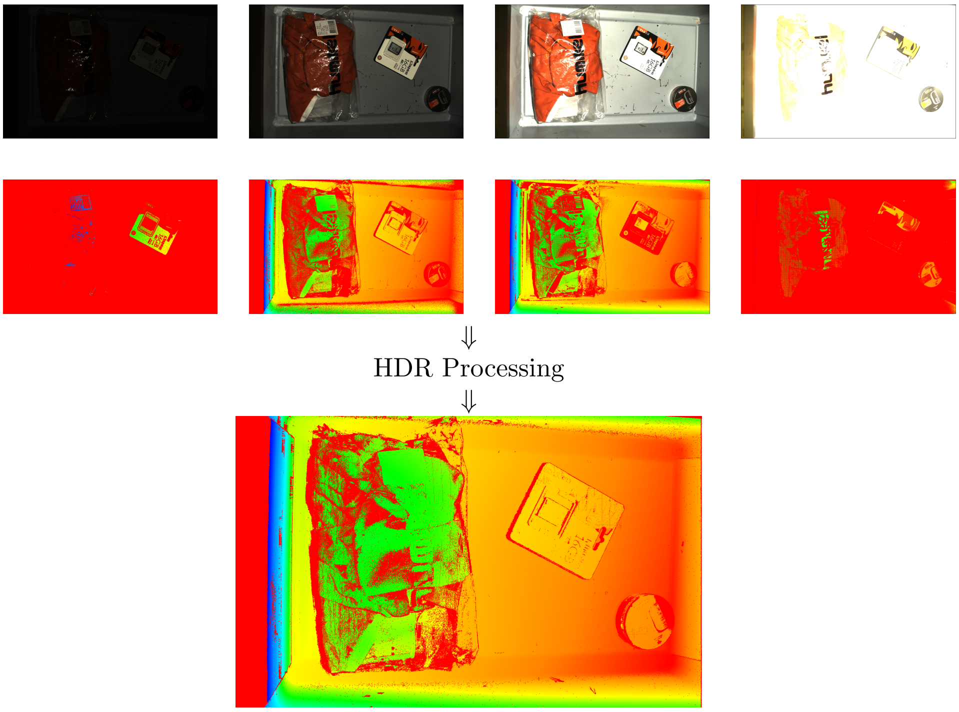


Figure 2.4: The two first rows shows the initial RGB and depth images with iris set at 13, 20, 30 and 70, from left to right. The processed image can be seen at the bottom, where a high dynamic range is obtained. Note that the images in the first row are not the actual structured light images used to obtain the depth image, but the RGB images that were taken with the same iris settings.

2.1.3.2 Bilateral Filtering

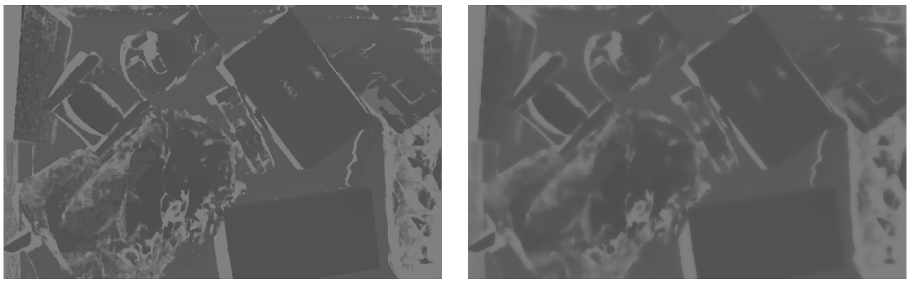
Bilateral filtering is a method for smoothing an image, while at the same time preserving the edge features. The filter is often applied in 2D and 3D computer vision for the purpose of filling holes and to reduce noise, and to accentuate edges. It is a non-linear filter where the output is a weighted average of the input [11]. Initially the filter starts with a standard Gaussian filter with a spatial kernel f, but the weight of each pixel also depends on a function g in the intensity domain. g ensures that the weight of pixels with large intensity differences are reduced. The resulting filtering J_s of a pixel s can be expressed as

$$J_s = \frac{1}{k(s)} \sum_{p \in \Omega} f(p-s)\, g(I_p - I_s)\, I_p$$

where Ω is all the pixels in the image, p is the neighbourhood pixels around s, and I_s and I_p are the intensity of the pixel and the neighbourhood pixels, respectively. $k(s)$ is a normalisation term on the form

$$k(s) = \sum_{p\in\Omega} f(p-s)\, g(I_p - I_s).$$

The result of a bilateral filtering operation on a depth image can be seen in Figure 2.5, where the edges are preserved while the overall image is smoothened.



(a) Depth image before bilateral filtering. (b) Depth image after bilateral filtering.

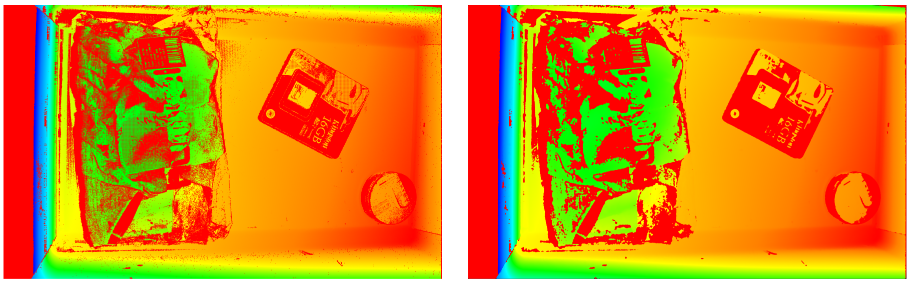
Figure 2.5: Illustration of the effects of a bilateral filter applied to a depth image to smoothen the image while preserving the edge features.

2.1.3.3 Median Filter

The median filter is a commonly used filter for 2D and 3D image processing, and is often applied for noise reduction or smoothing. It is widely used in digital image processing, due to its ability to reduce noise, while still preserving edges in the image. The filter considers each pixel in an image and replaces the value by the median of the surrounding neighbourhood given by a mask. The median filter process can be expressed as

$$g(x,y) = \mathrm{med}\{f(x-i,y-j), i,j \in W)$$

where $f(x,y)$ is the original pixel value and $g(x,y)$ is the new pixel value at position (x,y). W is a two dimensional mask with a defined size given by i,j. Figure 2.6 illustrates the before and after effect of a median filter, where much of the noise is replaced with more representative values.



(a) Depth image before median filtering. (b) Depth image after median filtering.

Figure 2.6: Illustration of the effects of a median filter applied to a depth image to reduce noise. This reduction can especially be observed around the area of the t-shirt. The mask has a size of 6×6 pixels.

2.1.3.4 Outlier Filter

Outlier filters are commonly used filters for reducing the noise in depth images and point clouds, where the outliers are points in the scene that are not considered as a part of an object. There are several approaches and algorithms that have been proposed for the purpose of this filtering. One typical implementation of this filter, is to calculate the average distance to each points k nearest neighbours. The resulting dataset has a mean and a standard deviation that is assumed to be Gaussian distributed. All the points that have a mean distance to its k nearest neighbours that is greater then the mean and standard deviation of the complete dataset, are considered to be outliers and therefore removed from the image.

2.1.4 Eye-in-Hand Calibration

The problem of identifying the relative position and orientation between a static mounted camera on a robotic hand and a known frame, can be solved by performing an eye-in-hand calibration. Following [12], calibration can be achieved by taking a series of images of a static calibration object, where the pose and orientation of the mounted camera is changed between each image. The corresponding base to hand transformations of the robotic manipulator are recorded at each iteration. Figure 2.7 illustrates the problem at hand, where **B** is the transformation between the hand coordinate frame to the robot base. **A** is the transformation from the camera coordinate frame to the world coordinate frame, and **Z** is the transformation between the base and world. Finally, **X** denotes the transformation between the hand and camera frame.

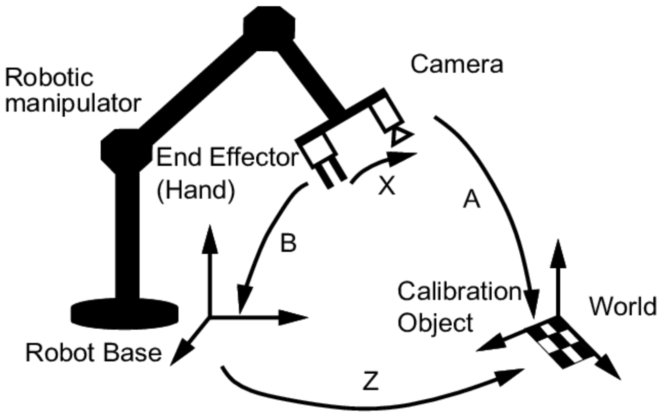


Figure 2.7: Illustration of the Hand-Eye calibration, where a camera (eye) is mounted at the end-effector (hand) of a robotic manipulator. Images of a stationary calibration object is acquired at several different robot poses, and used to estimate the distance **X**. [13]

Assuming that the transformation **B** are known, and the transformation **A** can be calculated by a calibration software, the problem can be formulated as:

$$\mathbf{AX} = \mathbf{ZB}, \tag{2.2}$$

where **X** and **Z** are unknown. The transformation **AX** is the transformation from the robotic hand to the calibration object via the camera, and **ZB** is the transformation from the robotic hand to the calibration object via the robot base. Equation 2.2 can be solved by passing a series of matrices $\mathbf{A_i}$ and $\mathbf{B_i}$ containing the relevant transformation for image i, and solving **X** and **Z**:

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

This set of equations corresponds to solving

$$\mathbf{AX} = \mathbf{XB} \tag{2.3}$$

where $\mathbf{A} = \mathbf{A}_i^{-1}\mathbf{A}_{i+1}$ and $\mathbf{B} = \mathbf{B}_i^{-1}\mathbf{B}_{i+1}$. The Equation 2.3 is called a Sylvester equation, and can be solved with respect to **X** though numerical or methodological approaches.

2.1.5 Deep Neural Networks

The concept of neural networks was first introduced in the 1940's by Warren S.McCulloch and Walter Pitts in the paper *A Logical Calculus Of The Ideas Ommanent In Nervous Activity* [14], and was inspired by the structure of the human brain. There are large varieties of structures among deep neural networks, but the basic principle is tho similar. The following section is related to deep neural networks in computer vision, but the general concepts may also be applicable for other deep neural network applications. A standard deep neural network consists of several layers of neurons, or nodes, that are connected between each layer [15]. Figure 2.8 shows a deep neural network consisting of an input and an output layer, and one *hidden layer.*

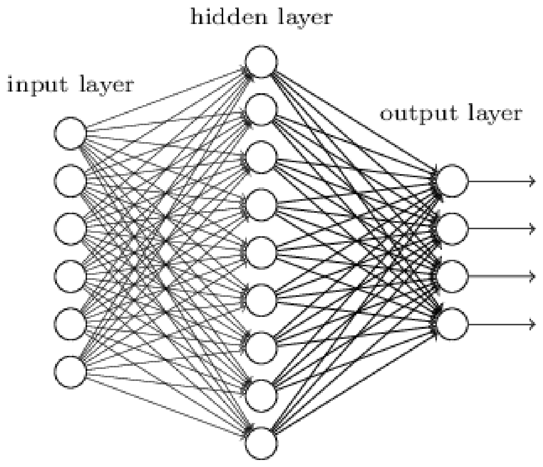


Figure 2.8: Illustration of a fully connected deep neural network structure with one hidden layer. [16]

The node is the basic computational unit of the network that outputs a single value based on one or several inputs from nodes in previous layers. Each input has an associated weight w depending on the relative importance of the connection. The neurons apply a function f to the sum of inputs, which is referred to as its *activation value.* This value is fed to the next layer of neurons, where the weighted sum from all connections to a particular neuron is computed along with its bias. This sum might be any positive or negative number. However, it is often desirable to represent the activation value as a value between 0 and 1, which can be achieved by applying the Sigmoid function to the weighted sum. In doing so, a very negative inputs results in an activation value close to zero, and high values results in an activation value close to one, as illustrated in Figure 2.9. The Sigmoid function is defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

where x is a given input value.

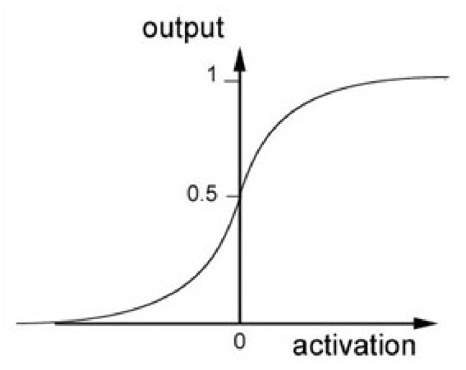


Figure 2.9: Illustration of the Sigmoid function. [17]

The actuation values of neurons in a given hidden layer in the network can be expressed in matrix form as

$$\mathbf{a}_i = \sigma(\mathbf{W}\mathbf{a}_{i-1} + \mathbf{b}), \quad i = 1, \cdots, N$$

where $\mathbf{a}$ is the actuation values for the neurons in a given layer i, $\mathbf{W}$ is the weight value matrix between the connections from layer $i-1$ to layer i. $\mathbf{b}$ contains the biases for each neuron, which are constant values that are added to each neuron. The biases increases the flexibility of the learned network, by shifting the Sigmoid function along the x-axis. The last layer of the network is the classification layer, where the neuron with highest value is the most probable class in an input image. During the training phase of the network, the weight and bias parameters are tuned in order to achieve the correct classification between the input and the output. This is a computational expensive operation, which often involves tuning and adjusting up to hundred million parameters [18].

2.1.5.1 Convolutional Neural Network

A *Convolutional Neural Network* (CNN) is a type of neural network structure that is often used in classification tasks related to computer vision. Krizhevsky et al, demonstrated the accuracy of the network structure in 2012 [19], where it won the computer vision competition *ILSVRC-12* [20], with significant margin.

The CNN structure usually consists of one or more characteristic convolutional layers, followed by one or more pooling and fully connected layers. This network structure takes advantage of the 2D input structure, by local connections and tied weights, followed by pooling, which results in translation invariant features.

Convolution Layer

The initial step of a convolution neural network is the convolution layer. Here, a given feature is projected on the input image, creating a stack of processed filtered images. The projected feature structure is projected over each pixel in the input image, and the rate of accuracy within the structuring element is mapped onto a new filtered image. This process can be observed in Figure 2.10, where the mapped pixel represents the weighted sum of itself and nearby pixels. The concept of the convolution layer is to quantify the probability for a given feature to be present in the image, represented by the filtered output image. Pixel areas with relative high mapped values, indicates that the given feature is present in the given area of the image. The process can be repeated for a set of different feature structures, creating a stack of filtered images.

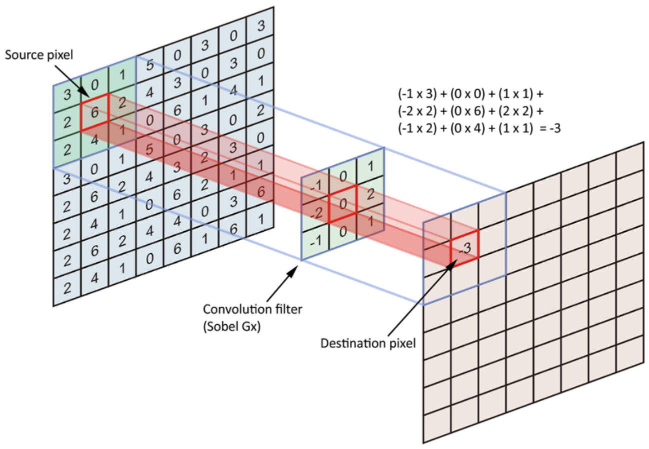

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

Figure 2.10: Convolution operation performed on a pixel. A source pixel is filtered with a convolution filter, and the new value is mapped in a new image. [21]

Pooling Layer

Pooling is often referred to as sub-sampling, and is an operation that extracts the most important information present in an image. This is normally achieved by *max pooling*, where the structuring element extracts the highest value within its boundaries as it moves through the image. This reduces the amount of information and size of the network, but retains the most important information in the an image, due to the fact that close outgoing connections usually contains similar information. The operation makes the information invariant to translation and rotation, and enables deeper layers to detect translated and rotated features. The size of the pooling structure is defined manually, where a too large filter might lead to too generalised information. A visual illustration of the pooling process can be seen in Figure 2.11, where a 2×2 max pooling filter is applied to an image.

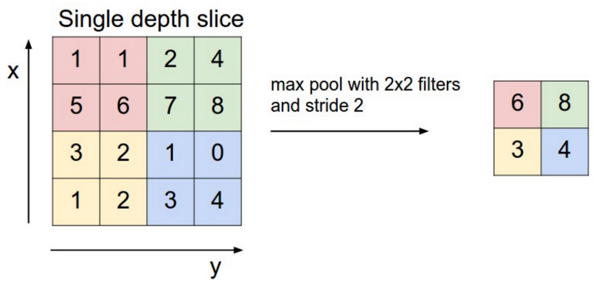


Figure 2.11: Max pooling with 2×2 filter and stride of 2. [22]

Fully Connected Layer

The last layer in a CNN is usually a fully connected layer, where high-level features from the previous layers are used to learn non-linear combinations. The number of neurons in the last layer is equal to the number of classes that the network is able to classify. A Softmax function is usually applied, in order to obtain a probabilistic representation of the presence of each class in the image. The Softmax function is defined as

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}} \tag{2.4}$$

where $\mathbf{z}$ is the input vector with n elements, and i is the index of the output units from $i = 1, \cdots, K$. Due to the nature of Equation 2.4, the output from the function is always between 0 and 1, and all outputs always sums up to 1 as illustrated in Equation 2.5. The output can be thought of as the network's certainty of presence for for each of the n classes in an image, where the class with highest output is most lightly to be present in the input image of the network.

$$\begin{bmatrix} 1.5 \\ -0.6 \\ -0.2 \\ 0.4 \end{bmatrix} \Longrightarrow Softmax function \Longrightarrow \begin{bmatrix} 0.61 \\ 0.07 \\ 0.11 \\ 0.2 \end{bmatrix}. \tag{2.5}$$

Figure 2.12 illustrates the final fully connected layer of a convolutional neural network, where the network is used to classify two different classes.

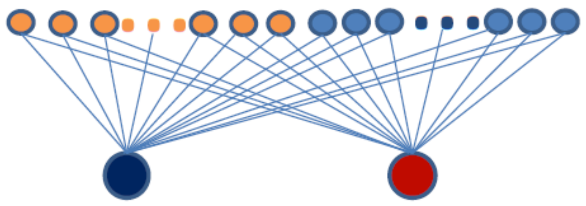


Figure 2.12: A fully connected layer where both neurons in the final layer are connected to all the neurons in the previous layer. [23]

2.1.5.2 Hyperparameters

Hyperparameters are variables that are chosen before initialising the training, that influences the networks ability to learn form the input data. This is important in order to achieve a good generalisation of the data pattern so that the model best can solve the deep learning problem. Hyperparameters effects the higher level properties of the network, such as the learning rate and complexity. This section will cover some of the most important hyperparameters, and cover how they effect the networks performance during training [24].

2.1.5.2.1 Loss Function

The *loss function* is used in the training phase of the network, and serves as an indicator of the accuracy of the network during training. It compares the difference between the predicted classification by the system, and the actual input label. Through iterations of training, the network will try to reduce the value of the loss by adjusting the parameters of the network. There are several different types of loss functions, as seen in Table 2.1, where $\mathbf{y}$ is the true label as one-hot encoding, $\hat{\mathbf{y}}$ is the true label as $+1/-1$ encoding. $\mathbf{o}$ represents the output of the last layer of the network, $^{(j)}$ denotes the jth dimension of a given vector, and σ denotes the classification and/or localisation loss. The classification loss concerns the proposed object label, while the location loss concerns the predicted placement of the object in an image.

Commonly Used Loss Functions		
Symbol	Name	Equation
$\mathcal{L}_1$	L_1 loss	$\|\|\mathbf{y}-\mathbf{o}\|\|_1$
$\mathcal{L}_2$	L_2 loss	$\|\|\mathbf{y}-\mathbf{o}\|\|_2^2$
$\mathcal{L}_1 \circ \sigma$	Expectation loss	$\|\|\mathbf{y}-\sigma(\mathbf{o})\|\|_1$
$\mathcal{L}_2 \circ \sigma$	Regularised expectation loss	$\|\|\mathbf{y}-\sigma(\mathbf{o})\|\|_2^2$
$\mathcal{L}_\infty \circ \sigma$	Chebyshev loss	$\max_j \|\sigma(\mathbf{o})^{(j)} - \mathbf{y}^{(j)}\|$
hinge	Hinge (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^j \mathbf{o}^{(j)})$
hinge^2	Squared hinge (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^j \mathbf{o}^{(j)})^2$
log	Log (cross entropy) loss	$-\sum_j \mathbf{y}^{(j)} log \sigma(\mathbf{o})^{(j)}$
log^2	Squared log loss	$-\sum_j [\mathbf{y}^{(j)} log \sigma(\mathbf{o})^{(j)}]^2$

Table 2.1: Table containing some of the most commonly used loss functions. [25]

2.1.5.2.2 Gradient Descent

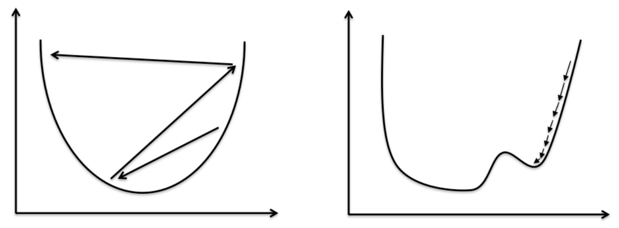
Gradient descent is an optimisation algorithm used to adjust the parameters in the network. The optimisation is achieved by minimising the loss function, which is dependent on the weights and biases of the network. Minimising the loss function with gradient descent implies finding and adjusting the partial derivatives of the weight and bias parameters. The updated parameters can be expressed as

$$\theta_{t=\theta_{t-1}-\epsilon_t \nabla \mathcal{L}(\theta_{t-1})}$$

where θ is a given parameter, ϵ is the learning rate of the network and $\mathcal{L}$ is the loss function that is to be minimised with respect to θ.

2.1.5.2.3 Learning Rate

The *learning rate* influences the networks rate of change with respect to the weights and biases, during the training phase. It is an important hyperparameter to tune, as it drastically changes the training dynamic of the whole network. A high learning rate results in large adjustments of the weights in the direction of the gradient of the mini batch. This may lead to the network overshooting the optimal weights that minimises the loss function. A low learning rate results in small adjustments of the weights, and as a result, it requires more training steps. Figure 2.13 illustrates how the learning rate affects the steps towards the global minimum of the loss function. A high learning rate will enable the weights to jump over a sub optimal local minimum, but never converge to a specific minimum as shown in Figure 2.13 (a). A low learning rate allows the network to converge to a minimum, but the optimisation may get stuck at a sub optimal local minimum, as shown in Figure 2.13 (b).



(a) A high learning rate during training.

(b) A low learning rate during training.

Figure 2.13: Comparison between the effect of a high and a low learning rate during training. A high learning rate will make large adjustments in the parameters to reach the minimum of the loss function, while a low learning rate makes small adjustments to reach the same minimum. [26]

It is therefore often desirable to use a decreasing learning rate, to overcome this issue. By first using a high learning rate to rapidly converge towards the global minimum, and a smaller learning rate to converge at a lower scale. This approach reduces training time, and ensures fine tuning of the weights at the end of the training phase. Bergstra and Bengio [27] suggests a method that continuously reduces the learning rate ϵ_t, as the amount of iterations t passes a given threshold $\mathcal{T}$ with an initial learning rate ϵ_0:

$$\epsilon_t = \frac{\epsilon_0 \mathcal{T}}{\max(t, \mathcal{T})}$$

2.1.5.2.4 Mini-Batch Size

The *mini-batch size* $\mathcal{B}$ determines the number of images passed though the network before adjusting the weights and biases according to the loss function. The direction of the gradient is calculated for each image, as it passes though the network. The average of all individual gradients is calculated after completing the batch, and the weights are modified according to this average. Mini-batch size $\mathcal{B}$ influences the computing time by enabling fast matrix calculations, and reduces the effect of noise in the labeled training set. Larger $\mathcal{B}$ yields faster computation as long as the GPU's have enough memory, thus $\mathcal{B}$ can be optimised independent of the other hyperparameters.

2.1.5.2.5 Number Of Training Iterations

The *number of training iteration* $\mathcal{T}$ is a hyperparameter that is not necessarily predefined before training, as it is a measure of how many times the weights have been updated. As mentioned in Section 2.1.5.2.1, the theoretical goal is to train the network until a number of iterations is reached where the loss function is minimised. In practice, this will not yield satisfying results as consequence of a overfitting issue. The ideal number of iterations $\mathcal{T}_{\text{ideal}}$ is reached when the accuracy of the validation set reaches its maximum. The loss function for the training set might continue improving as $\mathcal{T}$ increases, but in reality the network is losing its ability to generalise and recognise features in the validation set. To counteract this problem, the trained model should be saved at regular intervals as the training phase progresses.

2.1.5.2.6 Momentum

The *momentum* β is a parameter that influences how much the weights are updated, depending on the values from the previous iteration. While using gradient descent during training, each step is defined by Equation 2.6 and Equation 2.7, where the step is dependent on the current gradient and a proportion of the previous step. This allows weights to be updated even if the gradients becomes zero, and allows it to pass local inflection points and small local minimums. The size of the momentum decides how large local minimum curves the model can recover from.

$$\theta_t = \theta_{t-1} - v_t \tag{2.6}$$

$$v_t = \beta v_{t-1} + \epsilon_t \nabla \mathcal{L}(\theta_{t-1}) \tag{2.7}$$

2.1.5.2.7 Weight Initialisation

Weight initialisation is the initial value for the weights and biases in the network. A common practice in the early years of neural network was to set the weights with a small random deviation around 0, based on the assumption that half of the final weights would be negative and other half positive. As the neural networks increased in size, this method often caused signals to vanish or explode depending in the deviation around zero. Glorot and Bengio. [28] suggested a popular used technique to avoid this issue, by selecting the variance of the distribution around zero based on the network size:

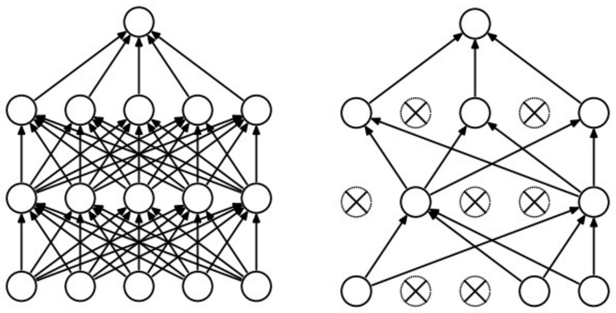
$$\mathrm{Var}(W) = \frac{2}{n_{\mathrm{in}} + n_{\mathrm{out}}}$$

where n_{in} and n_{out} are the number of inputs and outputs of the layers.

In the recent years, neural networks have become more complex, causing a need for a better weight initialisation in order to avoid vanishing and exploding signals. Kumar [29] suggested several techniques based on the activation function used in the network, while Kotutwar and Marchant [30] makes use of data statistics for weight initialisation, which proves to be more efficient in more practical approaches.

2.1.5.2.8 Dropout

Dropout is a method applied to reduce the issue of overfitting in neural networks used for classification, by randomly cutting connections between neurons during the training phase. This action prevents the units from co-adapting too much, meaning that the network can not rely on a given connection to be present at all times for a given class or feature. This forces the network to learn a redundant representation of an object, instead of relying on only a few characteristic features of the objects. This practice is illustrated in Figure 2.14, which results in increased robustness of the network in term of generalisation and reduced probability of overfitting.


(a) A standard neural deep neural network.

(b) A deep neural network after applying dropout.

Figure 2.14: Illustration of the before and after effect of applying dropout to a deep neural network, where random connections are cut. [31]

The paper *Dropout: A Simple Way to Prevent Neural Networks from Overfitting* [32], demonstrates the improvements in accuracy on different network structures while training on the well known datasets MNIST, SVHN, CIFAR-10, CIFAR-100 and ImageNet. In the demonstration, classification experiments were conducted with networks of several different architectures, while keeping all hyperparameters and the *dropout rate* fixed. The dropout rate represents the probability of retaining a connection in the network, and is usually set in the range of 0.5 to 0.8 for hidden layers and 0.8 for the input layers. Figure 2.15 illustrates the improvement in accuracy of the MNIST dataset on different network structures with and without dropout, and this is also the tendency for all the other datasets as well.

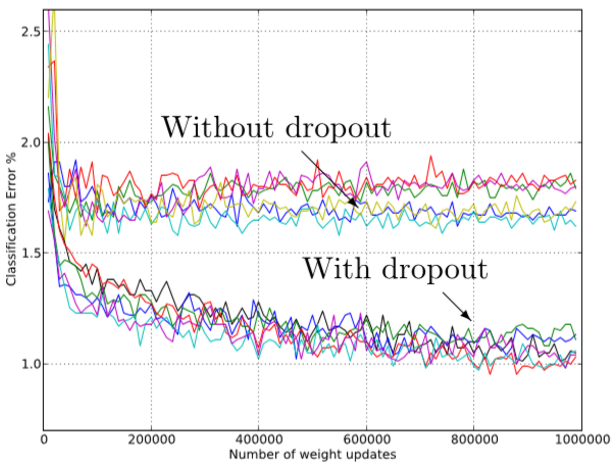


Figure 2.15: Classification error for different architectures with and without dropout on networks consisting of two to four hidden layers. A significant improvement in the classification error can be observed for all structures after dropout is applied. [32]

2.1.5.3 Overfitting

Overfitting of a network may occur when the parameters of the network are too adapted to the training data, and therefore the network loses its ability to generalise to new data. This may significantly effect the accuracy of a deep neural network based classifier, when classification is conducted on images that were not included in the training dataset. Models that are overfitted have failed to identify the general patterns that accurately differentiate the classes of the dataset, and instead characteristics of the training dataset have been memorised in the model. The presence of overfitting can be evaluated graphically by comparing the evaluation error and the test error during training, as shown in Figure 2.16. At the point where the validation error increases, the model starts to lose its ability to generalise the characteristic features of the training set. The resulting outcome is that the model achieves a high accuracy for the specific training set, but a low accuracy on new classifications that were not included in the training set.

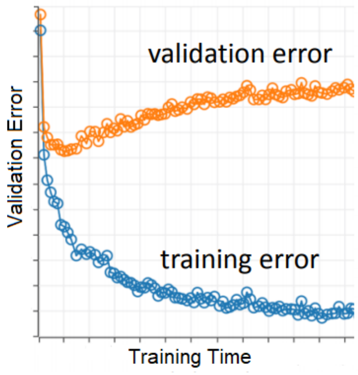


Figure 2.16: The presence of overfitting in deep learning models can be indicated by comparing the error of the validation dataset, and the training dataset. If the training error decreases while the validation error increases, the model shows signs at it is memorising rather then generalising the training data. [33]

Several actions can be applied to reduce the presence of overfitting in a model, such as adding more data to the training set, reducing the model complexity, and applying dropout.

2.1.5.4 Datasets

When deep learning within computer vision began to show promising results, it was mainly due to two factors: fast GPUs and availability of large amount of labeled training data. Deep learning models require high quality training data to achieve its potential, which often needs to be labeled or sorted by humans. This is a time consuming process, especially in supervised training where datasets usually has to be created manually or semi-automated. Furthermore, a dataset is made by collecting raw data, and structuring the information into a numerical representation that is suitable for deep learning algorithms. Datasets usually consists in pairs of input data and labeled or categorised output data, which represents the ground truth in the input data. To overcome the need of creating huge dataset for each specific task, large general training sets are used to train the model before the application relevant data is used to optimise the model for its application, as explained in Section 2.1.5.5. Datasets are usually divided into two parts, where about 80% is used for training, and the remaining section is used for testing or validation. Two popular datasets used for training are the *ImageNet* and the *COCO* datasets.

- **ImageNet**
  ImageNet is a large scale image database consisting of over 14 million images, where over 1 million of the images are assigned with bounding box annotation. The images in the dataset are distributed over 20 000 categories, with approximately 650 images per category [34].

- **COCO: Common Objects in Context**
  COCO is a large object detection and segmentation dataset, released by Microsoft. It consists of over 200 000 labeled images and 1 500 000 labeled objects. The dataset was created using Amazon Mechanical Turk, with focus on images containing contextual relationships and non iconic object views. The COCO dataset also contains a large amount of labeled objects in cluttered environments with a high number of instance per image [35].

2.1.5.5 Transfer Learning

Deep neural networks requires a massive amount of labeled data during training, in order to adjust weights and biases between the layers during training. The number of parameters might be in the range of hundred millions [36] and an extensive amount of labeled data and computational power is therefore necessary during training, both of which is often not accessible for many applications. Transfer learning is a way of reducing the dependence of the two, by reusing a pretrained network from a larger dataset [37]. Today there are several public available datasets of sufficient size for training large neural networks, such as COCO and ImageNet. The weight and biases learned from these datasets can be used for new target objects, due to the fact that neural networks learn general features early in the network structure, while specific features for classification are learned in the last layers. During transfer learning, only the last layers are trained for the classification task, which allows deep learning to be applied in applications where there are limited available labeled data. Figure 2.17 demonstrates the concept of transfer learning, where the classifier of the target task is trained after transferring the parameters from the source task.

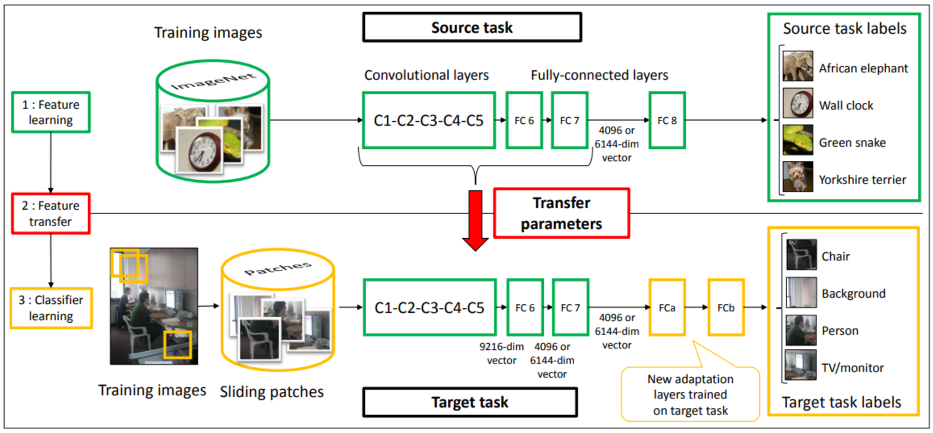


Figure 2.17: Illustration of how the layers and weights of a pre-trained model are transferred to a new classification application. During training, only the weights and biases of the last classification layers are trained. [36]

2.1.5.6 Deep Learning Architectures

Deep learning algorithms may have a large diversity in terms of architecture, due to the flexibility that is provided by the neural networks when creating an end to end model. This section will cover two commonly used architectures for object detection in 2D images.

2.1.5.6.1 Faster R-CNN: Faster Region based Convolutional Neural Network

The Faster R-CNN model is the third deep neural network based on the work of Girschick et al. in [38] and was presented in 2016 as *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks* [39]. This iteration introduced a region proposal network (PRN), that generates a set of object proposals with a corresponding objectiveness score from an image. Figure 2.18 illustrates the steps in the model, which drastically reduced the computation time from 2 seconds per image to 10 ms per image, compared to the previous released iteration. Faster R-CNN performed a state-of the-art result at VOC 2007 with a mAP score of 78.8%, when trained on the COCO, VOC07, and VOC12 datasets.

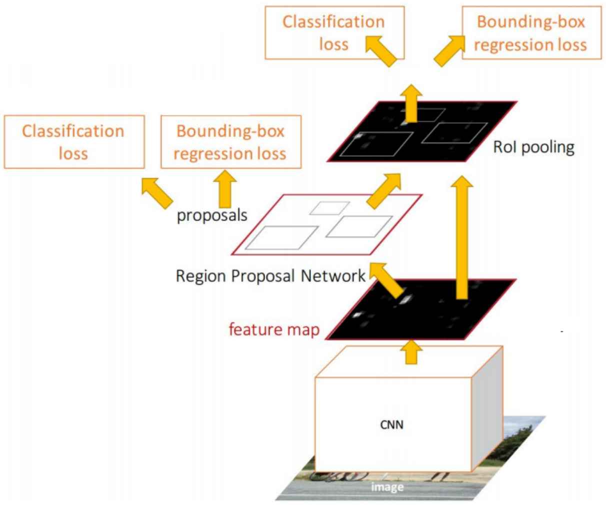


Figure 2.18: Illustration of a Faster R-CNN model. The region proposal module proposes regions of interest in the feature map, which reduces computation time in the classification module. [39]

The Faster R-CNN network consists of two separate modules, where the first is a deep fully convolutional network that proposes regions of interest. The second module is a Fast R-CNN detector that utilises the proposed regions of the first module.

Region Proposal Network

The main goal of the Faster R-CNN is to reduce computation time, by sharing computation between the Region Proposal Network (RPN) and the object detection network. The RPN takes an image as input and outputs rectangular bounding boxes of object proposals, where each proposal has its own objectness score. The region proposals represent a Region of Interest (RoI), and are generated by sliding a small network over the convolutional feature map output by the last shared convolutional layer. The input of this small network is a $n \times n$ spatial window of the input convolutional feature map, and each sliding window is mapped to a lower-dimensional feature. This feature is in turn fed as input to two sibling fully connected layers. The first layer is a box-regression layer (*reg*), and the other layer is a box-classification layer (*cls*). As a result of the sliding window approach, the fully connected layers are shared across all spatial locations in the image. This architecture is implemented with an $n \times n$ convolutional layer that is followed by two sibling 1×1 convolutional layers (*reg* and *cls*, respectively). Multiple region proposals are predicted at each sliding window location, where the maximum amount of proposals for each location is denoted as k. The *reg* layer has $4k$ outputs which contains the bounding box coordinates of k boxes, while the *cls* layer outputs $2k$ scores that estimates the probability of an object present in each proposal, or not. The term *anchor* represents the parameterization of k proposals relative to k reference boxes. An anchor is located at the center of a given sliding window, and is associated with a scale and aspect ratio, which can be seen in Figure 2.19.

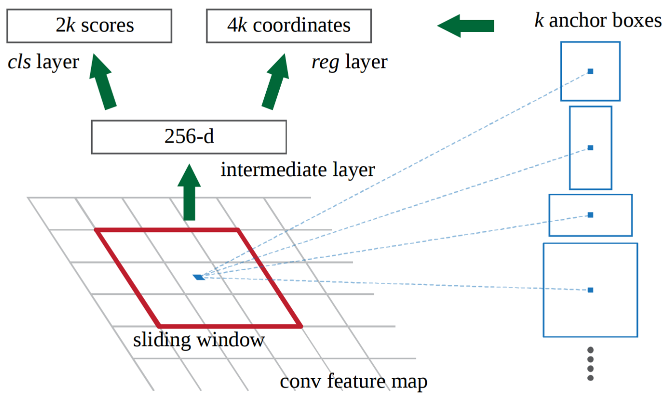


Figure 2.19: Illustration of the region proposal network of a Faster R-CNN network. The network outputs $2k$ scores as an estimation whether an object is present or not in the proposal, and $4k$ bounding box coordinates. [39]

Fast R-CNN Detector

The proposed regions as fed in to the second module, which is a Fast R-CNN detector [40]. For each proposal or RoI r, the forward pass outputs a class posterior probability distribution p, along with a set of predicted bounding box offsets relative to r. A detection confidence is assigned to r for each object class c, where the estimated probability is given by

$$\Pr(\text{class} = c \mid r) \triangleq p_c.$$

Further, a non-maximum suppression algorithm from a R-CNN model is applied for each class [38].

2.1.5.6.2 SSD: Single Shot MultiBox Detector

The Single Shot MultiBox Detector (SSD) was presented by Liu et al. [41], and is an object detection approach where the network is able to detect objects in images, using a single deep neural network. The model is based on feed-forward convolution networks that produces fixed size bounding boxes with corresponding scores, and predicts the presence of object classes inside each box. These boxes are applied on top of the feature map at multiple scales and aspect rations, as seen in Figure 2.20. The final classification is achieved by a non-maximum suppression step over all boxes.

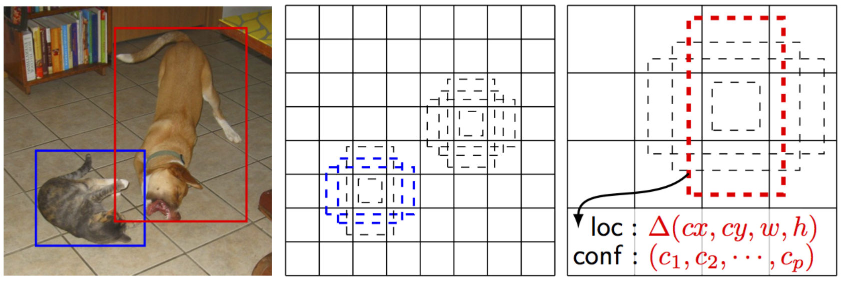

loc : $\Delta(cx, cy, w, h)$
conf : $(c_1, c_2, \cdots, c_p)$

Figure 2.20: The right image shows an input image with ground truth labels around the training objects. The center and right image illustrates the evaluation of a set of boxes of different aspect ratios and scales. [41]

The architecture of the network can be seen in Figure 2.21, where the first section consists of convolution layers taken from an image classification network *VGG-16* [42] referred to as the base model, truncated before the classification layer. This base model can be substituted with other feature extraction networks, such as *Inception-ResNet V2* [43]. The base model which performs feature extraction is followed by convolutional feature layers that decrease in size progressively, allowing predictions of detection at multiple scales.

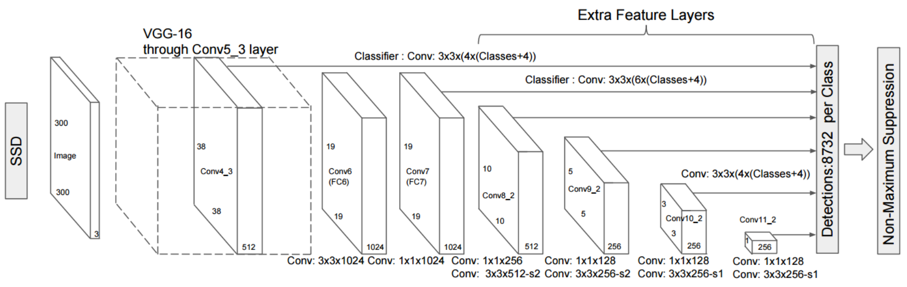


Figure 2.21: Illustration of the architecture of the SSD network, which uses a VGG16 base model, where several feature layers are added to the end of the network, which predicts the offsets to default boxes of different scales and aspect ratios and their associated confidences. [44]

For the feature layers of size $m \times n$ with p channels, a $3 \times 3 \times p$ small kernel predicts a score for a category, or a shape offset relative to the default box coordinates. At each of the $m \times n$ locations where the kernel is applied, an output value is produced. Each feature map cell is associated with a set of default bounding boxes, which tile the feature map in a convolutional manner. This connection ensures that the position of each box relative to its corresponding cell is fixed.

For each box out of k at a given locations, c class scores and the four offsets relative to the original default box shape are computed. This results in a total of $(c+4)k$ filters that are applied around each location in the feature map, yielding $(c+4)k \times m \times n$ outputs for a $m \times n$ feature map. Finally, the prediction is obtained by applying non-maximum suppression over all boxes.

2.2 Robotics

Robotics is concerned with the study of machines that can replace or aid humans in execution of different tasks, both in physical applications and in decision making. The following section will cover some of the most important aspects related to robotics applied for bin picking, such as motion, communication systems, and grasping.

2.2.1 Robot Kinematics

An industrial robot is a multifunctional manipulator that can be modeled as an open chain of rigid bodies (*links*), connected in series by kinematic *joints*. One end of the chain is mounted to a base, while an *end-effector* is mounted to the other end. The relationship between positions, velocities and acceleration of the links and end-effector of the robot is referred to the kinematics of the robot, and is obtained by composition of the elementary motions of each link with respect to the previous one.

This section will present some of the most important concepts related to kinematics of robotic manipulators. The theory presented in this section is derived from the book *Robotics - Modelling, Planning and Control* [45].

2.2.1.1 Joints

There are several types of joints that can be applied in robotics, but the two most commonly seen are the *revolute* and the *prismatic* joints.

Revolute Joints

Revolute joints creates a relative rotational motion between two links. There are two basic configurations of the revolute joint, depending on the relationship between the center line of the link and the axis of rotation. The axis of rotation can be coincident with the center line of the link, or it can be perpendicular to the center line of the link. For both cases, the revolute joint provides one degree of freedom, where the axis of rotation is in the Z direction.

Prismatic Joints

A prismatic joint creates a relative transnational motion between two links. Similar to the revolute joint, there are two basic configurations of the prismatic joint. The axis of translation can either be collinear with the fixed link, or it can be orthogonal to it. One degree of freedom is provided, and the axis of translation is in the Z direction.

2.2.1.2 Rotation Matrix

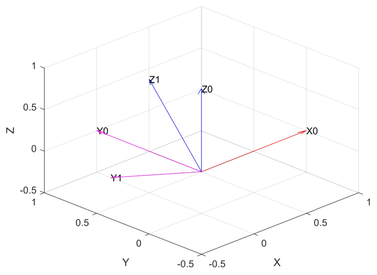
The rotation matrix is a method of representing rotation, or *orientation*, of a frame relative to a reference frame. This is done by projecting unit vectors $\mathbf{x}'\mathbf{y}'\mathbf{z}'$ of a frame onto the reference frame axis $\mathbf{xyz}$. The degree of projection onto each axis varies at a constant rate, as the degree of rotation about an axis varies. This relation is represented with the rotation matrix about the x, y and z axis of the reference frame, and are given by

$$\mathbf{R}_x = \begin{pmatrix} \cos\alpha & \text{-}\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.8}$$

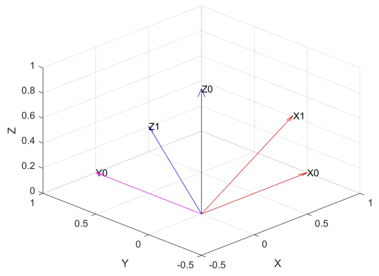
$$\mathbf{R}_y = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ \text{-}\sin\beta & 0 & \cos\beta \end{pmatrix} \tag{2.9}$$

$$\mathbf{R}_z = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & \text{-}\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{pmatrix} \tag{2.10}$$

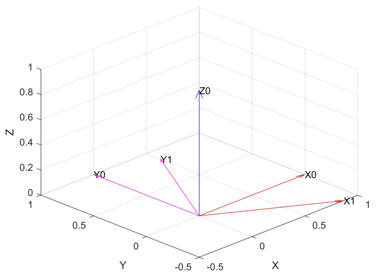
where α, β and γ are the angles rotated about the x, y and z axis, respectively. The rotation matrices describes the necessary rotation about an axis in space in order to align the axes of the reference frame with the corresponding axes of the body frame. This is illustrated in Figure 2.22.



(a) Rotation about the X axis.



(b) Rotation about the Y axis.



(c) Rotation about the Z axis.

Figure 2.22: Illustrations of a -30 degree rotation about the X, Y and Z axes.

2.2.1.3 Euler Angles

The rotation matrices are useful for calculating relative rotation between frames, but this representation provides a redundant description of frame orientations. The rotation is characterised by 9 elements in a 3×3 matrix, which are directly related by six constraints due to the orthogonality condition given by

$$\mathbf{R}^T\mathbf{R} = \mathbf{I}_3 \tag{2.11}$$

where $\mathbf{I}_3$ denotes the 3×3 identity matrix. The implication of Equation 2.11 is that only three independent parameters are needed to describe the relative orientation of frame in space. This representation is referred to as a *minimal representation*, and is denoted by

$$\mathbf{\Phi} = [\varphi \quad \vartheta \quad \psi]^T \tag{2.12}$$

where φ, ϑ and ψ are a set of angles that form a rotation sequence. In total there are 12 rotation sequences out of 27 possible combinations that guarantee that two successive rotations are not made about parallel axes. Each set represents a triplet of *Euler angles.*

YZX Angles

One of the most common triplet of Euler angles are the ZYX angles, also known as *Roll-Pitch-Yaw* angles or RPY angles. For this triplet, φ, ϑ and ψ in Equation 2.12 represents the rotation defined with respect to a fixed frame attached to the center of gravity of a rigid body. The rotation resulting from RPY angles can be obtained by

- Rotating the reference frame by the angle ψ about the x axis (yaw). The matrix $\mathbf{R}_\mathbf{x}(\psi)$ describes this rotation, which is defined in Equation 2.10.
- Rotating the reference frame by the angle ϑ about the y axis (pitch). The matrix $\mathbf{R}_\mathbf{y}(\vartheta)$ describes this rotation, which is defined in Equation 2.9.
- Rotating the reference frame by the angle φ about the z axis (roll). The matrix $\mathbf{R}_\mathbf{z}(\varphi)$ describes this rotation, which is defined in Equation 2.8.

The resulting frame orientation is obtained by composition of rotations with respect to the fixed frame, followed by computation via premultiplication of the the matrices of elementary rotation,

$$\mathbf{R}(\mathbf{\Phi}) = \mathbf{R}_z(\varphi)\mathbf{R}_y(\vartheta)\mathbf{R}_x(\psi)$$

where $\mathbf{R}(\mathbf{\Phi})$ can be written as

$$\mathbf{R}(\mathbf{\Phi}) = \begin{pmatrix} c_\varphi c_\vartheta & c_\varphi s_\vartheta s_\psi - s_\varphi c_\psi & c_\varphi s_\vartheta c_\psi - s_\varphi s_\psi \\ s_\varphi c_\vartheta & s_\varphi s_\vartheta s_\psi - c_\varphi c_\psi & s_\varphi s_\vartheta c_\psi - c_\varphi s_\psi \\ -s_\vartheta & c_\vartheta s_\psi & c_\vartheta c_\psi \end{pmatrix}. \tag{2.13}$$

The set of Euler angles can be obtained by comparing the inverse solution to a given rotation matrix

$$\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$$

and the expression of $\mathbf{R}(\boldsymbol{\Phi})$ in Equation 2.13. The resulting inverse solution is given in two separate ranges for ϑ. When $\vartheta \in (-\pi/2, \pi/2)$ the following solutions are true:

$$\begin{aligned} \varphi &= \text{Arctan2}(r_{21}, r_{11}) \\ \vartheta &= \text{Arctan2}(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}) \\ \psi &= \text{Arctan2}(r_{32}, r_{33}). \end{aligned} \tag{2.14}$$

When $\vartheta \in (\pi/2, 3\pi/2)$ the following solutions are true:

$$\begin{aligned} \varphi &= \text{Arctan2}(-r_{21}, -r_{11}) \\ \vartheta &= \text{Arctan2}(-r_{31}, -\sqrt{r_{32}^2 + r_{33}^2}) \\ \psi &= \text{Arctan2}(-r_{32}, -r_{33}). \end{aligned} \tag{2.15}$$

The solutions for Equation 2.14 and Equation 2.15 degenerate when $c_\vartheta = 0$ and the angle $\vartheta = \pi/2$. For this circumstance, it is only possible to determine the sum or difference of φ and ψ.

2.2.1.4 Quaternion Angles

A quaternion is a four element vector that provides a convenient mathematical notation for representing orientation and rotation of objects in three dimensions. Quaternions gives a simple way to encode this axis–angle representation in four elements, and can be used to apply the corresponding rotation to a position vector, representing a point relative to the origin in R^3. The Euler rotation is represented with a triplet of Euler angles, that have an ambiguity in the specification of the Euler angles, meaning that a (yaw, pitch, roll) convention does not interoperate with an algorithm that assumes (roll, pitch, yaw). Quaternion on the other hand represents the rotations as a single rotation which is faster in terms of computation time and avoids *Gimbal lock* [46]

A quaternion q is generally represented in the form

$$q = s + xi + yj + zk$$

where s, x, y, z are real numbers and i, j, k are the imagery fundamental *units quaternion*. An orientation and rotation defined with a unit rotation axis $\mathbf{u} =$

$[u_x, u_y, u_z]$ and a scalar θ can than be represented with unit quaternions ($s^2 + x^2 + y^2 + z^2 = 1$) as

$$\mathbf{q} = e^{\frac{\theta}{2}(u_x i, u_y j, u_z k)} = \cos\frac{\theta}{2} + (u_x i, u_y j, u_z k)\sin\frac{\theta}{2} = [q_w, q_x, q_y, q_z].$$

2.2.1.5 Translation

In the same way that rotation matrices describes the relative orientation between two frames, translation vectors describes the relative displacement between two frames in space. The translation vector $\mathbf{O}_1^0$ in Figure 2.23 illustrates the translation between the origin of frame 0 and and frame 1, and is given by

$$\mathbf{O}_1^0 = \begin{pmatrix} x \\ y \\ z \end{pmatrix}.$$

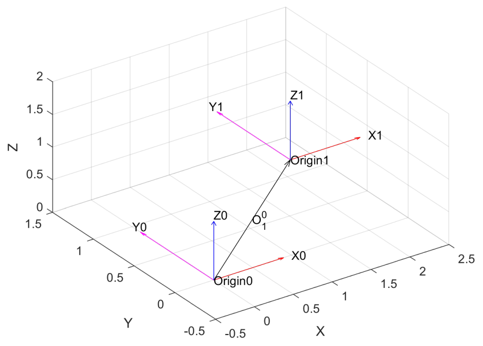


Figure 2.23: Translation vector $\mathbf{O}_1^0$ illustrating the relative displacement between frame 0 and frame 1.

2.2.1.6 Homogeneous Transformation

The rotation matrix, translation vector, and a $[001]^T$ vector can be combined in a single matrix, in order to express the relative orientation and displacement between two frames. The $[001]^T$ vector is included to simplify matrix operations, and constitutes the bottom row of the combined 4×4 matrix called the *homogeneous transformation matrix*. From Figure 2.24, let P be an arbitrary point in space, and $\mathbf{p^0}$ and $\mathbf{p^1}$ be the vector from the origin of frame 0 and frame 1, respectively. The vector $\mathbf{O_1^0}$ is the vector representing the translation between the frames, and $\mathbf{R}_1^0$ is the rotation matrix to frame 1 relative to frame 0.

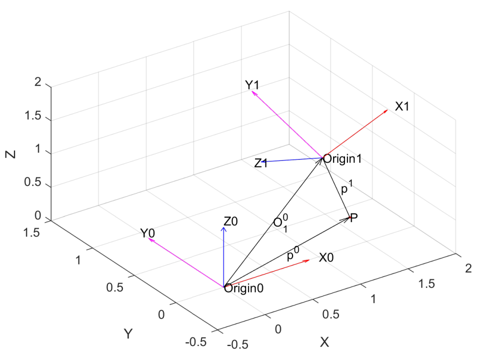


Figure 2.24: Representation of a point P in two different coordinate frames.

From this, the location of point P relative to frame 0 can be expressed as

$$\mathbf{p}^0 = \mathbf{O}_1^0 + \mathbf{R}_1^0\mathbf{p}^1 \tag{2.16}$$

and represents the *coordinate transformation* of a bound vector between two frames. The inverse transformation can be calculated by premultiplicating $\mathbf{R}_1^{0T}$ on both sides of Equation 2.16, resulting in

$$\mathbf{p}^1 = -\mathbf{R}_0^1\mathbf{O}_1^0 + \mathbf{R}_0^1\mathbf{p}^\mathbf{0}.$$

Introducing

$$\hat{\mathbf{p}} = \begin{pmatrix} \mathbf{p} \\ 1 \end{pmatrix}$$

where $\mathbf{p}$ is a point in a frame, a compact representation is achieved between the relationship the coordinates of the same point in two separate frames. The coordinate transformation can be expressed by the 4×4 matrix

$$\mathbf{T}_1^0 = \begin{pmatrix} \mathbf{R}_1^0 & \mathbf{O}_1^0 \\ \mathbf{0}^T & 1 \end{pmatrix}$$

which is the homogeneous transformation matrix.

2.2.1.7 Forward Kinematics

Forward kinematics relates a set of given joint angles on a robotic manipulator to the end position and orientation of the end-effector frame, relative to the base frame. Given the joint angles and the DH parameters for an open chain manipulator consisting of n number of links or rigid bodies, the position of the end-effector can be determinated by the successive homogeneous transformation from the base frame to the end-effector frame as follows

$$\mathbf{T}_n^0 = \prod_{i=1}^{n} \mathbf{T}_i^{i-1}(\theta_i). \tag{2.17}$$

Equation 2.17 can be written as

$$\mathbf{T}_n^0(\mathbf{q}) = \begin{pmatrix} \mathbf{n}_n^0(\mathbf{q}) & \mathbf{s}_n^0(\mathbf{q}) & \mathbf{a}_n^0(\mathbf{q}) & \mathbf{p}_n^0(\mathbf{q}) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where $\mathbf{q}$ is a $(n \times 1)$ vector containing the joint angles, and $\mathbf{n_n}, \mathbf{s}_n, \mathbf{a_n}$ are the unit vectors of a frame attached to the end-effector. $\mathbf{p}_n$ is the position vector of the end-effector frame, relative to the base frame. Figure 2.25 illustrates the issue of forward kinematics, where the joint angles are known, but the position of the end-effector is unknown. The vector $\mathbf{p_2^0}$ represents the displacement from the base frame to the end-effector frame, and is extracted from the homogeneous transformation from the base frame to the end-effector frame.

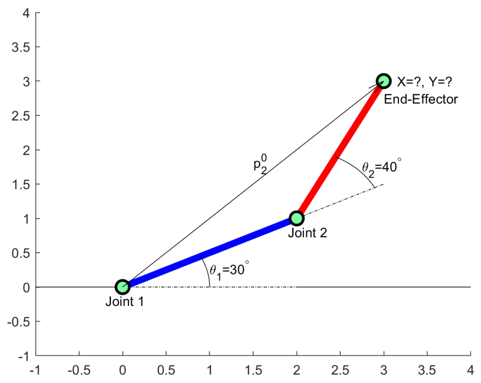


Figure 2.25: 2D illustration of the forward kinematics issue. The joint angles are know, but the position of the end-effector is unknown. The vector $\mathbf{p_2^0}$ represent the position of the end-effector relative to the base frame.

2.2.1.8 Inverse Kinematics

Inverse kinematics is the process of transforming a given position of the end-effector in space to the necessary values in the joint space to achieve the position. Especially in terms of bin picking, the inverse kinematics problem is generally more relevant than the forward kinematics problem described in Section 2.2.1.7, as the goal is to pick up items at certain known locations in space. For the forward kinematics, each set of joint values corresponds to a single position of the end-effector frame. However, for the inverse kinematics problem there might be multiple or an infinite number of joint values for a given end effector frame. Thus, the inverse kinematics problem is much more complex, as the equations to solve are generally nonlinear. Figure 2.26 illustrates the inverse kinematics problem, where the desired end-effector frame is known, while the joint angles are unknown.

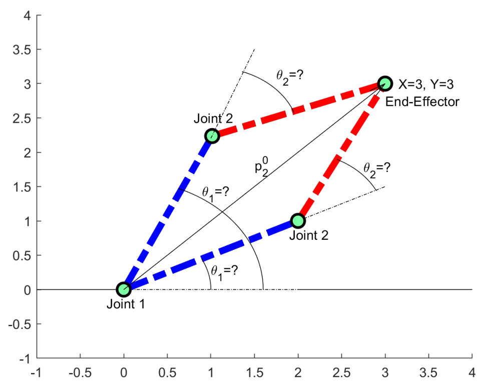


Figure 2.26: 2D illustration of the inverse kinematics issue. The relative position of the end-effector frame and the base frame $\mathbf{p_2^0}$ is given, but the corresponding joint values are unknown. Two possible configurations may result in the desired end-effector position.

2.2.1.9 Trajectory Planning

Path and trajectory planning is an important topic when using manipulators, as it is the process of generating geometric and motion paths between an initial state and a goal state.

In the simplest form a path planner generates a path between two states without any constrains and time concerns. However, for more complex application were the workspace is restricted with static and somethings dynamic obstacles, collision avoiding is needed. Furthermore, for the case of bin picking, the motion also needs to take into account the external forces that may occur between the gripper and the grasped object.

The objective of the trajectory planning is to generate the reference inputs for the manipulator control system in order to obtain desired movement. A trajectory planning algorithm uses a geometric path, and the kinematic and dynamic constraints of the manipulator in order to generate trajectory of the joints, expressed as a sequence of values of position, velocity and acceleration [47]. An extensive amount of research in robotics has focused on path and trajectory planning the last 30 year, resulting numerous different approaches [48].

2.2.2 ROS: Robotic Operation System

The *Robotic Operation System*, or *ROS*, was initially released in 2007, and is an open-source, meta operating system for a wide variety of robotic platforms. The software runs on Unix-based platforms and provides services that include hardware abstraction, low level device control, implementation of commonly used functionality, message passing between processes, and package management. Libraries are provided though ROS for obtaining, building, writing, and running code across multiple computers. The main goal of ROS is to support reusage of code in robotics research and development, enabling sharing and collaboration. This section will cover some of the fundamental concepts in the ROS structure, such as *nodes*, *messages*, *topics*, and *services*.

2.2.2.1 Nodes

A ROS node is an executable file in a ROS package that performs computations, and can either be written in Python or C++ [49]. Due to the software architecture of ROS, it is desirable to create individual modules, or nodes, that operate specific aspects of a system. A robot control system will therefore usually be composed of several individual nodes that are run simultaneously, communicating with each other in order to achieve the desired behaviour. Communication between nodes is most often accomplished by publishing messages that are sent to topics, and as services.

2.2.2.2 Messages

Messages are specific data structures that are used as communication between nodes. Message descriptions are stored in `.msg` files in a sub directory of a ROS package, where variable names and data types are defined. The structure of a `.msg` file can be observed in Figure 2.27, where two variables x and y are defined as an integer and a Boolean type, respectively. ROS supports standard primitive types (integer, floating point, Boolean), and a single message might publish several variables with different types [50].

```
datatype variable1
datatype variable2
```

For example

```
int x
bool y
```

Figure 2.27: The structure of a `.msg` file.

2.2.2.3 Topics

Topics are channels where nodes either publish messages, or subscribe to publications from separate nodes. A single node might subscribe or publish to several topics, which simplifies the communication structure of the system. This can be seen in Figure 2.28, where Node A subscribes to several other nodes. Topics are often used in situations where continuous data streaming is preferable, such as sensor data and actuator control data. For situations where nodes shall perform remote procedure calls, *services* might be more suited instead of topics.

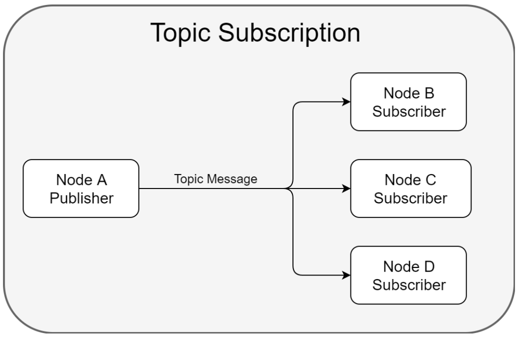


Figure 2.28: ROS nodes communicating over a topic. Node B, Node C and Node D subscribes to to topic message that is published from Node A.

2.2.2.4 Services

Services are another way nodes can communicate with each other in ROS, and offers a one-to-one request / response interaction between the nodes [51]. An operating ROS node offers a service under a sting name, and a client calls the service by sending a request message before waiting for a response. The response might be a computation or configuration of hardware and software. The call and the response messages are predefined in `.srv` files, that builds directly upon the ROS `.msg` format. The `.srv` file consists of a request and a response message type, that is separated by '----', as can be seen in Figure 2.29. Figure 2.30 illustrates the general interaction between two nodes during a service call.

```
int request
----
int response
```

Figure 2.29: The structure of a `.srv` file.

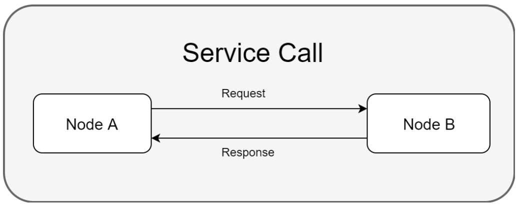


Figure 2.30: General interaction during a service call between Node A and Node B. Node A sends a request, and waits until it receives a response.

2.2.3 Grasping

Obtaining a robust and secure grasp on objects when performing bin picking, is critical to ensure a reliable system. Failure to locate a suitable grasp may result in a halt in the operation, which in turn reduced the productivity and profitability of the system. Damage to the objects may also occur as a result of mid air grasp failure. Several factors influence the grasp success rate, such as the geometry, size, mass, center of gravity, surface texture, as well as the end-effectors ability to pick up novel objects.

A large number of autonomous bin pick solution utilises model-based grasp planning which relies on object recognition and pose estimation [52], [53]. More recently data-driven approaches have become more popular [54], [55], [56] that uses neural networks for object detection followed by object pose estimation. These networks typically require an extensive amount of training data as well as a 3D model of the objects. In terms of bin picking, the grasp solution needs to be scalable for new products, since the range of observed object categories in warehouse settings is often large and dynamic. Some recent work conducted on novel grasping is presented in the following list:

- Lerrel Pinto and Abhinav Gupta [57] utilises a self-supervising algorithm that learns to predict grasp locations via trial and error, which proves to efficiently generalise to novel objects. After 700 robot hours and 50 000 grasping attempts, they achieved 73 % success rate on seen objects and 66 % on novel objects.

- Andy Zeng and Shuran Song et al. [58] used an affordance-based grasping algorithm which is model-free and agnostic to object identity, improving generalisation to novel objects without re-training, thus demonstrating great scalability.

- A research team at Google demonstrated that a learning-based hand-eye coordination approach for robotic grasping from 2D images was able to achieve satisfying results. They collected 900.000 grasps over a two months period using 14 manipulators in parallel and trained a deep neural network on the obtained grasp samples [59].

This section will cover one of the most promising approaches to suction grasping that might be applied for novel objects related to bin picking.

2.2.3.1 Dexterity Network Project

The *Dexterity Network* (Dex Net) project was first presented in 2017, and is one of several deep learning based grasp algorithms that has been developed within the recent years. The first iteration of the dexterity network project, Dex Net 1.0, introduced a dataset containing over 10 000 different 3D objects and over 2.5 million parallel jaw grasps for the objects. Each grasp includes data related to grasp parameters, uncertainty in grasp, contact model and quality matrix.

Multi-View Convolutional Neural Networks (MV-CNNs) was used for 3D object classification and trained on the introduced dataset. Dex Net 2.0 was introduced later in 2017, and was an improved version of the Dex Net 1.0 version that reduced the data collecting time by generating a synthetic training dataset from 6.7 million point clouds, grasps, and analytic grasp metrics generated from Dex Net 1.0. A third iteration was released in 2017, and is covered in the following section.

2.2.3.2 Dex Net 3.0

Dex Net 3.0 was released in the beginning of 2018 and is a suction grasping based dataset used to train deep neural networks associated to bin picking. Suction based end-effectors has several advantages compared to parallel jaw grippers, such as a single point of contact and increased agility. Suction based grasps has proven to generate good results in contexts of random object bin picking, and is the preferred end-effector used in Amazon Picking Challenge. The Dex-Net 3.0 dataset was designed to train a deep neural network that takes into account these external wrenches, material properties, and robustness to perturbations in the end-effector pose, in order to evaluate sampled grasps in a depth image. The dataset contains over 2.8 million points clouds with suction grasps and robustness labels, and 1 500 3D object models with a total of 350 000 labeled grasp poses.

Following [60], a robot equipped with an RGB-D camera examines a depth image $\mathbf{y}$, where the goal is to locate most robust suction grasp $\mathbf{u}$, so that the robot can lift and relocate the desired object. The grasps are parameterized by a target point $\mathbf{p} \in \mathbb{R}^3$ with a corresponding approach direction $\mathbf{v} \in \mathcal{S}^2$. Success is represented as a binary grasp quality function, S, where $S = 1$ represents a successful transport of the object, and $S = 0$ otherwise. However, the robot may not be capable of predicting the success of suction grasp directly from $\mathbf{y}$, due to several factors such as the latent state of the object $\mathbf{x}$. Valuable data regarding object geometry, inertial, material properties, and relative pose between object and camera may be inadequate due to occlusion and noise in the depth image. External wrenches from gravity and the surroundings on the object also represents a source of uncertainty. The *robustness* of a grasp, $\mathbf{u}$, from a given depth image $\mathbf{y}$, with respect to an environment p is the probability of success Q of grasping under uncertainty in sensing, control, and disturbing wrenches

$$Q(\mathbf{u}, \mathbf{y}) = \mathbb{P}(S \mid \mathbf{u}, \mathbf{y}).$$

Objective

The best suited grasp location implies finding the grasp that maximises the robustness given a depth image

$$\pi^*(\mathbf{y}) = \mathrm{argmax}_{\mathbf{u} \in \mathcal{C}} Q(\mathbf{u}, \mathbf{y})$$

where Q is the robustness function and $\mathcal{C}$ specifies a set of constrains on the set of available grasps, and includes kinematic feasibility or collisions. In practice, the robot does not have an explicit representation of the robustness function Q, however it has the ability to sample from the stochastic environment. As a hand coded database of possible object states is challenging, an approximation of π^* is calculated by training a Grasp Quality Convolutional Neural Network (*GQ CNN*) on samples containing success labels, point clouds, and suction grasps from the model by minimising the cross-entropy loss $\mathcal{L}$:

$$\min_{\theta\in\Theta}\sum_{i=1}^{N}\mathcal{L}(S_i, Q_\theta(\mathbf{u}_i, \mathbf{y}_i)).$$

Learning Q instead of directly learning π^* increases the flexibility of the model and allows for enforcement of task-specific constraints without the need to retrain the learned model. Given a pre trained GQ-CNN with weights $\hat{\theta}$, deep robust grasping policy can be executed by

$$\pi_\theta(\mathbf{y}) = \text{argmax}_{\mathbf{u}\in\mathcal{C}} Q_{\hat{\theta}}(\mathbf{u}, \mathbf{y}).$$

Contact Model

Dex Net 3.0 presents a contact model between the surface of the object and the suction cup, in order to estimate the deformation energy required to maintain a seal. The suction cup material is modelled as a quasi-static spring model, and the contact wrenches is calculated through the perimeter of the suction cup and the object surface. The success metric first evaluates whether or not a seal can be obtained between the perimeter of the suction cup and the surface, and secondly if a given seal can withstand external wrenches on the object. The upward lifting force is created as a result of a difference in air pressure between the inside of the suction cup and the outside surrounding. If a gap between the suction cup perimeter and the object surface occurs, the lift force will be reduced due to a reduction of the difference in air pressure. It is therefore critical to locate a planar surface on the object, and to examine if the location can handle wrench during transportation.

In order to determine the possibility of obtaining seal, the suction cup is represented as a spring system $\mathcal{C}$ parameterized by (n, r, h), where n is the number of perimeter components, r is the radius of the cup, and h is the height of the cup. An illustration of this spring system can be seen in Figure 2.32 and in Figure 2.31. The spring system $\mathcal{C}$ is composed of a structural spring that represents the physical structure of the spring, and a flexion spring that represents the resistance of bending along the objects surface. During evaluation whether seal formation is possible or not, a configuration of $\mathcal{C}$ is calculated and projected onto the surface of an object in the scene. The target's surface is modelled as a triangular mesh $\mathcal{M}$, and the seal is evaluated under quasi-static conditions as a proxy for the dynamic possibility of seal formation. The system classifies a complete seal between $\mathcal{C}$ and $\mathcal{M}$ if all of the perimeter springs of $\mathcal{C}$ have fully connection with the surface of $\mathcal{M}$.

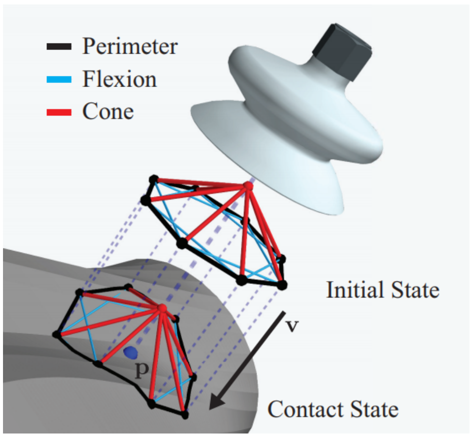


Figure 2.31: The suction cup is modelled as a quasi-static spring, where a seal formation is categorised as feasible if the energy required to maintain the seal in each spring is lower than a given threshold. [60]

External Wrench Forces

Several types of wrenches can be experienced by the suction cup during lifting, so a wrench analysis is introduced to determinate if the suction cup can withstand external wrenches during the pick. The following forces defines the wrench map $\mathcal{G}$, which also are illustrated in Figure 2.32:

- **Vacuum Force (V):** The resulting pulling force from the difference in air pressure.
- **Actuated Normal Force (f_z):** The force that the suction cup material applies by pressing into the object along the z axis.
- **Frictional Force ($f_f = (f_x, f_y)$):** The force in the tangent plane of contact, as a result of the normal force between the suction cup and the object, $f_N = f_z + V$.
- **Torsional Friction (τ_z):** The torque experienced by the frictional forces in the perimeter of contact.
- **Elastic Restoring Torque ($\tau_e = (\tau_x, \tau_y)$):** The torque about axes in the contact tangent plane resulting from elastic restoring forces that are present in the suction cup what are pushing on the object along the boundary of the contact ring.

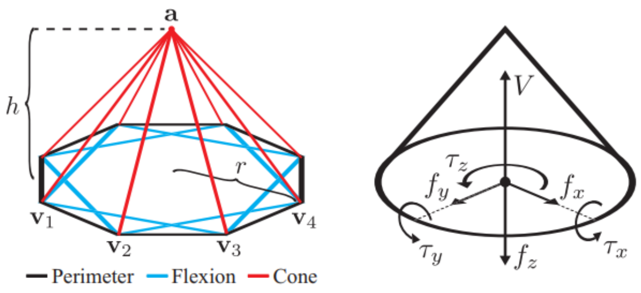


Figure 2.32: The left figure is an illustration of the suction cup modelled as a quasi-static spring, which is used for evaluation for seal formation. The right figure illustrates the wrench forces experienced on the modelled suction cup. [60]

Certain constrains are present, which limits the magnitude of the contact wrenches. These constrictions are due to the friction limit at the surface, limits on the elastic behaviour of the suction cup material, and limits in regards to the vacuum force. The final evaluation of the wrench analysis implies evaluation of the robustness of candidate suction grasps. The robust wrench resistance metric $\mathbf{W}$ for $\mathbf{u}$ and $\mathbf{x}$ is defined as

$$\lambda(\mathbf{u}, \mathbf{x}) = \mathbb{P}(W \mid \mathbf{u}, \mathbf{x})$$

and represents the probability of a successful grasp under perturbations in object pose, friction, gripper pose, and disturbing wrenches.

2.2.4 Grippers

The end-effector is located at the end of a robotic arm, and is a device that allows the robot to interact with the environment. Depending on the operation requirement, the end-effector is selected in order to best serve the task at hand. End-effectors for certain tasks include cameras, cutting tools, welding guns, magnets etc. For the practical case of bin picking, the end-effector is a *gripper* that allows the robot to pick up and move objects. There are a large variety of grippers available on the market, and the different grippers all have certain advantages and dis-advantages related to certain operations.

Grippers are generally divided into four categories based on the different psychical principles used for grasping [61]:

- **Impactive**
  Mechanical grippers where prehension is achieved by impactive forces that work against the surface of the object to be acquired.

- **Ingressive**
  Grippers that achieve prehension by penetrating the objects surface, and can include pins, needles, and hooks.

- **Astrictive** Grippers that achieve prehension by suction forces to the surface of the object. The suction force can be produced through vacuum, magnetism or electrostatic charge displacement.
- **Contigutive**
  Grippers that achieve prehension through direct contact where adhesion takes place on the surface of the object, such as glue, surface tension, and freezing.

The following section will cover one of the most relevant and most used gripper for grasping related to bin picking.

2.2.4.1 Vacuum Gripper

Vacuum grippers are a part of the astrictive grasping category, and are extensively used throughout the packing industry [62]. This is mainly due to its ease of implementation, low cost and gripping strength, which is achieved by applying a negative pneumatic pressure to one or several suction cups. An analysis from the 2016 Amazon Picking Challenge found that 62% of the competing teams that scored better than zero points relied on some form of vacuum suction for picking, while five teams relied on force closure and/or friction [63]. The Amazon Picking Challenge winners in 2016 and 2017 both used a vacuum based end-effectors.

Venturi Ejector

Vacuum can be achieved several ways, but a commonly used method is through a Venturi vacuum ejector. This method generates vacuum though a series of steps, and is illustrated in Figure 2.33:

- Compressed air is led from the supply source to the ejector.
- A constriction of the cross section in the nozzle results in increased flow velocity of the air.
- At the exit of the nozzle the cross section suddenly increases, resulting in a vacuum as the air expands.
- The compressed air along with the vacuumed air both leave through an outlet port.

The pressure drop created the vacuum is given by

$$p_1 - p_2 = \frac{\rho}{2}(v_2^2 - v_1^2)$$

where v_1 is the slower initial velocity, v_2 is the faster velocity at the narrow section, and ρ is the density of the air (or other used fluids). Venturi ejectors are easy to mount, and no additional equipment is necessary. However, high operational costs related to compressed air consumption may be present, and the operation produces loud noises.

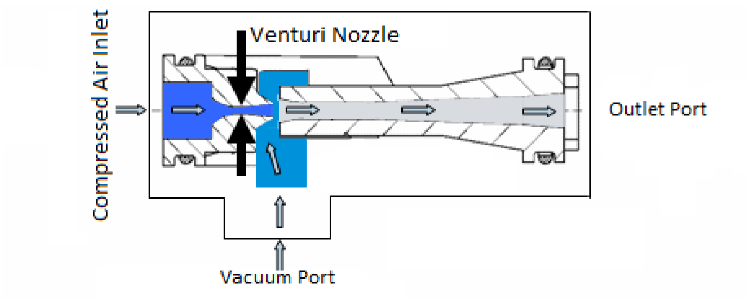


Figure 2.33: Illustration of the concept of Venturi vacuum generators. Compressed air is directed to the inlet, where the air is accelerated through a nozzle, before the the cross section suddenly increases. This generates a vacuum, as the pressure suddenly decreases. [64]

Vertical Lifting Force

At initial contact between the suction cup and the object surface, the suction cup is compressed against the surface of an object to obtain a seal. This will prevent leakage and thereby maintain suction force. For a typical circular suction cup, the lifting force on an object is given by

$$F = (P_a - P_v)A$$

where A is the area of contact between the suction cup and the object, P_a is the atmospheric pressure, and P_v is the applied vacuum pressure [65]. When vacuum grippers are used on smooth surfaces, the power consumption of the gripper head is minimal, where the largest contributing factor to reduction in efficiency is the air leakage rate. The rate is dependant of the degree porosity of the object material at the contact surface. The theoretical suction force F_s acting perpendicular to the object surface is given by

$$F_s = \Delta PA$$

where ΔP is the generated vacuum, and A is the area of the suction surface. The theoretical holding force F_h of the suction cup can be calculated by

$$F_h = m(g + a)S$$

where m is the mass of the object, g is the gravitational acceleration, and a is the acceleration experienced from the robotic movement [66]. S is a safety factor, that accounts for other external factors that may influence the actual performance. The safety factor is determined by a *NS-EN* standard [67], and is usually a factor of 2.

The theoretical vertical lifting capacity with a suction cup with diameter D and safety factor S can be expressed as

$$F_s = F_h$$

$$\Delta PA = m(g + a)S$$

$$m = \frac{\Delta PA}{(g + a)S}$$

$$m = \frac{\Delta P \pi D^2}{(g + a)S4} \tag{2.18}$$

and can be used to estimate the necessary suction cup diameter for a given weight.

Chapter 3

Method

This chapter will present our work and findings while pursuing an automated bin picking system, and will cover some of our discoveries and choices that we made while conducting our work. A simplified pipeline for the proposed system can be seen in Figure 3.1, where the *Start* state initialises the hardware and software of the system, before an RGB and a depth image is acquired in the *Image Acquisition* state. The RGB image is sent to the object detection module, where a user defined object is located using a deep neural network. The location of the object is used to crop out the corresponding area in the depth image, which is used to locate the best suited grasping location given by another deep neural network in the *Grasp Planning* state. The grasping location is sent to the *Robot Manipulation* state, where a robot performs the picking operation. The series of actions can be repeated for a new object after completing the pick, or the system can exit the operation. We substantiate this pipeline based on a set of system requirements that we define:

- **Realistic Setting**
  The goal of the practical work was to create an autonomous bin picking system compliant with AutoStore's ASRS solution. We therefore assumed that the inventory bin was placed at a static location.

- **Object Defined Picks**
  The system should be able to pick specific products from the bin, to replicate the product ID in an order. The solution should therefore included an object detection module for this purpose. In addition, the system should be able to pick novel undefined objects as well.

- **Grasping**
  The system should be able to handle a large variety of product geometries, to replicate the large selection of products that may be present in a realistic warehouse setting. The scope of this solution was however limited to objects up to 1 kg, and use of a suction gripper.

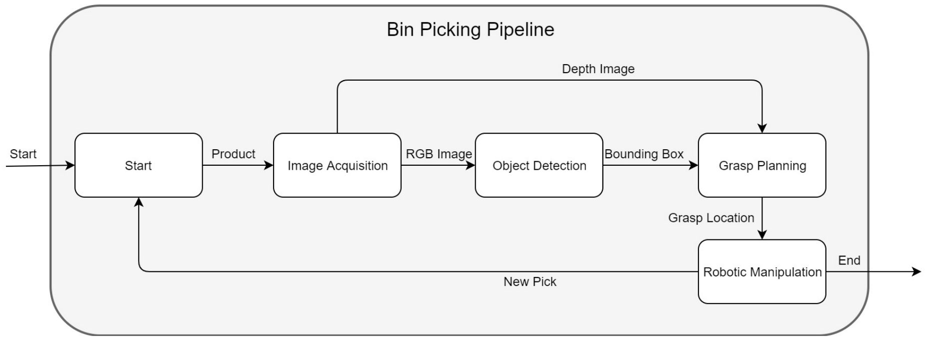


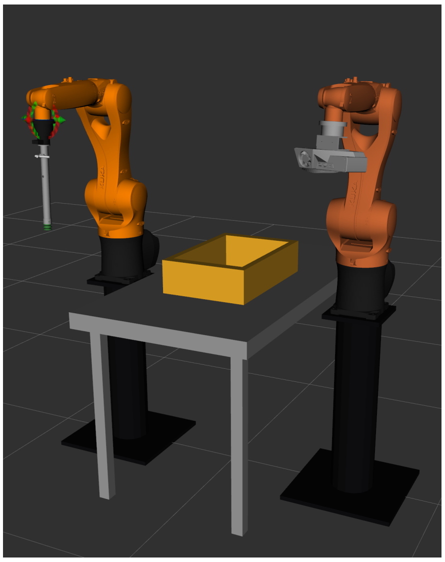
Figure 3.1: Pipeline of our proposed bin picking system. A product is specified in the *Start* state, which is used to obtain a bounding box of the object in the *Object Detection* state. The *Object Detection* and *Grasp Planning* states uses an RGB image and a depth image from the *Image Acquisition* state, and a grasping location is proposed and sent to the *Robotic Manipulation* state where the final picking action is performed.

3.1 Physical Setup

The practical work was conducted at the small robot cell at the *Department of Mechanical and Industrial Engineering* laboratory, which consisted of the following hardware:

- Two KUKA Agilus KR 6 R900 sixx, six axis robot manipulators
- Two KUKA KR C4 compact robotic controllers
- Four given objects used to train our object detection network
- Schunck suction cup test set
- Custom made gripper module
- Zivid 3D camera
- Master computer

The physical setup of the robot cell can be seen in Figure 3.2, along with a simulated model in Moveit!. The left robot will be referred to as *Agilus1*, and was equipped with a custom made vacuum gripper. The right robot will be referred to as *Agilus2*, and was equipped with a Zivid 3D camera for computer vision. The camera did not necessarily have to be attached to a robot since it only was used at a stationary position. However, due to the range of the camera and placement of the bin with respect to Agilus1, the camera had to be moved in order to avoid collision between the camera and gripper.


(a) Simulated robot cell in moveit!.

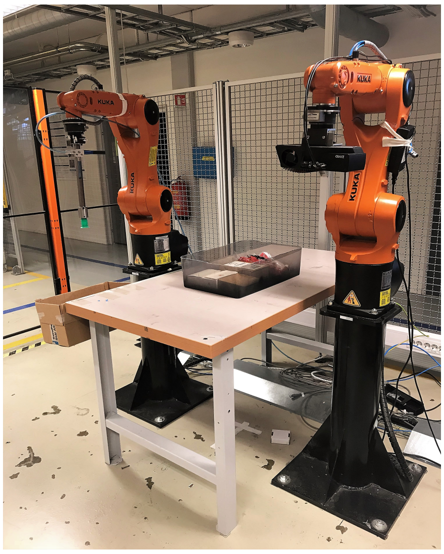
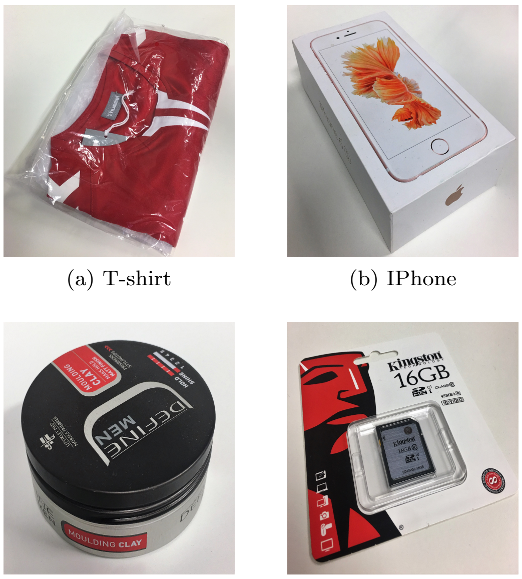

(b) Robot cell in the real world.

Figure 3.2: Illustration of the simulated and physical robot cell.

3.1.1 The Objects Used For Object Detection

The objects that we wanted to detect and pick in our system can be seen in Figure 3.3, and included a t-shirt, an IPhone, a hair wax box and a SD memory card. The objects have a large variety in terms of form, material, physical properties, mass, and were chosen to represent the large variety of product attributes that might be present in an actual warehouse.



(a) T-shirt (b) IPhone (c) Hair wax (d) SD-card

Figure 3.3: The four test objects used to train our object detection module in this thesis. The objects represents the large variety of product attributes in an industrial warehouse.

3.1.2 Gripper Module and Suction Cup

The gripper module and the suction cup serves as the connection between the robot and the environment, and was custom made for our setup. The gripper module is based the theory presented in Section 2.2.4.1, and is a single point of contact suction gripper. In order to select a suitable suction cup for our 1 kg system requirement, the maximum working capacity of the vacuum ejector had to be found. A vacuum gauge was therefore mounted between the vacuum ejector and a suction cup placed on a suitable surface for maximum seal. The maximum vacuum capacity of the ejector was estimated to -0.87 mPA, when the input pressure was measured to 4.5 bar. Figure 3.4 illustrates the theoretical lifting capacity versus cup diameter derived from Equation 2.18, with a safety factor $S = 2$ and robot acceleration $a = 3\text{m/s}^2$, and a constant vacuum $\Delta P = 0.87\text{mPA}$.

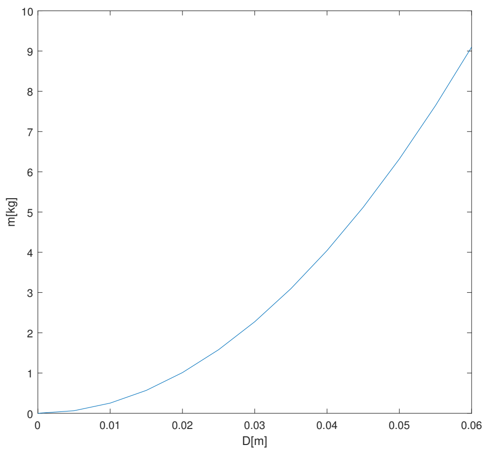
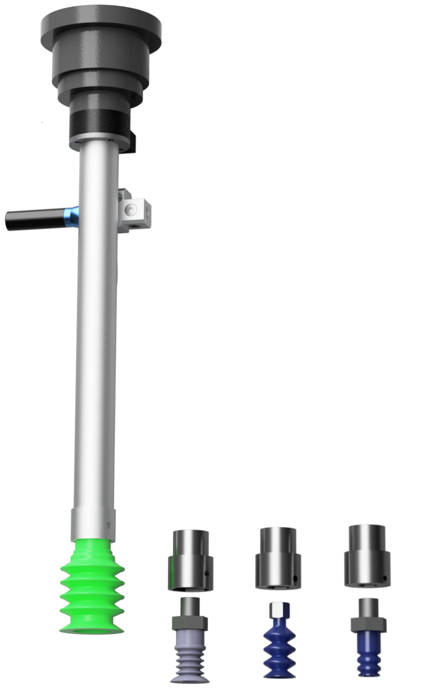


Figure 3.4: The theoretical lifting capacity m[kg] with a suction cup with diameter D[m], on a surface with $\mu = 1$, S = 2 and maximum system acceleration $a = 3\text{m/s}^2$. The vacuum ΔP is constant at -0.87 mPa.

From 3.4 we see that our 1 kg requirement is satisfied with a 2 cm suction cup diameter. A suction cup test set was therefore ordered from *Schunk*, where the majority of the suction cup diameters were in the range from 1 to 3 cm.

The gripper module consisted of three individual parts, as can be seen in Figure 3.5, and was designed in *Autodesk Fusion 360*. The gripper module was designed with a 30 cm long slender shaft with a mount for the vacuum ejector at the top, in order to pick objects next to the bin walls without collision. As a consequence of this additional length, the vertical approach angel was heavily restricted due to the walls. This could potentially be solved by a more complex gripper that has a revolution joint at the tip [68], [69]. The ordered suction cups used in our setup have a mounting diameter of either 1.5 cm, 1.7 cm, 2.4 cm or 2.7 cm. The gripper module was therefore made with a changeable mounting system, so that the different suction cups easily could be changed.


(a) The final CAD model of the gripper and suction cups.

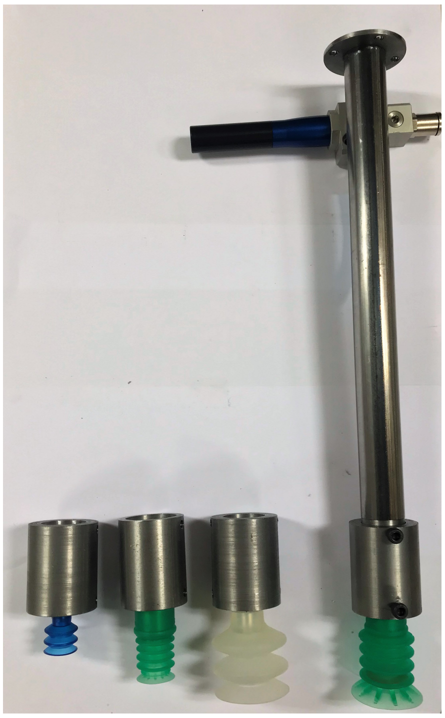

(b) The manufactured gripper, along with suction cups.

Figure 3.5: Illustration of the CAD design of the gripper and the manufactured product. The different suctions cups are easy to change with the designed mounting system.

3.1.3 KUKA Agilus KR 6 R900 SIXX

Two KUKA Agilus KR 6 R900 robots were used in the practical work of this thesis, where one robot was equipped with the Zivid camera, and the other robot was equipped with the vacuum gripper. This robot model is a 6 degree of freedom manipulator, which provides high agility in terms of pose and orientation of the end-effector. This is naturally an important feature for a robot that is intended to perform random bin picking, where objects have arbitrary locations and orientations. The technical data for the robot model can be seen in Table 3.1, along with an illustration of the robot and its axes of rotation in Figure 3.6. The robots were connected to two KUKA KR C4 controllers that handles input and output signals, robot control, PLC communication, and safety drivers.

KUKA Agilus KR 6 R900 SIXX	
Maximum Reach	901.5 mm
Maximum Playload	6 kg
Pose Repeatability (ISO 9283) [a]	±0.03 mm
Number of Axis	6
Footprint	320 × 320 mm
Weight	52 kg
Motion Range: A1	±170°
Motion Range: A2	−190°/45°
Motion Range: A3	−120°/156°
Motion Range: A4	±185°
Motion Range: A5	±120°
Motion Range: A6	±350°

Table 3.1: Technical data for the KUKA agilus KR6 R900 SIXX manipulator. [70]

[a] https://www.iso.org/obp/ui/#iso:std:iso:9283:ed-2:v1:en

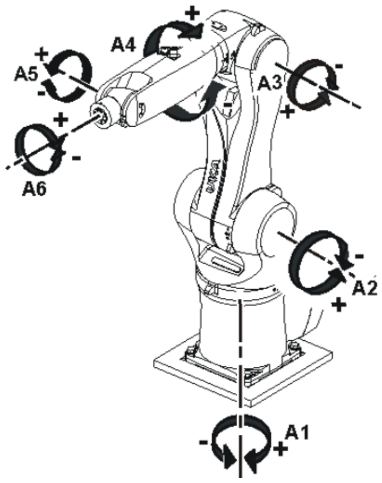


Figure 3.6: Illustration of directional rotation of the robot axes. [71]

3.1.4 Zivid 3D Camera

The Zivid camera is a high performance 3D camera for robotic and industrial applications, and was used in our system. The camera is capable of capturing colored full HD resolution 3D images with a 0.1mm depth accuracy at 0.6 meters, and operates on the principle of structured light, as covered in Section 2.1.2. The camera has a real time feature at 10 Hz, which allows robots and machines to rapidly see and understand the environment in three dimensions. The Zivid camera also has the ability to capture difficult parts such as shiny metallic objects and dark absorbing plastics, due to HDR processing (explained in Section 2.1.3.1). The camera was delivered with a software for capturing and viewing acquired point clouds, as well as live streaming. The camera supported APIs for C++ and .NET, and can be run on Windows and Linux operation systems.

The camera was chosen for our system, due to its high precision in depth measurements and its ability to acquire depth images of shiny or absorbing materials. In our system the camera was mounted the Agilus2 robot, and moved into position for each iteration where we needed to acquire RGB images and depth images. An illustration of the camera can be seen in Figure 3.7.

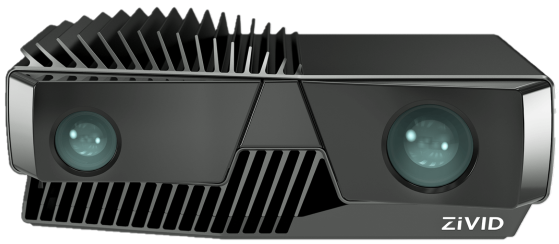


Figure 3.7: The Zivid 3D camera was used in the practical work of this thesis, due to its high precision in depth measurements. [72]

3.1.5 Eye-in-Hand Camera Calibration

Eye-in-hand calibration was applied in order to determine the transformation between the initial last link of the kinematic chain of Agilus2, and the mounted Zivid camera. The calibration allowed us to accurately determine the location of the camera relative to the robot hand frame, and also to transfer points in the captured image to the world frame. A precise calibration was necessary, as the final suction location was dependant of this calibration. Several algorithms and software programs are provided for the purpose of eye-in-hand calibration, such as OpenCV, MATLAB and ROS i.e. For our work, we used software provided by the Zivid company, where the outputs of the calibration were the intrinsic and extrinsic parameters of the camera, along with the camera distortion coefficients. Following the proposed procedure from Section 2.1.4, the calibration was conducted using a checkerboard of size 9×5 with a 20 mm pattern as the static calibration object. A total of 17 images were taken, and the corresponding transformations from the robot base to the hand were recorded. We made certain to have a large variation in the position and orientation in which the images were taken, so that the calibration was representative for entire work space.

Calibration

In order to perform eye-in-hand calibration with the proposed procedure from Section 2.1.4, the following equation was solved:

$$\mathbf{AX} = \mathbf{XB}. \tag{3.1}$$

A dataset with transformation **A** and **B** was gathered by capturing images of the stationary checkerboard, along with its corresponding base to hand transformation from different positions. Equation 3.1 was then solved by utilising Zivid's implementation of the calibration process [73]. Figure 3.8 illustrates different positions of the camera used for the calibration.

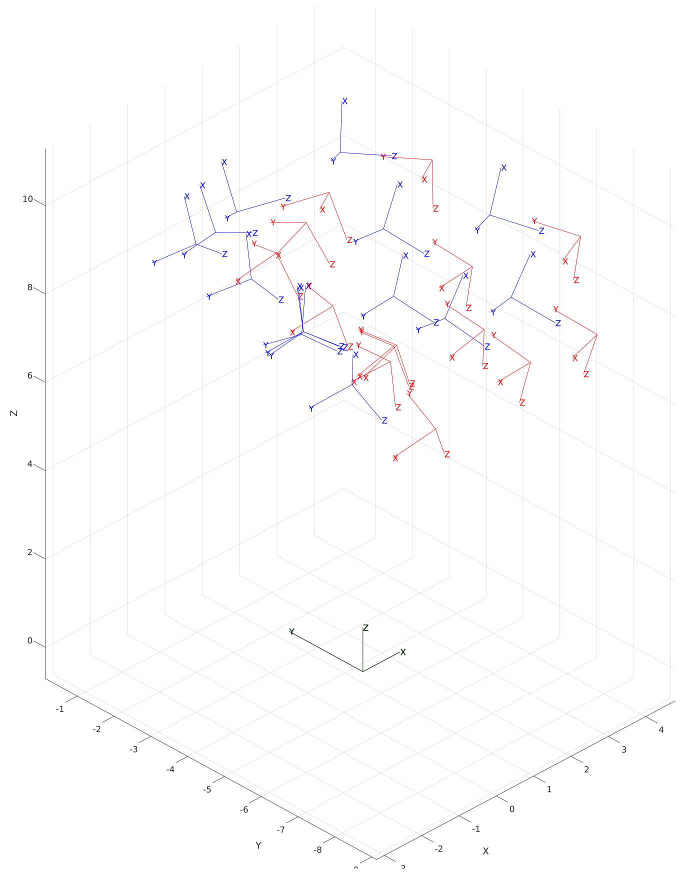


Figure 3.8: Illustration of the camera poses (blue frame), robot hand (red frame), and the checkerboard (black frame) used in the calibration process. The images were sampled with large variation in positions, in order to obtain a representative calibration for the entire work space.

3.1.6 Master Computer

The master computer was the control unit in the system, and received and transmitted signals to the other components in the system. The computer was a custom build computer with 16 GB RAM, and a Intel Core i7-4790K CPU processor operating at 4.00 GHz $\times$ 8. The graphic card was a GeForce GTX TITAN X, and the computer was run on Ubuntu 16.04.

3.2 Software Development

The software for the system was mainly written in Python, but some aspects of the software were written in C++ and MATLAB. The programs that were written in Python controlled the object detection module, along with the grasping and robot control modules. The Zivid camera was controlled with a C++ program, while MATLAB was used for supplementary modules.

3.2.1 Image Acquisition

The Zivid camera was controlled through a C++ program, and was responsible for the RGB-D image acquisition. The program initiated the connection between the computer and the camera, and defined the camera settings that had been pre-calibrated for better results. The camera acquired four images with varying iris openings (13, 20, 30 and 70), before HDR processing was performed. The specific camera settings for our setup can be seen in Table 3.2. The Zivid camera writes the captured data as a `.zdf` file that contains all the information obtained by the camera. Therefore, the `.zdf` file had to be post processed in order to extract the desired RGB and depth image. This was done with an image extraction module written in Python, that extracted the R, G, B and D channels, before combining them into an RGB image and a depth image, respectively. Both the Zivid program in C++ and the image extraction module in Python utilised several imaging processing functionalists, in order to enhance features and to reduce noise in the image. The Zivid API contained several classes where camera settings could be adjusted and different filters may be applied to the RGB-D image. The image extraction module imported functionalities from the OpenCV and NumPy API, and used filters to reduce noise, and to change other properties of the image. The following is a sequenced explanation of the processing steps used in in the Zivid camera and image extraction module, in order to capture a high quality depth image.

Zivid Camera:

- **Exposure Time**
  The exposure time defines the time interval where image sensor is exposed to the light of the scene. This parameter is correlated one-to-one with the amount of light entering the camera, and will therefore effect the overall brightness of the image. The Zivid class *Zivid::Settings::ExposureTime* was used to set the exposure time, and is assigned a value between 8333 microseconds and 100000 microseconds.

- **Brightness**
  The brightness defines the light output form the projector. This parameter will affect the overall enlightenment of the scene, which is projected from the structured light projector. The parameter was defined in the class

Zivid::Settings::Brightness, and the brightness is assigned a value between 0 and 1.

- **Outlier Filter**
  The outlier filter removes points based on the number of connected components. This filter will decrease the amount of noise in an image, and is explained in detail in Section 2.1.3.4. The filter was enabled through the class *Zivid::Settings::Filters::Outlier*, and the threshold of the filtering (the maximum number of connected components to be removed) was defined in the class *Zivid::Settings::Filters::Outlier::Threshold.* The threshold is assigned an integer above 0.

- **Reflection Filter**
  The reflection filter detects and removes points likely to arise from reflections, which is useful for shiny materials. Shiny objects may reflect some of the structured light pattern onto other surfaces, resulting in noise when calculating depth in the image. The filter was enabled in the class *Zivid::Settings::Filters::Reflection.*

- **Iris Opening**
  Several iris openings were defined for each of the images used in the HDR reconstruction, which is described in detail is Section 2.1.3.1. The iris opening effects the amount of light absorbed by the camera, and therefore allows the camera to capture images in different light ranges. The iris opening was defined in the class *Zivid::Settings::Iris*, and is assigned a value between 0 and 72.

Table 3.2 contains the specific camera settings used in our setup, that were found though experiments using different camera settings on different objects.

Zivid 3D Camera Settings	
Setting	Value
Exposure Time (microseconds)	11600
Brightness	1.00
Outlier Filter	ENABLED
Outlier Threshold	5
Reflection Filter	ENABLED
Iris Size	13U, 20U, 30U, 70U

Table 3.2: Table containing the specific camera settings for the Zivid 3D camera used in our practical work.

Image Extraction Module:

- **NaN Value Removal**
  NaN (Not a Number) values are a form of noise that is often present in

depth images, and occurs in areas of the image where the camera is unable to calculate the depth values from the structured light patterns. These values were removed with the NumPy imported function *np.nan_to_num()*, which replaces the NaN values with a defined value.

- **Image Resizing**
  The initial size of the depth images was 1920×1020, but for the purpose of reducing computational time during the bin picking operation, the depth image was resized to 720×450. This was done with the imported OpenCV function *cv2.resize()*.

- **Median Filtering**
  A median filter was applied to the depth image, for the purpose of reducing noise in the image. Isolated noise pixels was replaced with a more probable neighbourhood pixel value, resulting in a more complete depth image. Median filters are explained in Section 2.1.3.3, and the function was imported from OpenCV through *cv2.medianBlur()*.

- **Bilateral Filtering**
  A bilateral filter was applied to the depth image, to further smoothen the image, while preserving the edge features in the image. The operation is explained in Section 2.1.3.2, and the function was imported from OpenCV through *cv2.bilateralFilter()*.

3.2.2 Object Detection

The object detection module was written in Python, and was based on a deep neural network approach that used the TensorFlow framework. This approach was chosen over traditional object detection methods, due to the tendency seen in object detection benchmarks and competitions the recent years. The datasets described in Section 2.1.5.4 are often used as benchmarks for object detection, where new algorithms can be tested and compared to already existing approaches.

Object Detection Benchmarks

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [20] was created as a competition for image classification and object detection based on the ImageNet dataset, and is today considered the most prestigious computer vision competition. Figure 3.9 illustrates the performance of the winners of the classification competition from 2010 until 2016 along with the general approach used. Since the introduction of deep neural networks in 2012, a significant improvement can be seen in terms of top five score.

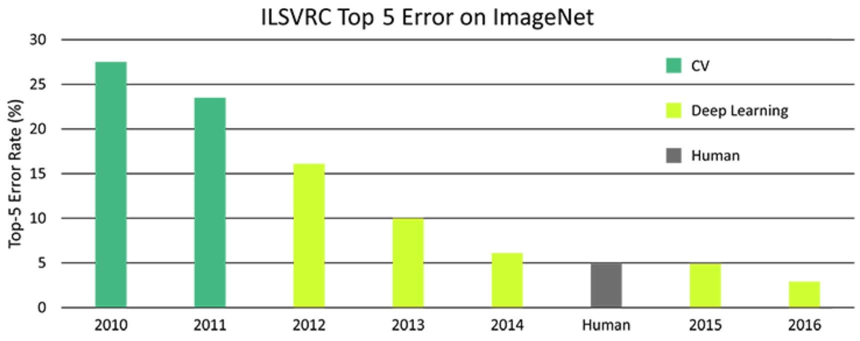


Figure 3.9: Best results on the ILSRRC competition from 2010 to 2016 for image classification. The introduction of neural networks in 2012 greatly improved the top five error score. The top five score indicates that the correct classification is present within the first five most probable classifications. [74]

Another annual benchmarking competition is the *Pascal VOC* challenge, which uses a dataset of 11530 images consisting of 27 450 RoI annotated objects and 6929 segmented objects. Figure 3.10 illustrates the winning approaches from 2007 until 2014, and the achieved top 5 error rate. As with the ILSRRC challenge, a significant improvement can be observed after the introduction of deep neural networks in 2013.

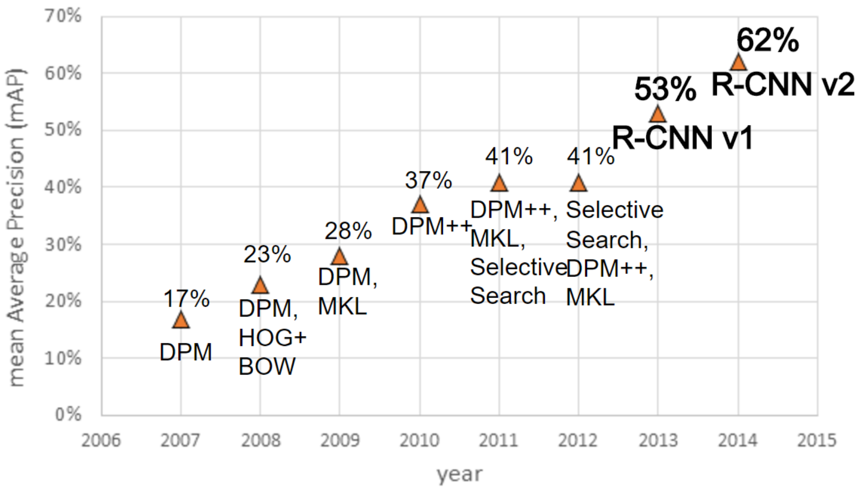


Figure 3.10: The best scores each year in the Pascal VOC challenge from 2007-2014. The introduction of deep neural networks in 2013 significantly increased the mAP score, compared to previous years. [75]

The leaderboards for the KITTI and COCO datasets for object detection related classifications are also dominated by deep learning approaches [76] [77].

The general tendency after the introduction of deep learning for object detection is clear, as it has improved the performance of object detection algorithms. Also, the traditional methods seems to have stagnated on a non-pleasing level. For our system, we therefore trained two networks for the task of object detection using the *TensorFlow* framework.

TensorFlow

The TensorFlow framework was developed by the Google Brain team and was released as an open source software at the end of November 2015. The software library allows for large numerical computations, using data flow graphs. The nodes in the graph represents mathematical operations, while the graph edges represents the multidimensional data arrays (tensors) communicated between them. There are several possible deep learning frameworks available, but TensorFlow was chosen as the framework for our deep learning network module based on the following factors:

- **General Market Development**
  There are today several different available framework for deep learning, but none of them are yet to be considered a standard in the marked. However, the market development indicates that TensorFlow is the preferred framework among developers and programmers, and we wanted our framework

to reflect the development at large software corporations, and the community in general. Large international companies e.g. Google, ebay, Intel and Lenova uses the TensorFlow framework in their products. Figure 3.11 illustrates the number of stars given on the TensorFlow repository as it was introduced on GitHub, which substantiates its popularity among developers. The same tendency can be seen in the number of Stack Overflow posts related to each framework [78].

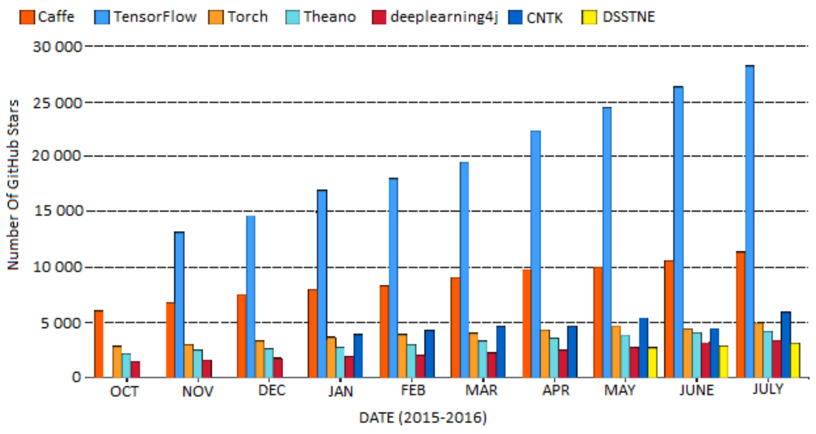


Figure 3.11: Histogram plot illustrating the number of stars on the TensorFlow GitHub repository compared to other framework repositories. The graph substantiates the popularity of the TensorFlow framework among developers. [79]

- **Availability of Pre-trained Models**
  TensorFlow offers a large variety of pre-trained networks for object detection on GitHub [80]. This allowed us to choose and customise pre-trained models according to the specific use case, by applying transfer learning.

- **Availability of Cloud Computing Optimised for the Framework**
  Deep learning training requires a large amount of computational power to train networks. One way of solving this issue is to outsource the computation to a cloud computing service such as Google Cloud Platform (GCP). TensorFlow is supported by most of the cloud computation services, but is facilitated on GCP [81] (since GCP and TensorFlow both are developed by Google).

- **Extensive Documentation, Tutorials, and Guides**
  We had little previous experience on the filed of deep learning, and were

therefore dependent on sufficient documentation, tutorials, and guides for building our system, and for understanding deep learning in general. TensorFlow offers high quality tutorials on their home page [82], along with an extensive API documentation [83].

The two networks that we trained for our object detection module were:

- Faster R-CNN
- SSD

Both networks are described in detail in Section 2.1.5.6, and were also available with pre-trained weights on the TensorFlow Model repository on GitHub [84]. The achieved scores and speeds for the two networks are listed in Table 3.3, and are based on the COCO dataset. The two networks were chosen due to previous performed experiments, where the networks have showed convincing results [39], [41]. The networks were also relatively fast compared to other models listed in the TensorFlow Model Zoo, which is important for processing speed.

Pre-Trained Object Detection Models		
Model Name	COCO mAP Score	Speed (ms)
ssd_inception_v2_coco	24	42
faster_rcnn_inception_v2_coco	28	58

Table 3.3: Achieved score and speed on the COCO dataset for faster_rcnn_inception_v2_coco and ssd_inception_v2_coco. A Nvidia GeForce GTX TITAN X graphics card was used in the test.

Deep learning algorithms are dependent on a large training datasets to achieve good results in terms of classification. The datasets are used for training and validation, and are often hard to acquire for specific use cases. The training phases also requires an extensive amount of computational power, which might not be easily accessible.

Image Gathering

The issue of capturing images for our training set was solved and automated by creating a camera rig where the Zivid camera was facing towards a rotating table. In the processes of gathering images, we placed one object at the time on the rotating table. The image acquisition was controlled by a Python script, and sequences of 20 images were captured where the object was placed at different positions on the rotating table and with different camera angles. After 500 images of each objects, we captured 500 more in a realistic setting with multiple objects present in the same image. A high percentage of our dataset consisted of single objects, something that might be an issue in terms of variation during training. However, Girshick et al. [38] demonstrated that good results can be obtained for detecting multiple objects within an image, when training on a dataset consisting of only single objects. Our final dataset had a total of 625 images/class, which is approximately the same ratio as the ImageNet dataset (650 images/class).

Labeling the Dataset

We were able to minimise the time spent on manually labeling our training images, by manipulating the scene in which the objects were placed. The majority of the images in the dataset were taken on a uniform black background, where the objects were clearly isolated on each image. With the use of binarization and morphological operations, we obtained a segmented image with a bounding box proposal around each object. The program was written in MATLAB, and an illustration of the result from the labeling process is shown in Figure 3.13. This process is also demonstrated in the demonstration video for this thesis, which is attached in the digital appendix and online [85]. From the segmented image we extracted the bounding box ($\mathbf{BB} = [x_1, x_2, y_1, y_2]$), which together with the class label represented the ground truth used for training. The final label also contained information regarding the image path and image dimensions, and was saved in a `.xml` file that contained all the labeling information. The general pipeline for the operation can be seen in Figure 3.12, where the different steps from the unlabeled to the labeled dataset are illustrated. The labeled images and `.xml` files were eventually run through a manual labeling program, for quality control and correction when needed. The labels and images were then converted into a single `.TFRecord` file, which is recommended file structure to use in TensorFlow during the training phase.

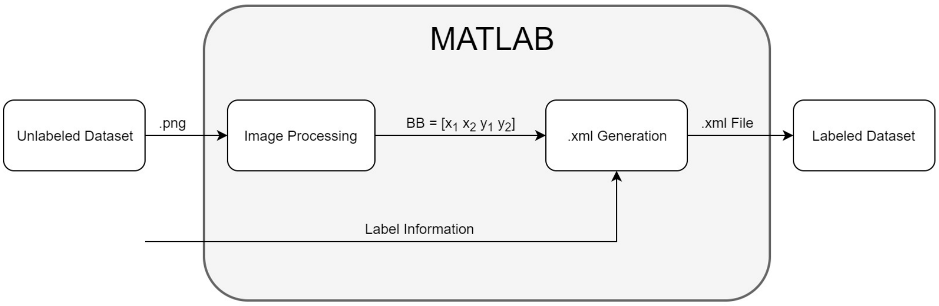


Figure 3.12: Illustration of the automatic labeling pipeline. Images from the training set were imported to MATLAB along with labeling information. The images were processed in order to obtain a binary representation, which was used to extract the bounding box **BB** around an object. The bounding box information was combined with the labeling information as the `.xml` file was created and exported.

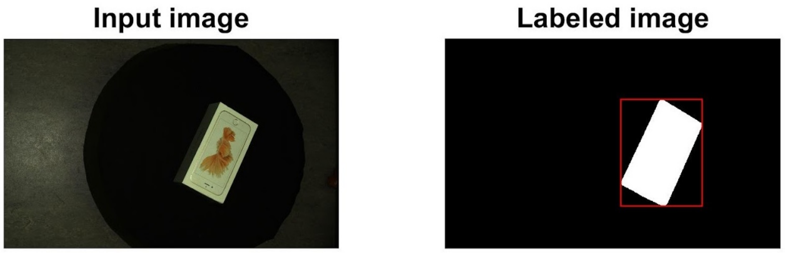


Figure 3.13: Illustration of the proposed bounding box region from the automated labeling program.

Training

Training a complex deep neural network on a large dataset is a time consuming operation, so we therefore applied transfer learning as described in Section 2.1.5.5. The most basic features such as edges, corners and colors were therefore already learned from a much more extensive dataset than our own, and as a consequence, we only needed to train the last layers in order to detect our four objects. Even so, a considerable amount of calculation power was necessary, and training on our local CPU became too time consuming. This issue was resolved by outsourcing the computation to a cloud computing service. There are several possible providers of this service, where the most popular are *Amazon Web Services*, *Google Cloud Platform* (GCP) and *Microsoft Azure*. Google's computing service supports the TensorFlow framework, and also allows for 300$ worth of computation for new users. This made GCP our desired choice, where we executed the computation. The resulting training time was drastically reduced to about an hour.

In order to train networks on the cloud, the input data, code, and dependencies had to be available to Google's servers. Our local computer and GCP communicated through Google Cloud SDK, which allowed us to perform operations on the cloud from the terminal on our Ubuntu computer. This would in turn allow us to upload our data, and to export the trained data files to our computer.

Running the command `gcloud init` allowed us to access several common SDK setup tasks, such as authorising the SDK tools to access the GCP, and setting up the default SDK configuration. After the successful configuration, the labeled dataset for our networks was uploaded to a *bucket* in the cloud, along with the hyperparameter configuration and file structures. The bucket is the storage center where the files are organised on GCP. Uploading was performed with the `gsutil cp` command, which allowed data to be copied between the local file system and the cloud. The uploaded files included the frozen weights from the pre-trained models, along with the training data, configuration for the hyperparameteres and the architecture of the network. Uploading the specified folder and files was done with the command

```
$ gsutil cp -r dir gs://objectdetection_prosjekt
```

Before starting the training phase, the configuration details had to be assembled. During training, the Cloud Machine Engine service will allocate one or several virtual machines based on the job configuration chosen. The computing specifications in terms of CPU, GPU and RAM was defined and adapted to the requirements of our networks. The hyperparameters were also defined, and the final values are summarised in Table 3.4.

Hyperparameters		
Hyperparameter	Faster R-CNN	SSD
Number of classes	4	4
Dropout rate	0.8	0.8
Activation function	RELU	RELU
Score converter	Softmax	Sigmoid
Learning rate	0.0003	0.004
Momentum	0.9	0.9
Batch size	1	24
Evaluation size	167	167
Initialiser (ssd,mean)	(0.1,0)	(0.03,0)
Loss function	Loc: L_1 Clas: Cross entropy	Loc: L_1 Clas: Sigmoid

Table 3.4: Overview of some chosen hyperparameters used during training of the Faster R-CNN and SSD networks. Loc is short for location loss, and Clas is short for classification loss.

During training, the weights and biases in the model were saved at given intervals after a given number of global steps was processed. These checkpoints allowed us to validate the model against the test data during training, and to continue training at a later point in time.

3.2.3 Grasp Planning

Gasping was planned using the GQ-CNN deep neural network presented in Section 2.2.3.2 for grasp quality prediction. While we trained two networks for object detection, we did not prioritise to train a new network for grasp quality prediction. The reason being that it would be a tedious process to generate a large enough training set with grasp proposals. Also, the characteristics that determines whether a surface is suited for grasping is not directly dependant on specific objects, but rather the properties of the objects.

The GQ-CNN model trained on the Dex Net 3.0 dataset claims to be the one of the most robust solutions so far in terms of success rate on novel objects (98% on simple shapes, 82% on typical household items, 58/81% on adversarial objects) [60]. The pre-trained GQ-CNN model is trained on the Dex Net 3.0 dataset that contains 2.8 millions single viewpoint point clouds with corresponding suction grasp with robustness labels.

The viewpoint in the training dataset is directly above the objects, with a distance of 65 cm and with a planner background. Due to this static viewpoint, a similar viewpoint was necessary in our physical setup. The depth image also had to be processed before being used by the network, to resemble the training data as much as possible in order to obtain robust results. We also took advantage of the pipeline proposed in the Dex Net 2.0 paper [86] for utilising the network, and the corresponding GitHub repertory [87].

The GQ-CNN takes a depth image and a grasp proposal as inputs and returns a grasp quality score, which is the predicted probability of success. A sample of possible grasp candidates is thus needed to be generated in order to find a robust quality grasp. Algorithm 1 shows the pseudo code of the grasp sampler, and Algorithm 2 shows how we utilised the grasp sampler and GQ-CNN to obtain a final grasp location. Following is a list of the most important parameters in the code, and how they affect the behaviour of the grasping module.

- **SamplesInit**
  The number of initial grasp location sampled in the image. These locations are placed at random locations within the allowed work space area, which is defined in the code. A high num_seed_value results highly representative sampling of the entire image, but results in a longer computation time.
- **SamplesIter**
  The number of sampled grasps around the best proposed locations from the previous sample iteration.
- **numIters**
  Number of sample-and-refit iterations to compute the final grasp location.
- **minDist**
  Minimum admissible distance between suction points. A high value ensures increased diversity of the samples.
- **angleThresh**
  The maximum angle between the optical axis of the camera and the grasp approach axis (surface normal). This ensures that the grasp is possible to be executed with the limitations that occur due to the walls in the bin.
- **minDepth**
  The minimum allowed picking distance from the camera in z direction.
- **maxDepth**
  The maximum allowed picking distance from the camera in z direction.
- **pgraspThresh**
  A threshold on the prediction score from GQ-CNN during grasp sampling. A low threshold increases computation time since more grasps are evaluated between each iteration.

[0]https://berkeley.app.box.com/s/szbchyt3tou9e4ct6dz8c5v99vhx0s84

- **GaussianVariance**
  The sampled points generated form the previous iteration are placed around the proposals with scores over pgraspThresh. These samples are distributed over the previous proposal using a Gaussian distribution with variance G. A high G results in higher diversity in the samples.

Algorithm 1: Grasp Sampler(depthIm,Gasussian=false,pgrasp=false)

```
Result: Sample of grasp candidates
grasp;
SamplesInit;
SamplesIter;
angleThresh;
MinDist;
minDepth;
maxDepth;
pointCloudNormal = surfaceNormal(depthImage);
GaussianVariance;
if pGrasp is true than then
    Samples = SamplesIter;
else
    Samples = SamplesInit;
end
while length of grasps is lower than Samples do
    if Gaussian is true then
        grasp(u,v) = RandomGaus-
          sian(GaussianVariance,pGrasp(i),0:im.height,0:im.width);
    else
        grasp(u,v) = RandomGaussian(0:im.height,0:im.width);
    end
    grasp(n) = pointCloudNormal(grasp(u,v));
    dist = distToOtherPoints(grasp(u,v),grasps(u,v));
    angle = angleToCamereaOpticAxis(grasp(n));
    depth = depthImage(grasp(u,v));
    if angle is larger then angleThresh, dist is larger then MinDist, depth is
      larger then minDepth and smaller than maxDepth then
        grasp(add to end) = grasp;
    else
    end
end
```

Algorithm 2: Get Grasp

```
Result: Final grasp
30 numIters;
31 pgraspThresh;
32 grasps = GraspSampler(depthImage);
33 for i to numIters do
34     for j to length of grasps do
35         pGrasp = GQCNN(depthIm,grasps(j));
36         if pgrasp is higher then pgraspThresh then
37             pgrasps(add to end)=pgrasp;
38         else
39         end
40     end
41     grasps = GraspSampler(depthIm,Gaussian=True,pgrasp);
42 end
```

The pixel coordinates and the surface normal of the grasping point that achieved the highest prediction score from the network were used to calculate the transformation between the camera and the suction location. First, the pixel coordinates (u, v) were used to calculate the relative translation $\mathbf{O}$ between the camera and the suction location in the scene:

$$\mathbf{O}_S^C = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ d \end{pmatrix} \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}. \tag{3.2}$$

Secondly, the unit surface normal (x_u, y_u, z_u) was used to calculate the relative rotation $\mathbf{R}_S^C$ between the suction point to the camera, and was given by

$$\mathbf{R}_S^C = \begin{pmatrix} x_u^2 & x_u y_u - z_u & x_u z_u + y_u \\ y_u x_u + z_u & y_u^2 & y_u z_u - x_u \\ z_u x_u - y_u & z_u y_u + x_u & z_u^2 \end{pmatrix}. \tag{3.3}$$

Finally, Equation 3.2 and Equation 3.3 are combined to create the homogeneous transformation between the camera and the suction location:

$$\mathbf{T}_S^C = \begin{pmatrix} \mathbf{R}_S^C & \mathbf{O}_S^C \\ \mathbf{0}^T & 1 \end{pmatrix}.$$

3.2.4 Robot Control

ROS, as covered in Section 2.2.2, was chosen for the overlying control software of the Agilus1 and Agilus2 robotic manipulators, due to its general popularity among developers and the availability of detailed explanations of concepts [88], tutorials [89], and API's [90]. This software utilises several other software frameworks such as Moveit! [91] and RViz. The Moveit! software provides software plugins for motion planning, inverse kinematics, trajectory planning and supports APIs for Python and C++. Moveit! and ROS were already installed and configured for Agilus1 and Agilus2 at the robot cell at IPK, along with RViz for visualisation. This pre-setup allowed for a simple drag and drop placement of the end-effector in RViz, and the corresponding inverse kinematics, motion and trajectory planning was computed and executed by the Moveit! software. However, several modifications was necessary for our purpose in the configuration of Moveit!. New end-effectors for the robots, and objects in the scene were added to the model, in order to calculate new collision matrices and to adjust the joint limitations for our use.

Moveit! Configuration

The new configuration of the robot cell was conducted with the *Moveit! Setup Assistant*. The configuration was initialised by loading a `kuka_kr6r900sixx.xacro` file which contained the information regarding CAD files for visualisation and collision, pose and orientation of the objects relative to a world frame, and relations between parent and child links for the different CAD models. This `kuka_kr6r900sixx.xacro` file included several other `.xacro` files that contained the specific limitations for the robotic joint rotations. These limitations were modified such that the motion planner generated more desirable trajectories, in terms of the end-effector orientation and joint configurations. The modifications of the joint ranges limited the work area/joint space for the suction gripper while performing a pick, so that the external wrenches to the suction cup were reduced. It also ensures that the trajectory of the Zivid camera was controlled, so that the power cable and the USB data cable were not at risk of being stretched during motion. The CAD model of our custom made suction module was attached as the final child link of the kinematic chain of Agilus1, while the CAD model of the Zivid 3D camera was attached as the final link of the kinematic chain for Agilus2. The new joint limitations are listed in Table 3.5.

Modified Axis Range		
Joint	Agilus1	Agilus2
A1	$-140°/20$	$-30°/120°$
A2	$-120°/30°$	$-150°/-40°$
A3	$10°/140°$	$-120°/156°$
A4	$-90°/90°$	$-90°/90°$
A5	$-10°/120°$	$-120°/120°$
A6	$-180°/180°$	$90°/270°$

Table 3.5: Modified axis range for the suction gripper end-effector (Agilus1), and the Zivid camera end-effector (Agilus2). The new configurations have significantly reduced range compared to the robots maximum capability, in order to control the trajectory proposed by the inverse kinematic solver.

After loading the `kuka_kr6r900sixx.xacro` file to *Moveit! Setup Assis-*

tant, the program generated a *Self-Collision Matrix*. This matrix maps the robots physical joints that never will collide, and therefore reduces the computation time during inverse kinematics computations by defining these configurations as a default success. Two planning groups were created, where a set of joints and links were defined for each group. The two defined groups can be used by any ROS node to control the robots through a Python API named */moveit_commander*. The two planning groups we defined were named

- **agilus1**
- **agilus2**

where agilus1 was the Agilus1 robot with the addition of the gripper module. Likewise, agilus2 was the Agilus2 robot with the Zivid camera attached. The final step of the *Moveit! Setup Assistant* was to create the configuration files that defined our robot properties in Moveit!, in order to control the two newly defined planning groups.

Robot Control

The robots were controlled through a Python API for ROS and Moveit!. During system launch, a primary node is provided by Moveit! called */move_group*. This node served as an integrator between the user interface, and ROS parameter server. The node provided a set of ROS actions and services that could be used for different operations, such as inverse and forward kinematics, collision control and visualisation integration. The configuration files that were generated by the *Moveit! Setup Assistant* were imported by the node, in order to know the physical properties of the manipulators.

Our proposed system consisted of four static manipulator poses, and one dynamic pose given by the grasping module. The pipeline of the different robot poses during a picking operation can be seen in Figure 3.14, where the poses are as follows:

- **Default Start Position**
  This is the starting position of the robots, defined by the KUKA KR C4 controller. The robots will always start at this given position as the system is initialised.
- **Image_Position**
  The agilus2 planning group manoeuvres into position over the bin, in order to take an RGB-D image of the scene with the Zivid camera.
- **Zivid_Home**
  The agilus2 planning group retreats from the bin after the images are taken, so that the other robot can perform the pick.
- **Suction_Standby**
  The agilus2 planning group manoeuvres into position directly over the bin and is ready to perform the grasping operation.

- **Suction_Location**
  The aguilus2 planning group moves to the suction location given by the grasping module, in order to perform the actual picking operation.

- **Move_to_New_Bin**
  The agilus2 planning group transports the picked object from the storage bin, to a new container. The whole series of operations might be repeated for a new object, or the system can end the operation.

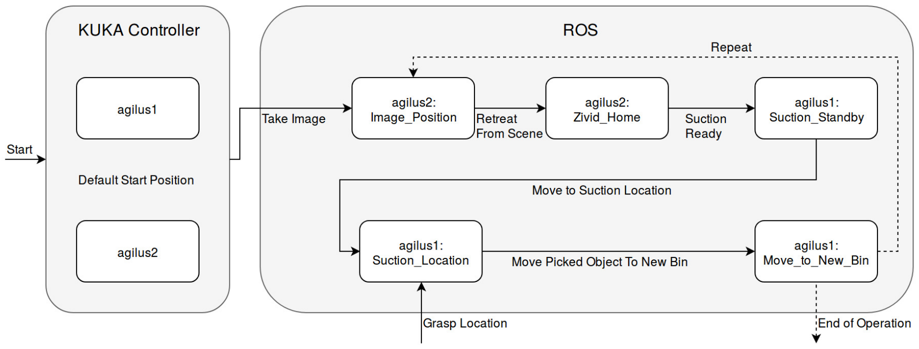


Figure 3.14: Illustration of the general movements performed by the agilus1 and agilus2 planning groups.

The end-effector frame was defined in Cartesian coordinates (x, y, z) for the location in space, and quaternion coordinates (x, y, z, w) for the orientation. The movement of the manipulators were controlled through actions provided by the mentioned Python API */moveit_commander*. First, a new node is created for the operation with the following code:

```
rospy.init_node('move_group_python_interface',
                anonymous=True)
```

A MoveGroupCommander object was instantiated to create an interface to a group of joints. The group of joints we wanted to use is referred to as a *move group*, and was defined as a planning group in the Moveit! Setup Assistant. Two objects were defined, *gripper_group* and *zivid_group*, which were used to plan and execute motions for the robot with the suction module attached and the Zivid camera, respectively.

```
  gripper_group = moveit_commander.MoveGroupCommander("agilus1")
  zivid_group= moveit_commander.MoveGroupCommander("agilus2")
```

In order to visualise the planned trajectories for gripper_group and the zivid_group, the *move_group_python_interface* node was set to publish a stream of joint values that was used by RViz.

```
display_trajectory_publisher = rospy.Publisher('/move_group/
    display_planned_path',moveit_msgs.msg.DisplayTrajectory,
    queue_size=20)
```

Moving to a defined position and orientation in space was done with the following actions, where the *state* variable contains the location and orientation necessary to perform the picking operation calculated by the grasping module:

```
pose_target = geometry_msgs.msg.Pose()
        pose_target.orientation.w = state[9][4]
        pose_target.orientation.x = state[9][5]
        pose_target.orientation.y = state[9][6]
        pose_target.orientation.z = state[9][7]
        pose_target.position.x = state[9][1]
        pose_target.position.y = state[9][2]
        pose_target.position.z = state[9][3]
        gripper_group.set_pose_target(pose_target)
```

A motion plan was created and visualised by RViz for the target as the motion was being executed:

```
plan1 = gripper_group.plan()
        display_trajectory.trajectory.append(plan1)
        display_trajectory_publisher.publish(display_trajectory);
        collision_object = moveit_msgs.msg.CollisionObject()
        gripper_group.go(wait=True)
        gripper_group.execute(plan1)
```

3.3 Complete System

The complete bin picking system was modelled as a state machine, which is illustrated in Figure 3.15. Each state represents a sequence of operations in the system, and the machine can only be in one state at any given time. This structure allowed us to control the conducted sequence of operations during bin picking operations, where transition between two states are controlled by the inputs and outputs from the previous state. This allowed the system to recover from errors or failures that might occur in a state, without necessarily needing to restart the whole system. The state variables and all inputs and outputs of each state were defined in the main Python script as a global array:

```
  globals()['state'] = np.array([['Robot_zivid_state', 0], ['
    Robot_gripper_state', 0],['Zivid_pose', 0],['Image_state', 0],['
    Gripper_pose', 0],['Grasping_pose', 0],['Vacuum', 0],['
    Pick_success', 0],['Place_success', 0],['Tw_g', 0,0,0,0,0,0,0]],
     dtype=object).
```

A full overview of the inputs and outputs of the system are listed in Appendix A.2.

During operation, a typical series of operations were conducted:

- **RobotsNotConnected**
  This is the initial state of the system, which initialises robot connection to the master computer, along with communication between the master computer and the Zivid camera.

- **RobotsHome**
  This is the initial operating state where all necessary system components are connected, and the system is ready to perform bin picking.
- **Perception**
  This is the state where perception related computations are conducted. This includes the image acquisition, image processing, object detection, and grasp computation.
- **GripperReady**
  This is the state that performs trajectory planning and gripper activation.
- **Grasping**
  This is the state that performs the final suction and picking operation.
- **Placing**
  This is the state that transports the grasped object from the initial bin, to the new bin.

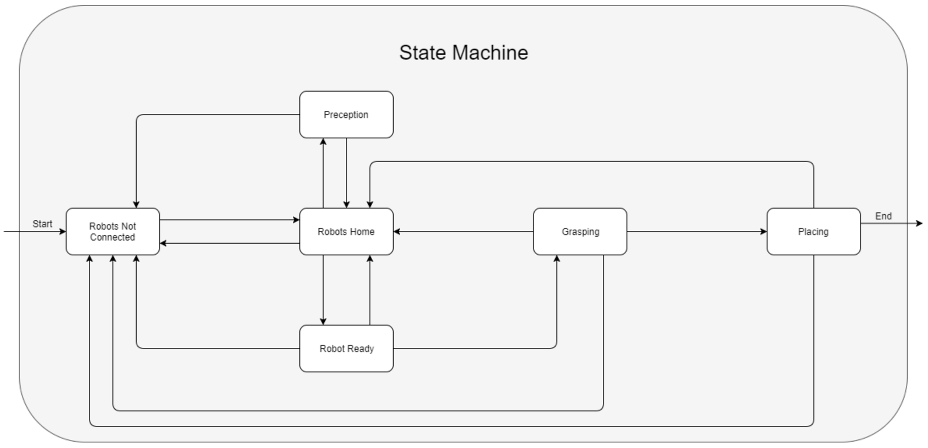


Figure 3.15: Diagram that illustrates the different states of the system and the possible transitions. The condition for the transition between two states were given by the outputs from previous state, and is described in detail in Appendix A.2

.

Chapter 4

Results

The results from our proposed system will be presented in the following chapter. First the results from the individual modules and modifications are presented, and thereafter the system as a whole.

4.1 Physical Results

The following section will cover the results and discoveries made related to the physical development of the practical work.

4.1.1 Physical Setup

The proposed bin picking system consisted of several physical components that communicates through a series of input and output ports. Figure 4.1 illustrates a simple schematic representation of the system, where the master computer controls the Zivid camera, and the KUKA robots.

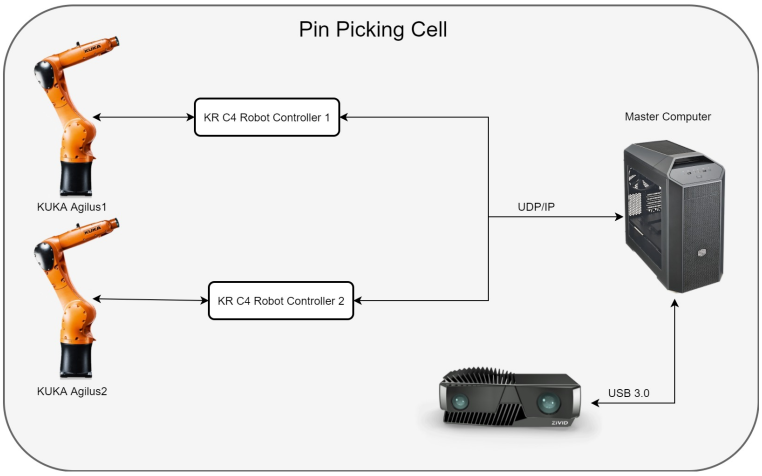


Figure 4.1: Pipeline of the hardware in our proposed bin picking cell, where the master computer is the control unit in the system.

4.1.2 Eye-In-Hand Calibration

The eye-in-hand calibration was performed according to Section 3.1.5 with software provided by the Zivid company. In total, 17 images were taken along with the recorded robot pose at each iteration. The output from the calibration software were the intrinsic and extrinsic parameters of the camera, and the distortion coefficients.

Intrinsic Parameters

The intrinsic parameters were contained in a camera matrix $\mathbf{K}$ is defined as

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

where f is the focal length of the camera, s is the axis skew, and (x_0, y_0) is the optical center measured in pixels. These values are only a function of the camera, meaning that they are independent of camera positioning and orientation in space. However they may differ among assumed identical camera models, due to montage differences and production variations of the individual parts.

The resulting camera matrix of the calibration was estimated to

$$\mathbf{K} = \begin{pmatrix} 1042.844930 & 0.000000 & 361.2227610 \\ 0.000000 & 1043.510680 & 233.4071770 \\ 0.000000 & 0.000000 & 1.000000 \end{pmatrix}$$

and camera distortion coefficients $\mathbf{K}_d$ was estimated to

$$\mathbf{K}_d = \begin{pmatrix} -0.285000 & 0.414000 & 0.141000 & 0.00149 & 0 \end{pmatrix}.$$

The parameters obtained from $\mathbf{K}$ and $\mathbf{K}_d$ were used in the transformation of pixel coordinates to image coordinates, which was necessary to express the position of objects detected in the scene. The result of applying lens correction with the values from $\mathbf{K}_d$ can be seen in Figure 4.2.

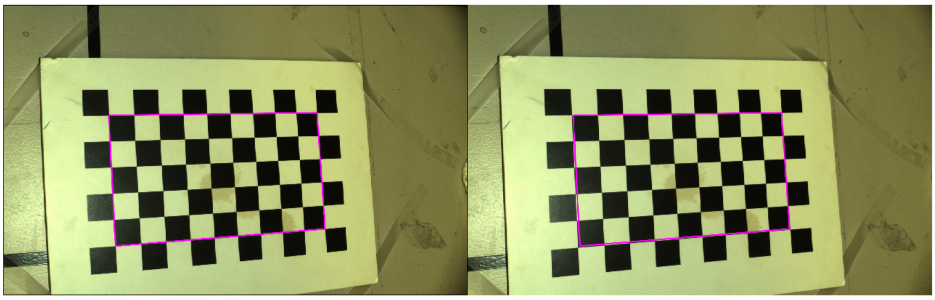


Figure 4.2: The left image is taken before the lens correction, where a square is drawn between the outer corners. The left image illustrates the effect of the lens correction, where the identical square is drawn over the corrected image.

Extrinsic Parameters

The extrinsic camera parameters describes the cameras location and orientation in the world. The calibration estimated the homogeneous transformation matrix between the robot hand and the mounted camera $\mathbf{T}_C^H$ as

$$\mathbf{T}_C^H = \begin{pmatrix} -0.21450341 & -0.0099235 & -0.9766728 & -0.01687626 \\ 0.97644116 & -0.02620892 & -0.21418624 & -0.08095245 \\ -0.02347206 & -0.99960723 & 0.01531161 & 0.16465542 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

The mean translation deviation was estimated to 0.0032 ± 0.0015m and a mean orientation deviation of 0.0045 ± 0.0024rad. The deviations were measured by compering the estimated position calculated using $\mathbf{T}_C^H$ in some random sampled positions against the true positions as

$$\mathbf{T}_C^W \mathbf{T}_H^C - \mathbf{T}_H^W = \text{Deviation}$$

where $\mathbf{T}_H^W$ is the transformation from the world frame to the hand of Agilus2, $\mathbf{T}_C^W$ is the transformation from the world frame to the camera, and $\mathbf{T}_H^C$ is the transformation from the camera to the hand on Agilus2.

The transformation matrix $\mathbf{T}_C^H$ was used to estimate the rigid transformation $\mathbf{T}_G^W$ between the world frame W and the grasping location G:

$$\mathbf{T}_G^W = \mathbf{T}_H^W \mathbf{T}_C^H \mathbf{T}_S^C \mathbf{T}_G^S$$

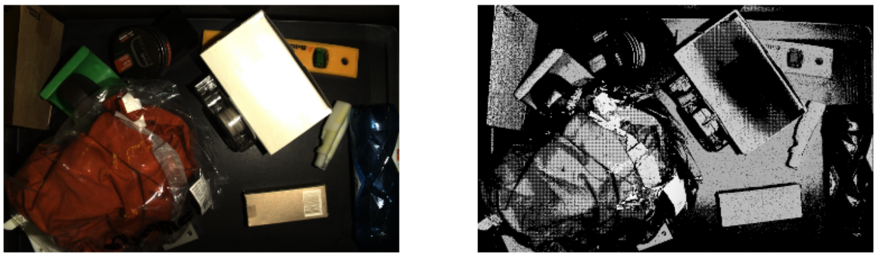
where $\mathbf{T}_S^C$ is the transformation from the Zivid camera to the suction location found bu the grasping module, and $\mathbf{T}_G^S$ is the transformation between the suction location and the hand of Agilus1. The transformation $\mathbf{T}_G^W$ was used by Moveit! to generate the trajectory to the suction location that was executed by the robots, as described in Section 3.2.4.

4.2 Software Results

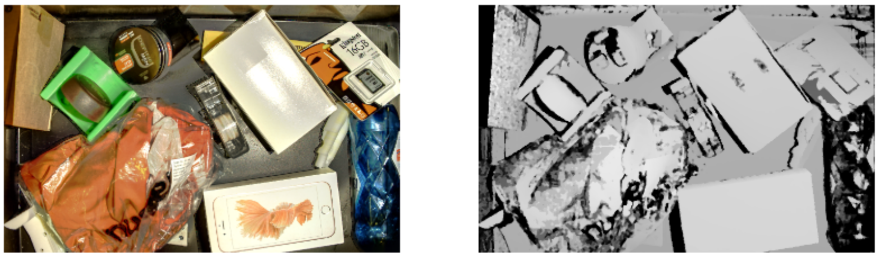
The following section will cover the results and discoveries made related to the software development of the practical work.

4.2.1 Image Acquisition

Following the proposed image processing pipeline from Section 3.2.1, we were able to capture high quality depth images of the bin with the Zivid camera. This was essential for the picking operation, as the grasping module determines the suction location based on this depth image. Figure 4.3 illustrates the improvement from a captured depth image where the default camera settings are used, to a depth image where we used our proposed camera settings and processing tools.



(a) Depth image using default camera settings



(b) Depth image using custom camera settings and post processing.

Figure 4.3: Comparison of the initial depth image which was acquired while using the default camera settings on the Zivid camera (a), and a depth image with custom camera settings and post processing (b).

4.2.2 Object Detection

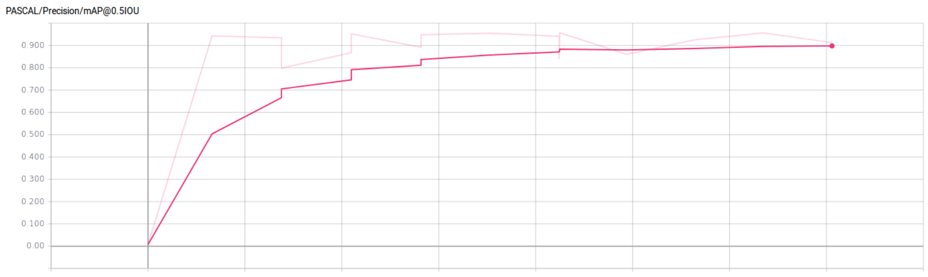
The following section will present the results obtained form the development of our object detection module. The results are separated into a training section and testing section.

Training

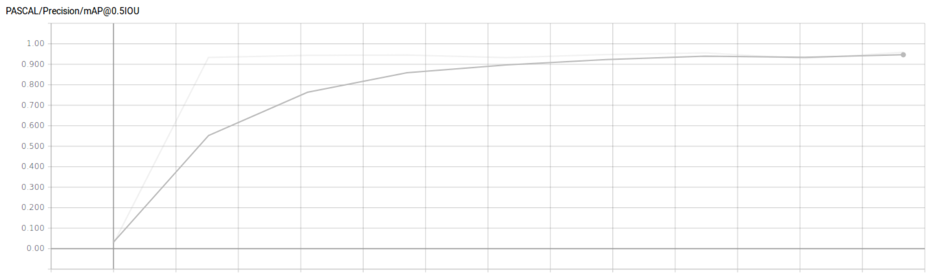
During the training of the SSD and Faster R-CNN networks, the visualisation tool *TensorBoard* was used to interpret the results of the networks, and to get an understanding of how the network behaved during training and evaluation. TensorBoard was accessed though the command

```
$ tensorboard --logdir=gs://objectdetection_prosjekt
```

which created a local host server where a visual representation is presented by TensorFlow. TensorBoard allowed us to evaluate to progress of the networks as they trained on the provided dataset, through graphs, diagrams, and histogram plots. Based on the theory presented in Section 2.1.5.3 regarding overfitting during training, we continuously monitored the development of the evaluation accuracy as the training progressed. Several configurations with different hyperparameters were tested for both networks, to evaluate the performance and behaviour during training. On the basis of the theory covered in Section 2.1.5.2 and the results we observed, we found that hyperparameters listed in Table 3.4 yielded the best results. The validation precision for both networks can be seen in Figure 4.4, and the corresponding total loss graph during the same training period can be seen in Figure 4.5.



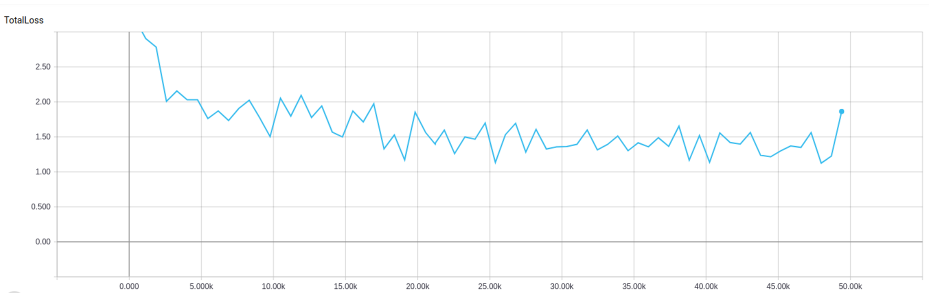
(a) Validation accuracy SSD



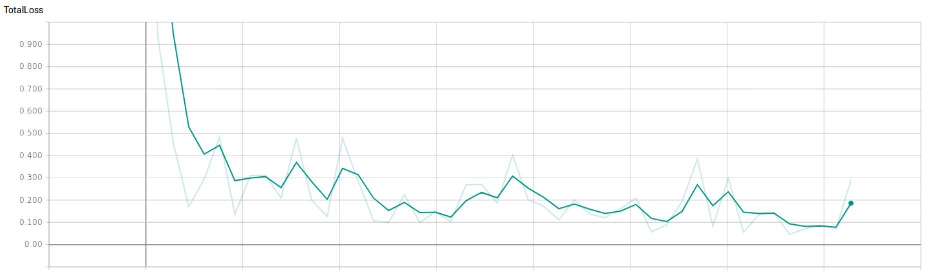
(b) Validation accuracy Faster R-CNN

Figure 4.4: Illustration of the development for validation accuracy for the SSD and the Faster R-CNN networks. Both graphs shows that the accuracy converges to between 0.9 and 1.

The validation precision is calculated according with Pascal mAP@0.5IOU [92]. From the validation accuracy curve we observed that both networks experienced a rapid increase in accuracy, before it stabilises around 0.9. Is is not possible to exclude the possibility of overfitting being present in the networks at a given point, as we do not observe a sudden decrease in the score. This can be explained by the limited variation we have between the training set and the validation set, as these are relatively similar. The results form the total loss graphs indicated that our trained networks yielded sufficient classifications for the training set. The validation graph indicated that the networks converged towards their highest validation accuracy, and we therefore stopped the training and saved the frozen graphs at the highest peak in the validation accuracy. Further testing after we exported the trained networks showed that they performed satisfying results on new images, and we therefore stopped further training with new hyperparamters.



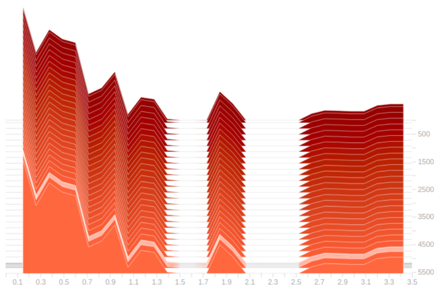
(a) SSD



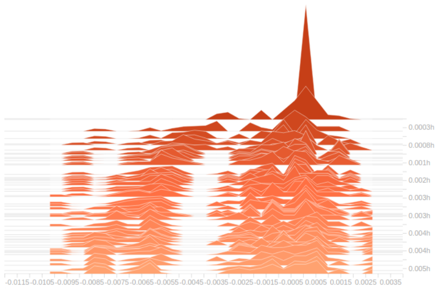
(b) Faster R-CNN

Figure 4.5: Illustration of the total loss development of the two networks during training. The total loss curves converge towards 1.2 and 0.1 for the SSD and the Faster R-CNN networks receptively.

The illustration in Figure 4.6 (a) illustrates the biases in a feature extraction layer early in the Faster R-CNN network structure, where the height of each histogram represents the frequency of occurrence for a given bias value. The figure illustrates that the frozen bias values that were imported from transfer learning are not changed during the training period, because the layer extracts a general feature that is not specific for our objects. Figure 4.6 (b) illustrates how the bias values from the class prediction layer evolved during training, as these were trained for our specific objects. We observe that the biases initially were normally distributed around zero, but rapidly changed in the fist steps. The bias values stabilised at the same number of steps as the total loss and validation accuracy stabilised. More graphs from our trained models can be found in Appendix A.3.



(a) The frozen biases from a feature extraction layer.



(b) Evolution of the bias values from the class prediction layer.

Figure 4.6: Histogram illustrating the evolution of the bias values from a pre-trained feature extraction layer, and a class prediction layer during training for the Faster R-CNN network.

Testing

The results from TensorBoard indicated good results for both networks, where the Faster R-CNN network seemed to have a small advantage in terms of the validation accuracy and total loss results. To test and compare our networks in a realistic setting, we created a new dataset consisting of 40 images in realistic and challenging bin situations. The trained networks were loaded into a Python script along with the new images, that outputted all object predictions with bounding box proposals that were over a given threshold $t = 0.75$. Some of the results can be seen in Figure 4.7, and proves that both networks were able to detect the test objects, but in some varying degree. However, the results illustrates the neural networks ability to learn and generalise features, as it detects objects in new positions and orientations not used in the training set.

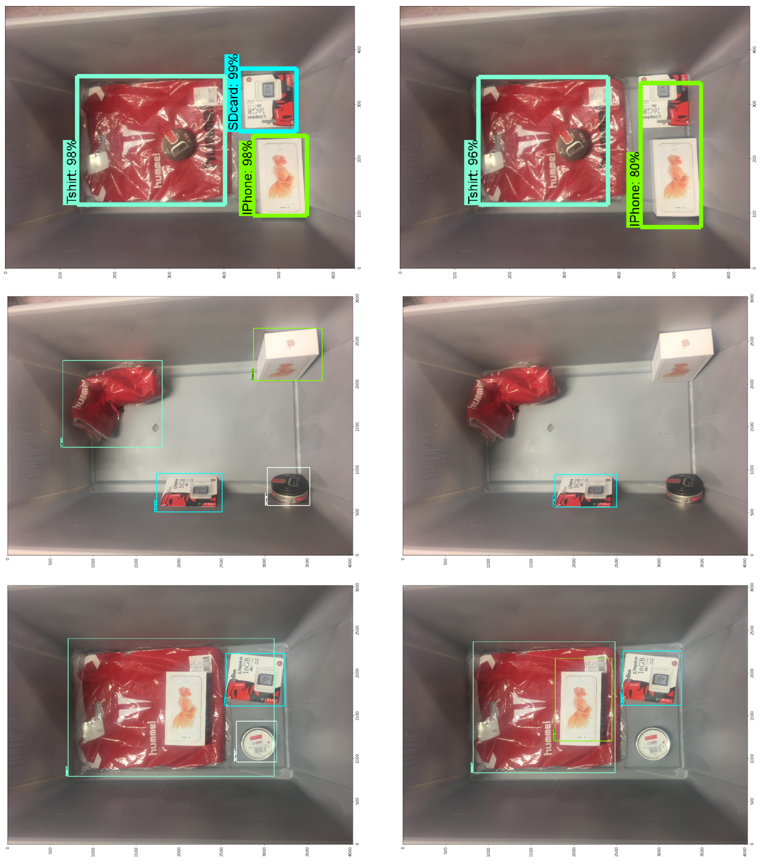


Figure 4.7: Illustration of the prediction made of the SSD and Faster R-CNN networks. The left side of the figure are the results from the Faster R-CNN network, and the right side are the results from the SSD network.

A score value S_1 for both the networks was defined and calculated with the given formula

$$S_1 = \frac{\text{TruePositive}}{\text{TruePositive} + \text{TrueNegative} + \text{FalsePositive}}$$

where

- **TruePositive**
  indicates that the network was able to output a classification, and the classification was correct.
- **TrueNegative**
  indicates that the network was able to output a classification, but the classification was incorrect.

- **FalsePositive**
  indicates that the network did not output a classification, even if there was a trained object in the image.

The resulting data and the calculated S_1 score for each object and the total score for the networks can be seen in Table 4.1 and Table 4.2.

S_1 Score SSD Network					
	iPhone	SD Card	Hair Wax	T-shirt	$\sum$
True Positive	11	7	4	8	30
True Negative	3	0	0	0	3
False Positive	6	4	6	1	17
S_1 score	55%	64%	40%	89%	60%

Table 4.1: Test results for the SSD network.

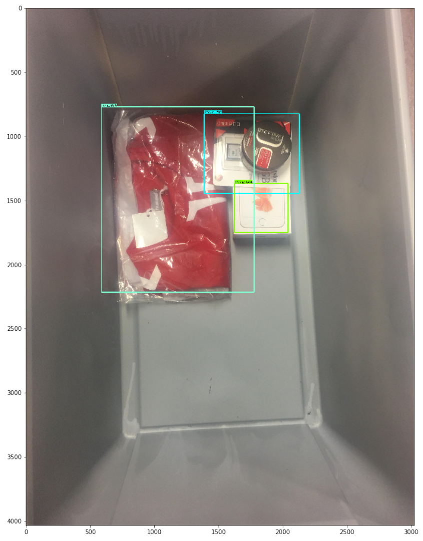
S_1 Score Faster R-CNN					
	iPhone	SD Card	Hair Wax	T-shirt	$\sum$
True Positive	17	6	7	8	38
True Negative	10	0	0	5	15
False Positive	0	5	3	1	9
S_1 score	63%	55%	70%	57%	61%

Table 4.2: Test results for the Faster R-CNN network.

The total S_1 score is almost identical between the two networks, but there are some interesting variations internally. As we can observe in Table 4.1 and Table 4.2, the Faster R-CNN network has a five times higher TrueNegative rate, compared to the SSD network. In a realistic automated bin picking situation, there would be a product ID in the incoming order, that would correspond to the correct object in the bin. In this situation, the system would score a higher accuracy, since it only outputs the prediction of the highest probability to the corresponding product ID. The accuracy in this scenario can be calculated as a S_2 score:

$$S_2 = \frac{\text{TruePositive}}{\text{TruePositive} + \text{FalsePositive}}$$

assuming that the TrueNegative probabilities generally are lower than the True-Positive probabilities. The SSD network has twice the FalsePositve rate, which contributes to a lower S_2 score. The resulting S_2 scores are calculated to 81% and 64% for the Faster R-CNN and SDD network, respectively. The S_2 score favours the Faster R-CNN network, since it generally outputs a higher number of classifications, which can be seen in Figure 4.8.


(a) The Faster R-CNN network finds the Tshirt, Iphone and SD card, but not the Hair Wax.

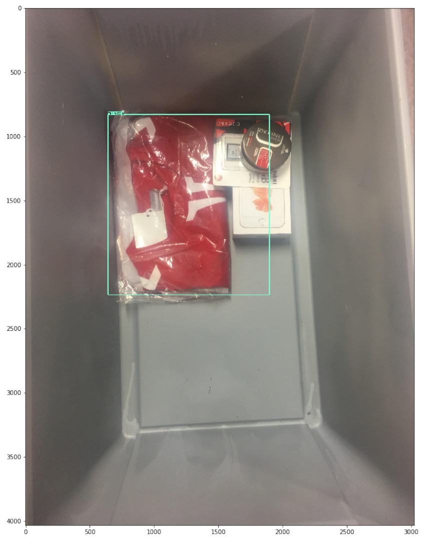

(b) The SSD network was only able to find the Tshirt.

Figure 4.8: The Faster R-CNN network generally outputs a higher amount of classifications, compared to the SSD network.

4.2.3 Grasping

Our grasping module was configured for our physical setup, and modified to perform as intended for our use case. The module first sampled a total of 250 random grasps over the entire depth image input, before three iterations of 150 grasp samples were sampled around the best proposals from the previous iteration. The sampled grasps were evaluated by the GQ-CNN network, where the final grasping proposal is the assumed best grasp of all the samples. The number of samples effects the quality and the overall processing time of the module, and should therefore be adapted to the scene in which it is applied. If the scene contains only a few objects, the grasping module will quickly converge at the assumed best grasping location. Likewise, if the scene contains several objects, a higher sample rate might be necessary in order to get a representative final suction location. A series of test were conducted to evaluate the final proposed grasping location, based on our own intuitive understanding related to grasping. The results of the tests are illustrated in Figure 4.9, and yields satisfying results in terms of accuracy and robustness. The figure should be read from top to bottom, where each column illustrates the prediction sequence for a given image, and each row illustrates the steps (random sampling, three iterations of proposals, final suction location in the depth and RGB image) from the input image to the final proposal.

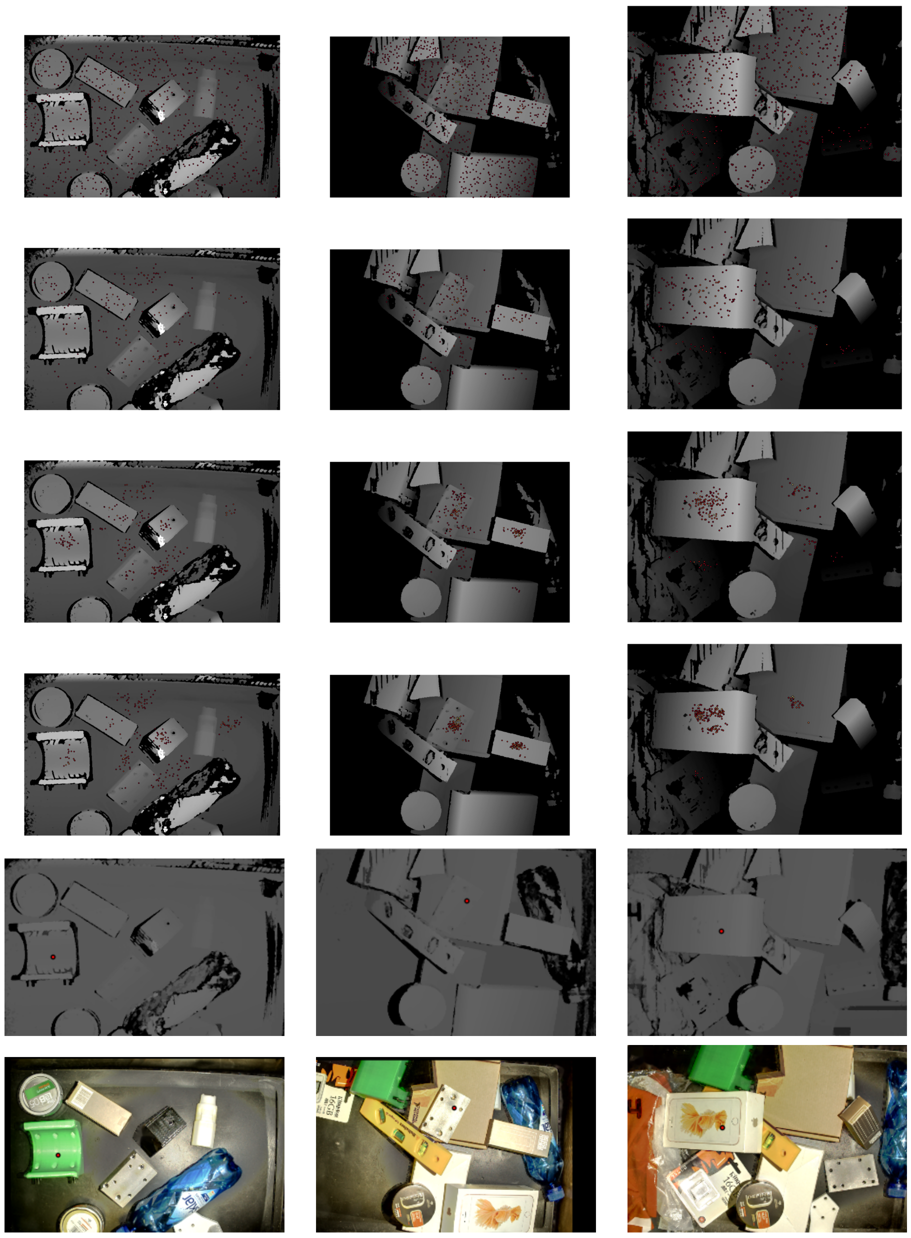
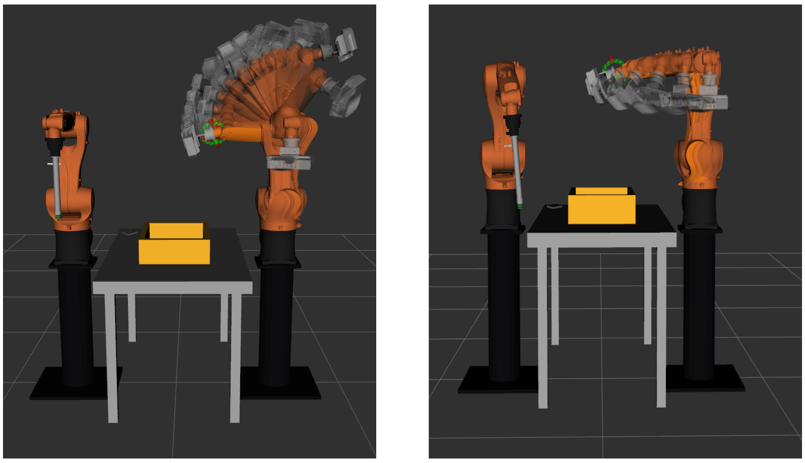


Figure 4.9: Illustration of the steps in the GQ CNN grasping module. Each column represents the steps for a given input image, the upper four rows shows grasping proposals, and the two lower rows shows the final proposal in the depth and RGB image.

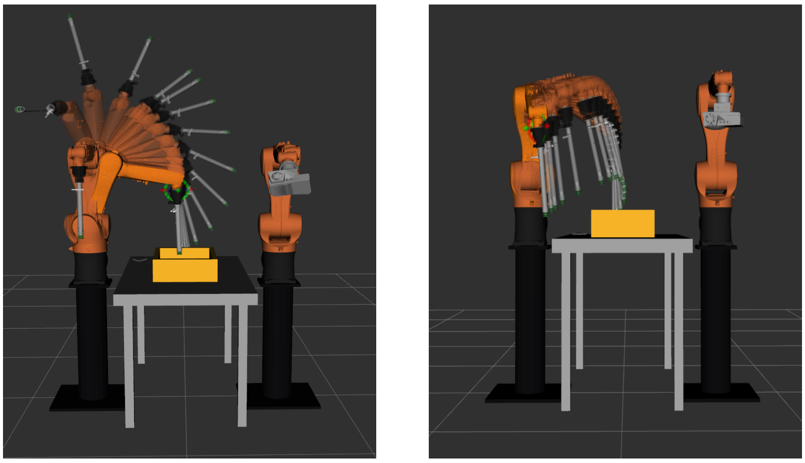
4.2.4 Robot Control

The modified axis range configuration from Section 3.2.4 resulted in improved end-effector trajectories, in terms of movement control during the transition between robot poses. Movement between the the robot pose **Default Start Position** and **Image_position** for the agilus2 move group is illustrated in Figure 4.10, before and after the modified configuration. The new configuration ensured that the power supply cable of the Zivid camera and USB cable never were tangled or stretched. Likewise, Figure 4.11 illustrates how Agilus1 moves from the **Suction_Location** to **Move_to_New_Bin**. The new configuration ensured that the external forces on the suction cup was reduced during transportation.


(a) The original inverse kinematic movement performed by Agilus2.

(b) The result of modification to the axis range.

Figure 4.10: Illustration of the movement performed by the agilus2 move group while moving from robot pose **Default Start Position** to **Image_position**.



(a) The original inverse kinematic movement performed by Agilus1.

(b) The result of modification to the axis range.

Figure 4.11: Illustration of the movement performed by agilus1 move group while moving from robot pose **Suction_Location** to **Move_to_New_Bin**.

4.3 Complete System

Following the state machine implementation described in Section 3.3, we combined the separate modules of the system to create a complete autonomous bin picking system. The sequence and transition between states were controlled by a Python script, where the only user input was whether to pick specific objects using the Faster R-CNN network, or to pick objects based on the most probable suction location. The Faster R-CNN network was preferred above the SSD network based on the S_2 score, which was significantly higher. The S_2 is more relevant than the S_1 in our bin picking system, as this score generally gives a better indication if a given object is present in the image. A series of tests were conducted in order to evaluate the robustness of the system as a whole.

4.3.1 Bin Picking Without Object Detection

In the first test (Test A), we wanted to evaluate the robustness of the system while picking novel objects without object detection. In this test, grasping was perform on the object that was best suited within the entire bin, based on the outcome of the grasping module. A total of 12 objects were placed at random locations in the bin before the test was started, as can be seen in Figure 4.12. The test was repeated three times to calculate the average success score. The results can be seen in Table 4.3, and illustrations of each object can be found in Appendix A.1.

Bin Picking: Test A							
		Test 1		Test 2		Test 3	
Item Nr	Item	Prob	S/F	Prob	S/F	Prob	S/F
1	Iphone	0.68	1	0.66	1	0.16	1
2	SD Card	0.89	1	0.89	1	0.49	1
3	Hair Wax	0.41	1	0.95	1	0.36	1
4	3D Printed Object1	0.58	1	0.21	0	0.25	0
5	3D Printed Object2	0.215	0	0.25	1	0.30	1
6	3D Printed Object3	0.32	1	0.50	1	0.42	1
7	3D Printed Object4	0.13	0	0.69	0	0.20	0
8	Small Box	0.73	1	0.32	1	0.65	1
9	Large Box	0.03	1	0.08	0	0.23	0
10	Tape	0.51	1	0.11	0	0.23	1
11	Metal Object	0.20	1	0.28	1	0.76	1
12	Bottle	0.22	0	0.18	1	0.49	1
Score		0.75		0.67		0.75	
Average Score		0.72					

Table 4.3: Test A: Bin picking without object detection. The *Prob* column lists the estimated probability of success outputed from GQ-CNN, and *S/F* is short for Success/Failure during the picking operation.

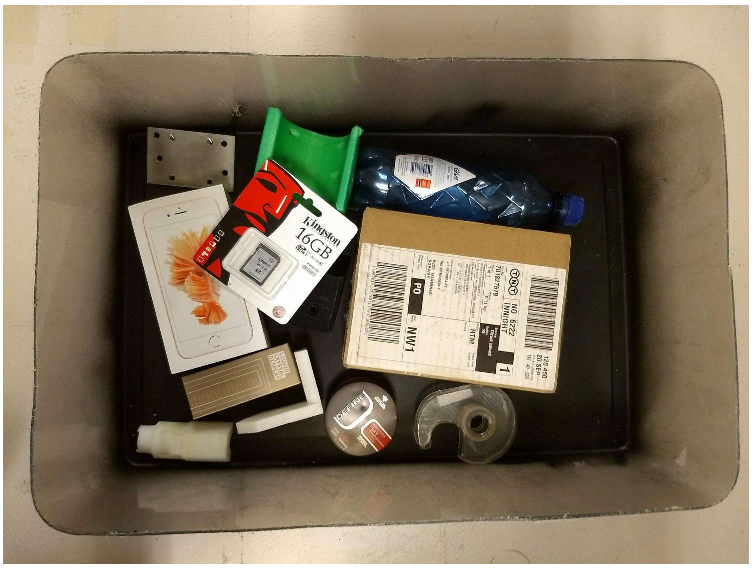
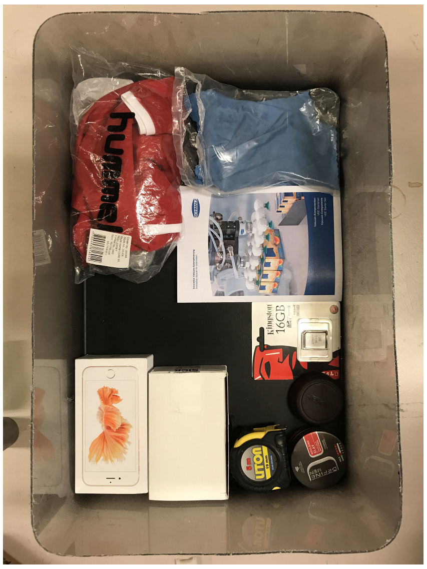


Figure 4.12: The objects and the setup used for one of the tests in test A.

4.3.2 Bin Picking With Object Detection

In the second test (Test B), we wanted to pick up specific objects from the bin, using object detection to recognise and locate the objects. Our four test objects were put in a bin a long with 8 novel objects that were not used during the training phase of the object detection network. The novel objects all show some resemblance in features and characteristics to the trained objects, so that the object detection network had a challenging task at hand. The four test objects and the novel objects can be seen in Figure 4.13, along with the final setup of the test. The test was repeated three times to calculate an average score, and the results can be seen in Table 4.4.


(a) The trained objects (Iphone box, SD card, t-shirt and hair wax) along with similar novel objects.

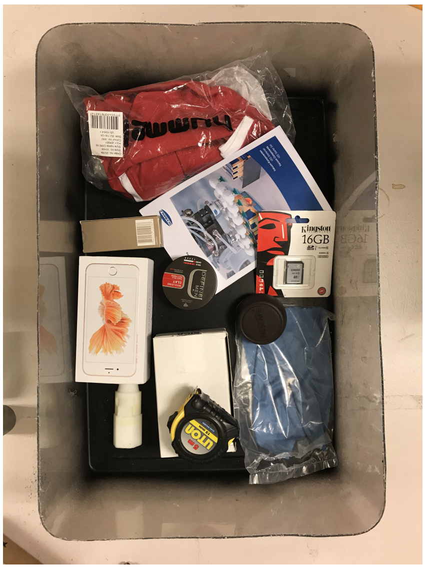

(b) The final setup of the objects in the bin.

Figure 4.13: Test B: Bin picking specific objects using deep neural network for object detection and grasp planning.

Bin Picking With Object Detection: Test B						
	Test 1		Test 2		Test 3	
Item	OD/GQ	S/F	OD/GQ	S/F	OD/GQ	S/F
Iphone	0.97/0.76	1	0.98/0.45	1	0.96/0.98	1
SD Card	0.96/0.645	1	0.90/0.54	1	0.97/0.82	1
Hair Wax	0.97/0.87	1	0.93/0.71	1	0.85/0.002	1
T-shirt	0.95/0.76	0	0.97/0.47	1	0.95/0.15	0
Score	0.75		1.00		0.75	
Average Score	0.83					

Table 4.4: Test B: Bin picking without object detection. The *OD/GQ* column lists the predicted object detection score, and the probability of success from GQ-CNN. The *S/F* column lists the Success/Failure for each operation.

Demonstration Video

A demonstration video was produced to show the performance of the system during random bin picking with object detection. The demonstration also illustrates how we automated the labeling process of the images for the training phase of the object detection networks, and our development of the custom made gripper module. The video can be found in the digital appendix under the name "DemonstrationVideoMaster.mp4", and is also available online [85].

Chapter 5

Discussion

The following chapter will present discussions based on the obtained results from our practical work. Strengths and weaknesses of the system will be covered, and certain improvements are suggested.

5.1 Object Detection

After testing the two object detection networks with multiple images, it became evident that both the Faster R-CNN and SDD network had a tendency to perform correct classification more often on certain objects. From Table 4.1 and Table 4.2, we observed that both networks scored the highest number of TruePositive for the IPhone object, whereas the hair wax scored the worst. In particular, both networks achieved low scores when trying to detect the hair wax when it was positioned on top of another object. On the contrary, we observed that the IPhone was detected with a relatively high accuracy, despite experiencing occlusion or placement on top of other objects. The geometry of the iPhone and the hair wax can explain why the iPhone generally has a higher detection rate, and is related to the form of the ground truth bounding box label. As a result of the iPhone's rectangular shape, the labeled bonding box generally fits better around the object, whereas the bounding box for the circular Hair Wax will have a large portion of the background in every labeled training image. In addition, our training data had little variation in the background of the scene, which led the network to weight the black background within the labeled rectangle too high. Placing the object on top of another object, made the network unable to recognise the object, since the dark uniform background in the bounding box was not present. When changing the background of the input images to resemble the black background of the training set, we observed that the accuracy of the Faster R-CNN and the SSD networks improved. This is illustrated in Figure 5.1.

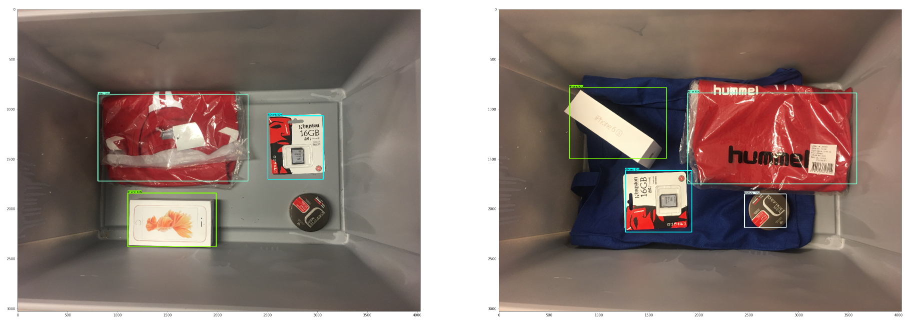


Figure 5.1: Illustration of the improvement in accuracy for the SSD network, as the background is changed to one which is resembles the black background of the training set.

5.1.1 Improved Autonomous Labeling

We suggest a solution to the explained issue related to the lack of variation in the background of the training set, by improving the automatic labeling program from Section 3.2.2.

The labeling program could be improved by automatically replacing the uniform black background with a texture image. This could by done by obtaining a segmentation mask of the binary object representation, and cropping out the background in order to replace it with a new image. The original, binary, and texture images are possessed beforehand, so that they all have the same resolution. This way, the placement of the cropped object and bounding box are in the same position in all cases. The results can be seen in Figure 5.2, where the variation in the background is clear. The cropping process is not perfect, but it demonstrates the concept of the improved automatic labeling system. If we were to train a new network using this technique, we would have to retake all the images so that the background of the object would be completely black. This would improve the threshold value when converting to binary representation, and therefore improve the cropping at the edges of the objects. As can be observed in Figure 5.2, most of the objects are only partly cropped as a result of too low threshold for the specific image.

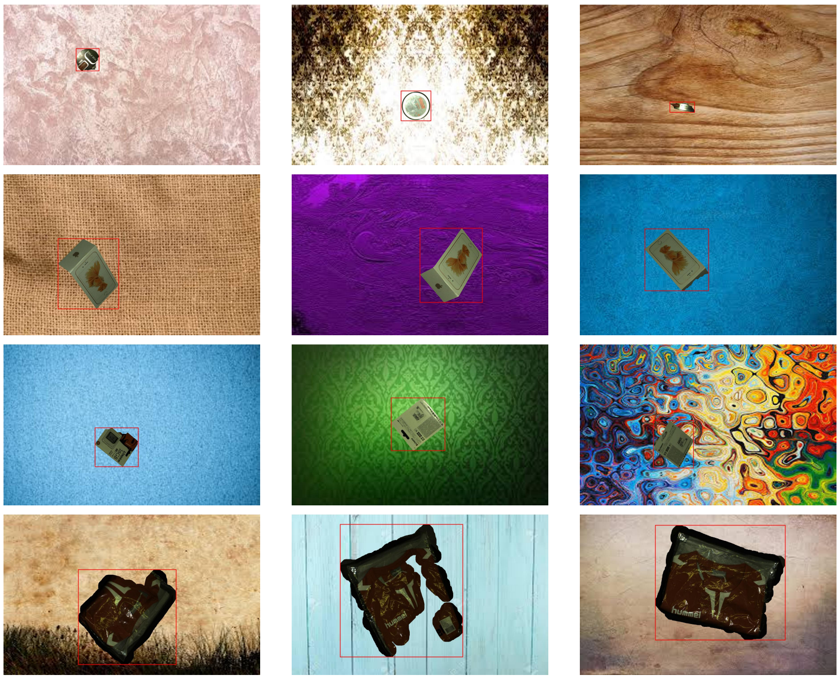


Figure 5.2: Illustration of the labeled output images with the improved version of our automatic labeling program. The variation in the background should theoretically decrease the weighting of the black background as an attribute of the objects.

5.2 Grasping

The grasping module generally located logical and intuitive suction locations based on the input depth map, but certain factors effected success and execution of the proposed location.

Depth Image Quality

A high quality depth image is essential for the GQ-CNN, in order to locate to assumed best suction location. As explained in Section 3.2.1, the initial depth image undergoes a series of feature enhancement and noise removal operations. The improvement in the processed image is evident, however it proved hard to obtain a high quality depth image for all object surfaces, textures, and angles. As can be observed in Figure 5.3, the noise in the depth readings of the image forces the GQ-CNN to exclude the best suited suction locations on the object. As a consequence, the execution of the picking operation may result in failure due to the torque that occurs during lifting.

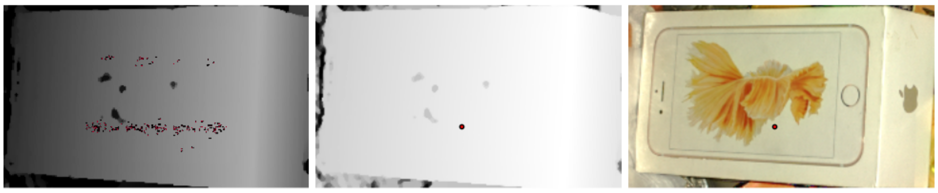


Figure 5.3: The grasping module avoids the best grasping locations around the center line of the object, due to the noise that is present in that area.

This issue can partly be resolved by applying more smoothing filters to even out and remove more noise in the depth image, but at the expense of reduced texture and edge details. The texture and edge details are crucial for determining whether or not the suction cup is able to establish a tight seal on the surface, and should therefore be prioritised over noise removal by smooting filters.

Object Segmentation

The GQ-CNN network generally performed well on semi structured bin picking operations, where the objects are clearly separated. A higher failure rate was observed when performing bin picking without object detection, when two or more objects were stacked side-by-side. This often caused the GQ-CNN network to evaluate the side-by-side objects as one solid object. The final suction location was therefore located close to the center of the combined objects, which in reality might be close to the edge of the objects. This type of failure may be caused by the fact that the network is only trained on single object depth images. The scenario is illustrated in Figure 5.4, where execution of the pick results in failure due to the external forces that occur. This issue is however greatly reduced when object detection is performed during the picking operation, as the detection network allows the system to crop out specific areas of the depth image where the specific object is located.

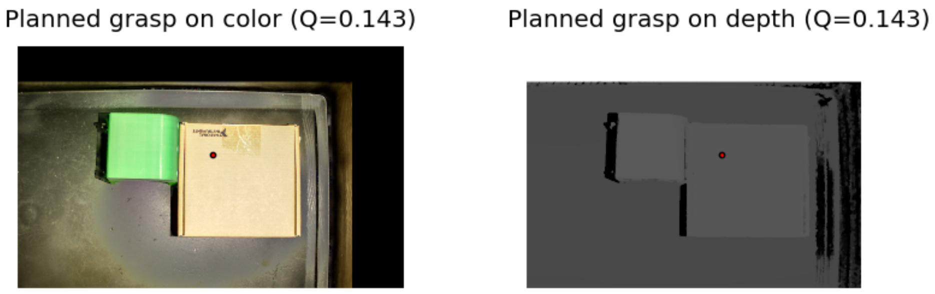

Planned grasp on color (Q=0.143)
Planned grasp on depth (Q=0.143)

Figure 5.4: The GQ-CNN evaluates two objects stacked side-by-side as one solid object. The final suction location is therefore given as if the two object were one, and the execution of the pick may result in failure.

In order to solve the issue described above, we propose a separate object segmentation module in the bin picking pipeline that has the potential to resolve the issue where several object are evaluated as one. Figure 5.5 illustrates the result of a proposed object segmentation module, that could be implemented in our system to increase the success rate. The module is based on the work presented in [93], and uses a modified Canny edge detector to extract robust edges in the image. The result from testing the grasping module using this segmentation method is shown in Figure 5.6, where a clear improvement can be observed compered to initial grasping proposal in Figure 5.4. The proposed module gives a good demonstration of the improvement, but it is limited to simple geometric shapes as it performers poorly on more complex object such as the t-shirt. A more robust segmentation algorithm is therefore necessary in order to be successfully applied in a realistic setting. A deep neural network might be used for the purpose of complex segmentation, as demonstrated in [94] and for uniform bins in [95].

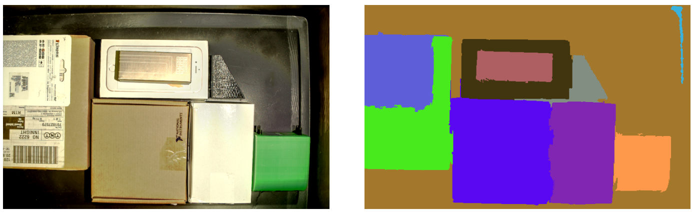
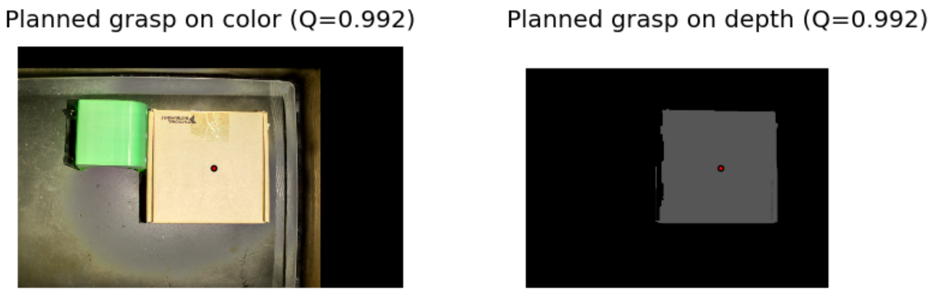
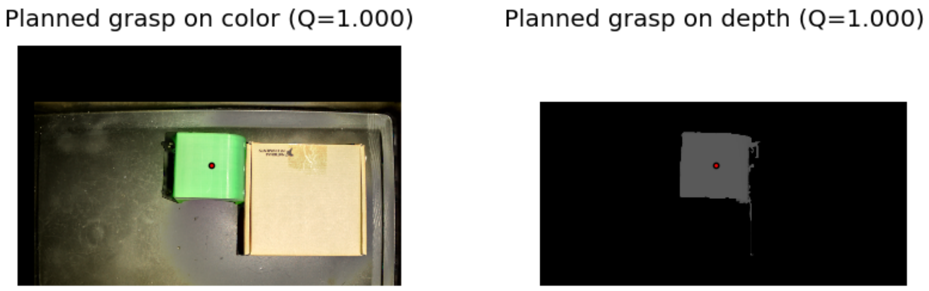


Figure 5.5: Illustration of a possible object segmentation algorithm that can be used to separate the objects in the scene.


Planned grasp on color (Q=0.992)
Planned grasp on depth (Q=0.992)

(a) Final grasping location when using object segmentation to crop out the large box.


Planned grasp on color (Q=1.000)
Planned grasp on depth (Q=1.000)

(b) Final grasping location when using object segmentation to crop out the green object.

Figure 5.6: Illustration of the same situation as in Figure 5.4, but here segmentation is applied to separate the two objects. The resulting grasping location is located at a more desirable location for both objects.

We also observed that segmentation module affected the estimated probability of grasping success given by the grasping module, which can be explained as result of the segmented image resembles the images used during the training phase of the network. The output probabilities from our tests (Test A and Test B) are generally misleading, due to the same reason related to the difference between the training images and the input image.

Object Occlusion

Bin picking with object detection may partly resolve the issue of object segmentation, as described above, but it may also introduce a new issue. Since the location of the object within the image is represented with a single rectangular bounding box, there are no assurances that the bounding box only contains the desired object. In cases when an object is occluded the object simply does not fill the whole bounding box area, the bounding box may contain several objects. The current system have no method of separating these objects from each other. Furthermore, the policy of the grasping module is to favour grasp locations close to the camera, in order to prevent picking objects that are occluded by other objects. Therefore, the objects laying on top of the intended object are more likely to be grasped. This is illustrated in Figure 5.7, where the bounding box proposal of the Iphone also includes a smaller object laying on top. As a result, the final suction location is placed on another object then intended, and the systems fails to pick the correct object.

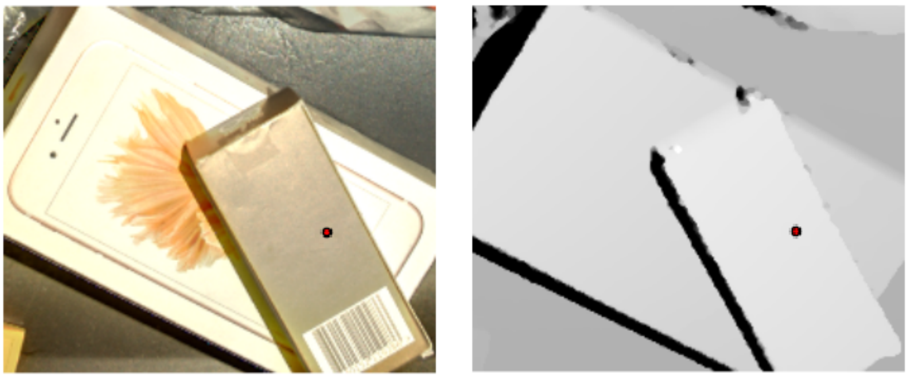


Figure 5.7: The figure illustrates how the large bounding box proposal of the Iphone also includes a separate object that is more suited for grasping. As a result, the final suction location is located on an unintended object.

This issue may be resolved by replacing the current object detection networks with a segmentation mask network, such as the *Mask R-CNN* network [96]. This deep learning network was introduced in January of 2018, and is an extension of the Faster R-CNN used in our practical work, that predicts an object mask in parallel with bounding box proposals. This allows object segmentation to be predicted at a much higher accuracy, since the proposals can have an arbitrary form, as illustrated in Figure 5.8. Training this network would however require a dataset with segmentation labeling, which is harder to require then the rectangular bounding box notation. A more suitable approach would be combining object detection with a novel segmentation method.

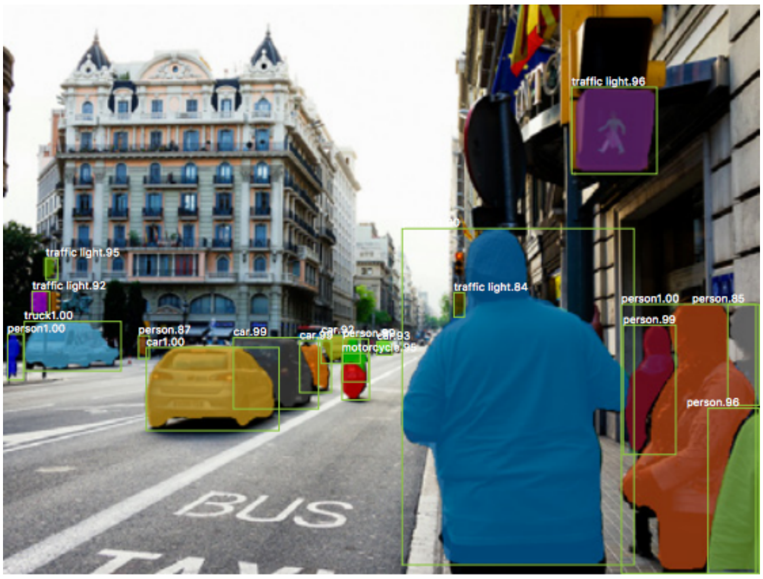


Figure 5.8: The Mask R-CNN network outputs object masks in parallel with bounding box proposals. The mask boundaries of the detected objects are much more accurate than the Faster R-CNN and the SDD networks used in our practical work.

5.3 Suction Cup

The choice of the suction cup also had a large impact on the number of failed attempts during the test, and was mainly due to two reasons:

- **Diameter Size**
  The diameter of the suction cup limits the areas on an object where a successful suction can be performed. A small diameter allows us to pick objects with a small region suitable for forming seal, and is therefore generally more suited for small objects. The small diameter will however struggle while picking large objects, as the smaller cup has lower lifting capacity compared to a larger sized suction cup. A small suction cup is also more prone for external forces during lifting. On the contrary, a large diameter will handle these forces better, but will struggle to create a seal between the object surface and the suction cup on small objects.

- **Material and Design**
  The material properties and design of the suction cup affects the performance of the picking operation. The flexibility of the material and the design determines how well the suction cup performs on different surfaces. A soft suction cup is able to obtain a thigh seal on a non planner continuously differential surfaces, due its the ability to deform accordingly to the

surface. A flexible and long suction cup is able to tilt in order to obtain seal, and works well when the approach angle deviates from the surface normal of the suction location. The flexibility and soft material comes with the expense of the ability to resist wrenches during lifting, which reduces the lifting capability in terms of weight and force tolerance.

In a realistic bin picking operation for a warehouse, these issues could be reduced by grouping similar objects to different picking stations that are customised for the product range characteristics. Small objects would generally require a small and flexible suction cup, while larger objects would require one or several larger suction cups to perform the picking operation. A combination of suction grippers and parallel grippers could also be considered for certain objects, to improve the robustness during transportation.

5.4 Bin Picking

The results of the bin picking operations presented in Section 4.3 clearly proves that our proposed system was able to pick random and specific objects from a cluttered environment. We observed that many of the failed attempts were related to the training dataset that was used during training of the grasping network. This dataset only contains grasp proposals for scenes with a single object, which often resulted in poor suction locations proposals when objects were cramped together in the bin. An improvement for this issue would be to extend the dataset to include training grasps with multiple objects in the scene, or utilising object segmentation as suggest earlier. An improved success rate could be achieved by introducing some form of quality control of the picking operation, to ensure that the correct object was pick and transported as planned.

Quality Control

In a realistic setting, it would be natural to perform some sort of quality control of the picking operation to ensure that the correct object was picked and that the transportation was successful. Possible control methods might include:

- **Object Detection**
  A new round of object detection might be performed while the intended object is transported from the storage bin. The image used for this object detection should be taken from an angle that limits the view to only the picked object, and not the rest of the bin. The output from this object detection will indicate whether or not the pick was successful. Bar code scanning might also be a possibility, as it is fast and able to differentiate products that are nearly identical (such as an Iphone 6 box and an Iphone 6S box).

- **Vacuum Measurement**
  A vacuum sensor might be mounted between the vacuum generator and

the suction cup, in order to continuously read the vacuum value during the picking operation. A sudden drop in the vacuum for this time interval would indicate that the object was dropped during transportation.

- **Torque Measurement**
  Torque sensors could be used at the end-effector of the robotic arm, so that measurements before and during the picking operation could be compared. No difference in torque would indicate that the pick was a failed attempt, and a sudden drop in difference during transportation indicates that the object was dropped. The torque sensor could also be used to measure the weight of the picked object, in order to compare the value with the known object weight. This quality control presupposes that the weight of each object inside a bin is differentiable. This strategy has the advantage to of being simple to implement in a large scale, since the weight of each product is already available in most settings.

- **Grasp Probability Control**
  In situations where the grasping module proposes a final grasp location with a corresponding success probability under a given threshold, the system should notice a manual worker for guidance. The worker would be able to correct or guide the robot to a more suited grasping location, to ensure that the pick does not result in failure. This functionality is implemented and demonstrated in the bin picking systems provided by *SuperPick* [97]. The same approach could be implemented for the object detection module, if the classification is under a given threshold.

5.5 Bin Picking In The Industry

AutoStore AS put us in contact with one of their many costumers, Komplett, which is Norway's larges e-commerce store. Through conversations with director Pål Vindegg, we got insight to the requirements needed for a potential automated bin picking system. On average Komplett receives between 2 000 and 3 000 new products every month, with a total of 20 000 to 30 000 products in their inventory. Following is an evaluation of the feasibility of implementing a similar bin picking system as we have proposed in this thesis in terms of object detection and grasping.

5.5.1 Object Detection

In the case of Komplett, an inventory size of 20 000 to 30 000 products would require an immense amount of labeled training images to train networks for object detection, as well as continuously updating the networks as new items are removed or added to the product list. The issue of acquiring object specific training datasets has traditionally been one of two drawbacks related to deep learning, where the other being the necessary computation power needed. A storage facility with the rate of product change and size in product inventory as Komplett would require a dataset lager in size than the largest available public datasets today, which for comparison is the ImageNet dataset with 14 million images distributed over 20 000 categories. The computation issue is a problematic aspect due to the tremendous amount of computing power required to continuously retrain the network in order to add new classes. This is however not a problem in regards to capability, but rather in terms of cost and time. Incrementally adding new classes and retraining a previous trained network introduces new issues in regards to decreasing performance. Konstantin Shmelkov et. al proposes a solution to this issue by introducing a loss function that balances the interplay between predictions and the new classes [98], but the final results will still experience a lower success rate.

Object Segmentation

A object detection based bin picking system introduces several problematic issues, but it would not be necessary to implement this module in certain bin picking scenarios. At Komplett, all the different products are kept separate in individual bins, so there will not be situations where multiple different products are present within the same bin. In this scenario, individual object detection will not be necessary, since the bin only contains one particular product type. In an automated bin picking system it would therefore be sufficient to use computer vision as a tool to determine the grasping location for a robotic manipulator. An object segmentation network could be implemented in the pipeline, to segment the objects that are suited for grasping. The advantage with this approach, is that the object segmentation network does not typically require additional training as new products are added to the inventory. Assuming that most products are contained in some form of box etc., the variation in product geometry between new and present products will be limited, thus the segmentation success rate will not be noticeably affected. Several segmentation networks have been proposed and published, that yields satisfying results [99], [100], [101].

5.5.2 Grasping

The deep learning based grasping module applied in the practical work of this thesis has demonstrated that it is capable of locating robust grasping locations under the right conditions. Through our research and experiments, we have found that many of the grasping algorithms that are being developed and published are capable of consistently locating robust grasping locations on novel objects. The Dex Net 3.0 publication achieved a 95% success rate when grasping simple novel objects [60] with a suction gripper, and Amaury Depierre et. al. achieved a 86.88% success rate when grasping novel objects with a parallel gripper [102]. The Google Brain Team achieved a success rate of 90% on novel objects while training a deep neural network from 900 000 grasp attempts during self-supervised learning, which was obtained from 14 robotic manipulators in parallel [103].

The advancements and success rates demonstrated by these state-of-the-art grasping algorithms shows the promising potential for a fully implemented autonomous bin picking system. If the quality control proposals from Section 5.4 are implemented in the bin picking system, the success rate would increase further and make the overall system more robust.

Chapter 6

Concluding Remarks and Future Work

6.1 Conclusion

The practical purpose of this master thesis was to create an autonomous bin picking system, capable of recognising and picking specific objects from a cluttered environment. Through an extensive literature research, we got a clear understanding of the possible technological approaches that could be used for the task at hand. The problem formulation from Section 1.2 was concretised into six objectives that have been successfully achieved through our practical work. First, the kinematics of the robot cell was described, and the eye-in-hand coordination from the camera to the robots were estimated with and mean deviation of 0.0032 m in translation and 0.0045 rad in orientation. Secondly, a gripper was designed and manufactured according to the system requirements, in order to be complaint with AutoStore's ASRS bins.

Next, deep neural networks were chosen as the preferred method for object detection and grasping, as these methods have shown significant increased performance since their introduction on their respective fields. This tendency was confirmed by comparing benchmarks, and through the reports from previous Amazon Picking Challenges. From the results, it became evident that our object detection and grasping module were able to detect and pick a large variety of objects in a realistic setting. The results also substantiates the robustness generally displayed by deep learning approaches. The creation of a sufficient training dataset for object detection was a challenge, but it was resolved to some extent by automating the labeling process, which yielded satisfying results. The SSD and Faster R-CNN object detection networks both provided adequate stability in terms of classification, where the latter achieved a higher S_1 and S_2 score (0.61>0.60 and 0.81>0.64). The use of the rectangular bounding box notation, combined with the lack of variation in the scene of the training dataset, were identified as the main sources of inaccuracy. On the basis of this finding, we proposed an improved automatic labeling pipeline that included background manipulation in order to increase the variation in the scene.

Finally, the individual modules were combined to a single autonomous bin picking system, and was evaluated and discussed in terms of robustness and feasibility of implementation in a realistic setting. The overall results of the proposed bin picking system yielded promising results, as it achieved a success rate of 0.72 while picking unspecified objects, and a success rate of 0.83 while using the object detection module. The quality of the acquired depth image of the scene had a large impact on the success rate of the proposed system. The grasping module was very sensitive to small amounts of noise in the image, which resulted in grasping locations that were not particular suited for grasping. The GQ-CNN also struggled to distinguish and separate objects that were stacked side-by-side, as the network often evaluated these objects as a single object. However, a possible solution to this issue was demonstrated by the introduction of an object segmentation module in the pipeline.

Our work presented in this thesis substantiates that bin picking could be implemented in a realistic warehouse setting in the near future, given that the object detection module in our system is replaced with an object segmentation module. The object segmentation module could easily be scaled up as new products are added to the inventory, while our object detection module would require new training data and retraining.

The proposed work presented in this thesis fulfills the described problem formulation, but certain improvements could be implemented to further increase the success rate and the robustness of the system.

6.2 Future Work

The practical work and experiments conducted in this master thesis demonstrated how deep learning, computer vision, and robotics can be combined to create an autonomous bin picking system. Each of the separate modules in our system have been thoroughly explained, and weaknesses and strengths have been identified. The following section will address some of the aspects of our system that we feel could be improved for future work.

- **Improved Training Dataset for Object Detection**
  As mentioned in Section 5.1, our object detection dataset has an limited amount of variation in terms of the object scenery and in the background. We proposed a solution to this issue in Section 5.1.1, where the black background of the scene is replaced with a random texture image. This method would be of interest to test, in order to compare the effect in accuracy of a more varied dataset.

- **Segmentation Module**
  The problem of object segmentation, as described in Section 5.2, may be

resolved by introducing a separate object segmentation module in our proposed system pipeline. Introducing a such module could reduce the failure rate of the picking operation, as the system would be able to separate tightly stacked objects within the bin, and not evaluate them as one single object.

- **New Object Detection Network**
  The issue of object segmentation for grasping discussed in Section 5.2 may be resolved by replacing our Faster R-CNN network with a network capable of object detection with object mask proposals. This could increase the success rate of the bin picking operation by replacing the current rectangular bounding boxes for object proposal with a more accurate region proposal. Pre-trained neural networks for object detection, including the Mask R-CNN network, can be found at TensorFlow's GitHub repertory [84].

- **Dex Net 4.0**
  The suction location module of our proposed system is based on the Dex Net 3.0 contribution, which is currently the latest released iteration in the Dex Net project. Our experiments indicated that this network performed better when the object in the bin were separated, as opposed to when the objects were stacked tightly together. During our work, a new iteration in the Dex Net project (Dex Net 4.0) was announced to be released during the summer months of 2018. This release will introduce several improvements compared to its predecessor, such as dual grasping with suction and/or parallel-jaws, use of high resolution depth images, and an improved training dataset for multiple objects. Replacing the Dex Net 3.0 based module with a Dex Net 4.0 based module in our system will most likely improve the success rate of our experiments. News, releases, and updates regarding the Dex Net project are posed continuously on their home page [104].

References

[1] Retail e-commerce sales worldwide from 2014 to 2021 (in billion u.s. dollars). `https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/`. Accessed: 2018-04-14.

[2] E-commerce in the nordics 2017. `https://www.postnord.com/globalassets/global/english/document/publications/2017/e-commerce-in-the-nordics-2017.pdf`. Accessed: 2018-05-24.

[3] Capsen. `http://www.capsenrobotics.com/capsen-pic.html`. Accessed: 2018-05-24.

[4] Superpick. `https://www.softroboticsinc.com/superpick/`. Accessed: 2018-05-24.

[5] Righthand robotics. `https://www.righthandrobotics.com/`. Accessed: 2018-05-24.

[6] Peter Corke. *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*, volume 73 of *Springer Tracts in Advanced Robotics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[7] Structured-light 3d surface imaging: a tutorial. `https://www.osapublishing.org/aop/fulltext.cfm?uri=aop-3-2-128&id=211561`. Accessed: 2018-06-01.

[8] Jason Geng. Structured-light 3d surface imaging: atutorial. *Adv. Opt. Photon.*, 3(2):128–160, Jun 2011.

[9] Tyler Bell, Beiwen Li, and Song Zhang. *Structured Light Techniques and Applications*, pages 1–24. American Cancer Society, 2016.

[10] Zhan Song, Hualie Jiang, Haibo Lin, and Suming Tang. A high dynamic range structured light means for the 3d measurement of specular surface. *Optics and Lasers in Engineering*, 95:8 – 16, 2017.

[11] Frédo Durand and Julie Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. *ACM Trans. Graph.*, 21(3):257–266, July 2002.

[12] Eye-in-hand calibration. http://wiki.zividlabs.com/files/handeye/Skotheim_EyeInHand.pdf. Accessed: 2018-05-07.

[13] Parameterizations for reducing camera reprojection error for robot-world hand-eye calibration. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7353795. Accessed: 2018-05-05.

[14] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943.

[15] Heung-Il Suk. Chapter 1 - an introduction to neural networks and deep learning. In S. Kevin Zhou, , Hayit Greenspan, , and Dinggang Shen, editors, *Deep Learning for Medical Image Analysis*, pages 3 – 24. Academic Press, 2017.

[16] Illustration of a simple fully connected neural network. https://hackernoon.com/deep-learning-cnns-in-\tensorflow-with-gpus-cba6efe0acc2. Accessed: 2017-10-27.

[17] Illustartion of the sigmoid function. http://www.ai-junkie.com/ann/evolved/nnt5.html. Accessed: 2017-10-28.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[20] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[21] Illustration of convolution operation. https://datascience.stackexchange.com/questions/23183/why-convolutions-always-use-odd-numbers-as-filter-\size. Accessed: 2017-10-28.

[22] Illustartion of convolution operation. https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks. Accessed: 2017-10-28.

[23] Illustartion of convolution operation. `https://sites.google.com/site/zhangleuestc/tracking-with-randomized-convnets`. Accessed: 2017-10-28.

[24] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *CoRR*, abs/1206.5533, 2012.

[25] Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *CoRR*, abs/1702.05659, 2017.

[26] Illustration of different learning rates. `http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html`. Accessed: 2017-12-02.

[27] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, February 2012.

[28] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[29] Siddharth Krishna Kumar. On weight initialization in deep neural networks. *CoRR*, abs/1704.08863, 2017.

[30] Saiprasad Koturwar and Shabbir Merchant. Weight initialization of deep neural networks(dnns) using data statistics. *CoRR*, abs/1710.10570, 2017.

[31] Dropout: a simple way to prevent neural networks from overfitting. `https://www.semanticscholar.org/paper/Dropout%3A-a-simple-way-to-prevent-neural-networks-Srivastava\-Hinton/34f25a8704614163c4095b3ee2fc969b60de4698`. Accessed: 2018-06-01.

[32] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[33] H2o world-top 10 deep learning tips & tricks - arno candel. `https://www.slideshare.net/0xdata/h2o-world-top-10-deep-learning-tips-tricks-arno-candel`. Accessed: 2018-06-01.

[34] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[35] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[36] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1717–1724, June 2014.

[37] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, Oct 2010.

[38] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[39] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[40] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[41] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[42] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[43] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.

[44] Image of ssd architecture. `http://silverpond.com.au/img/blog/multiclass-details/architecture.png`. Accessed: 2017-12-12.

[45] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: Modelling, Planning and Control*. Springer Publishing Company, Incorporated, 1st edition, 2008.

[46] James Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. `https://www.astro.rug.nl/software/kapteyn-beta/_downloads/attitude.pdf`, 2006.

[47] Trajectory planning in robotics. `https://link.springer.com/content/pdf/10.1007%2Fs11786-012-0123-8.pdf`. Accessed: 2018-05-20.

[48] Alessandro Gasparetto, Paolo Boscariol, Albano Lanzutti, and Renato Vidoni. *Path Planning and Trajectory Planning Algorithms: A General Overview*, pages 3–27. Springer International Publishing, Cham, 2015.

[49] Nodes. `http://wiki.ros.org/Nodes`. Accessed: 2018-04-21.

[50] Messages. `http://wiki.ros.org/Messages`. Accessed: 2018-04-23.

[51] Services. `http://wiki.ros.org/Services`. Accessed: 2018-04-23.

[52] Kai-Tai Song, Cheng-Hei Wu, and Sin-Yi Jiang. Cad-based pose estimation design for random bin picking using a rgb-d camera. *Journal of Intelligent & Robotic Systems*, 87(3):455–470, Sep 2017.

[53] J. Pyo, J. Cho, S. Kang, and K. Kim. Precise pose estimation using landmark feature extraction and blob analysis for bin picking. In *2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pages 494–496, June 2017.

[54] Pat Marion, Peter R. Florence, Lucas Manuelli, and Russ Tedrake. A pipeline for generating ground truth labels for real RGBD data of cluttered scenes. *CoRR*, abs/1707.04796, 2017.

[55] Andy Zeng, Kuan-Ting Yu, Shuran Song, Daniel Suo, Ed Walker Jr., Alberto Rodriguez, and Jianxiong Xiao. Multi-view self-supervised deep learning for 6d pose estimation in the amazon picking challenge. *CoRR*, abs/1609.09475, 2016.

[56] M. Schwarz, A. Milan, C. Lenz, A. Muoz, A. S. Periyasamy, M. Schreiber, S. Schller, and S. Behnke. Nimbro picking: Versatile part handling for warehouse automation. pages 3032–3039, May 2017.

[57] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. *CoRR*, abs/1509.06825, 2015.

[58] Andy Zeng, Shuran Song, Kuan-Ting Yu, Elliott Donlon, Francois Robert Hogan, Maria Bauzá, Daolin Ma, Orion Taylor, Melody Liu, Eudald Romo, Nima Fazeli, Ferran Alet, Nikhil Chavan Dafle, Rachel Holladay, Isabella Morona, Prem Qu Nair, Druck Green, Ian Taylor, Weber Liu, Thomas A. Funkhouser, and Alberto Rodriguez. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. *CoRR*, abs/1710.01330, 2017.

[59] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *CoRR*, abs/1603.02199, 2016.

[60] Jeffrey Mahler, Matthew Matl, Xinyu Liu, Albert Li, David Gealy, and Ken Goldberg. Dex-net 3.0: Computing robust robot suction grasp targets in point clouds using a new analytic model and deep learning. *arXiv preprint arXiv:1709.06670*, 2017.

[61] G.J. Monkman. Robot grippers for use with fibrous materials. *The International Journal of Robotics Research*, 14(2):144–151, 1995.

[62] Giacomo Mantriota. Theoretical model of the grasp with vacuum gripper. *Mechanism and Machine Theory*, 42(1):2 – 17, 2007.

[63] N. Correll, K. E. Bekris, D. Berenson, O. Brock, A. Causo, K. Hauser, K. Okada, A. Rodriguez, J. M. Romano, and P. R. Wurman. Analysis and observations from the first amazon picking challenge. *IEEE Transactions on Automation Science and Engineering*, 15(1):172–188, Jan 2018.

[64] How does a vacuum generator work? `https://www.festo.com/wiki/en/Function_of_a_vacuum_generator`. Accessed: 2018-06-05.

[65] Gareth Monkman. An analysis of astrictive prehension. 16:1–10, 02 1997.

[66] Theoretical holding force of a suction cup. `https://www.schmalz.com/en/vacuum-knowledge/the-vacuum-system-and-its-components/system-design-calculation-example/theoretical-holding-force-of-a-suction-cup/`. Accessed: 2018-06-03.

[67] Ns-en standard (13155:2003+a2:2009). `https://www.standard.no/no/Nettbutikk/produktkatalogen/Produktpresentasjon/?ProductID=382830`. Accessed: 2018-05-25.

[68] S. Wade-McCue, N. Kelly-Boxall, M. McTaggart, Douglas Morrison, Adam W. Tow, J. Erskine, R. Grinover, A. Gurman, T. Hunn, D. Lee, Anton Milan, Trung Pham, G. Rallos, A. Razjigaev, T. Rowntree, R. Smith, K. Vijay, Zheyu Zhuang, Christopher F. Lehnert, Ian David Reid, Peter I. Corke, and Jürgen Leitner. Design of a multi-modal end-effector and grasping system: How integrated design helped win the amazon robotics challenge. *CoRR*, abs/1710.01439, 2017.

[69] Carlos Hernandez, Mukunda Bharatheesha, Wilson Ko, Hans Gaiser, Jethro Tan, Kanter van Deurzen, Maarten de Vries, Bas Van Mil, Jeff van Egmond, Ruben Burger, Mihai Morariu, Jihong Ju, Xander Gerrmann, Ronald Ensing, Jan van Frankenhuyzen, and Martijn Wisse. Team delft's robot winner of the amazon picking challenge 2016. *CoRR*, abs/1610.05514, 2016.

[70] Kr agilus. `https://www.kuka.com/en-de/products/robot-systems/industrial-robots/kr-agilus`. Accessed: 2018-06-04.

[71] Kuka agilus kr 6 r900 sixx rotation axis image. `https://www.eurobots.net/used-robot-agilus-kr-6-r900-sixx-en.html`. Accessed: 2018-06-01.

[72] The zivid camera. http://www.zividlabs.com/. Accessed: 2017-12-02.

[73] Frank C Park and Bryan J Martin. Robot sensor calibration: solving AX= XB on the Euclidean group. *IEEE Transactions on Robotics and Automation*, 10(5):717–721, 1994.

[74] Real-time, in situ intelligent video analytics: Harnessing the power of gpus for deep learning applications. https://www.dsiac.org/resources/journals/dsiac/winter-2017-volume-4-number-1/real-time-situ-intelligent-video-analytics. Accessed: 2018-06-01.

[75] Pascal voc benchmark. http://player.slideplayer.com/26/8688198/data/images/img46.png. Accessed: 2017-12-12.

[76] Kitti leaderboard. http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=2d. Accessed: 2018-05-10.

[77] Coco leaderboard. http://cocodataset.org/#detection-leaderboard. Accessed: 2018-05-10.

[78] The unreasonable popularity of tensorflow. http://deliprao.com/archives/168. Accessed: 2018-05-12.

[79] Will google own ai? https://whatsthebigdata.com/2017/03/07/8505/. Accessed: 2018-05-12.

[80] Tensorflow model zoo. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md. Accessed: 2017-10-04.

[81] Running distributed tensorflow on compute engine. https://cloud.google.com/solutions/running-distributed-tensorflow-on-compute-engine. Accessed: 2018-05-12.

[82] Tensorflow tutorials. https://www.tensorflow.org/tutorials/. Accessed: 2018-05-12.

[83] Tensorflow api documentation. https://www.tensorflow.org/api_docs/. Accessed: 2018-05-12.

[84] Tensorflow github repository. https://github.com/tensorflow. Accessed: 2018-06-03.

[85] Development of an automated bin picking system for cluttered environments. https://youtu.be/bWHmY9RrQKc. Accessed: 2018-06-04.

[86] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *CoRR*, abs/1703.09312, 2017.

[87] Berkeley autolab's gqcnn package. `https://github.com/BerkeleyAutomation/gqcnn`. Accessed: 2018-05-26.

[88] Ros concepts. `http://wiki.ros.org/ROS/Concepts`. Accessed: 2018-05-31.

[89] Ros tutorials. `http://wiki.ros.org/ROS/Tutorials`. Accessed: 2018-05-31.

[90] Ros api. `http://wiki.ros.org/APIs`. Accessed: 2018-05-31.

[91] Moveit! `https://moveit.ros.org/`. Accessed: 2018-05-05.

[92] The pascal visual object classes (voc) challenge. `http://host.robots.ox.ac.uk/pascal/VOC/pubs/everingham10.pdf`. Accessed: 2018-06-05.

[93] Giorgio Toscana and Stefano Rosa. Fast graph-based object segmentation for RGB-D images. *CoRR*, abs/1605.03746, 2016.

[94] X. Li, L. Zhao, L. Wei, M. H. Yang, F. Wu, Y. Zhuang, H. Ling, and J. Wang. Deepsaliency: Multi-task deep neural network model for salient object detection. *IEEE Transactions on Image Processing*, 25(8):3919–3930, Aug 2016.

[95] Zheng Wu, Ruiheng Chang, Jiaxu Ma, Cewu Lu, and Chi-Keung Tang. Annotation-free and one-shot learning for instance segmentation of homogeneous object clusters. *CoRR*, abs/1802.00383, 2018.

[96] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[97] Superpick. `http://info.softroboticsinc.com/binpicking-orderfulfillment-superpick`. Accessed: 2018-05-26.

[98] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. Incremental learning of object detectors without catastrophic forgetting. *CoRR*, abs/1708.06977, 2017.

[99] Mary J Bravo and Hany Farid. Recognizing and segmenting objects in clutter. *Vision Research*, 44(4):385–396, 2004.

[100] D. Rao, Q. V. Le, T. Phoka, M. Quigley, A. Sudsang, and A. Y. Ng. Grasping novel objects with depth segmentation. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2578–2585, Oct 2010.

[101] Max Schwarz and Sven Behnke. Data-efficient deep learning for rgb-d object perception in cluttered bin picking. 2017.

[102] Amaury Depierre, Emmanuel Dellandra, and Liming Chen. Jacquard: A large scale dataset for robotic grasp detection. March 2018.

[103] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, April 2018.

[104] Dex net github documentation. `https://berkeleyautomation.github.io/dex-net/`. Accessed: 2018-05-24.

Appendix A

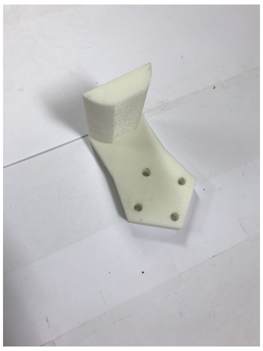
Appendix

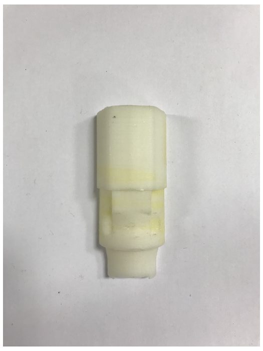
A.1 Test Objects

(a) Iphone

(b) HairWax

(c) SD-card


(d) 3D Printed Object4


(e) 3D Printed Object1

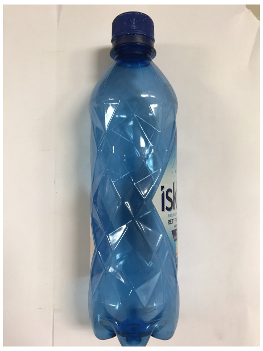
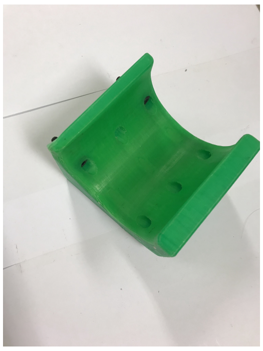

(f) Bottle

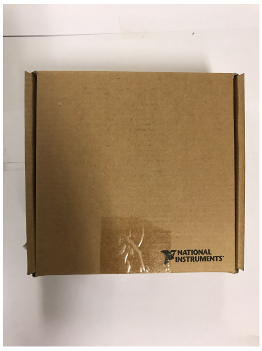
Figure A.1: Objects used in Test A.

(a) 3D Printed Object2


(b) 3D Printed Object3

(c) Tape

(d) Large Box



(e) Small Box

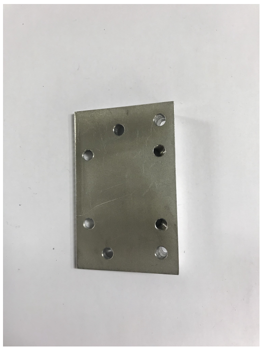

(f) Metal Object

Figure A.2: Objects used in Test A.

A.2 State Machine Details

Current State	Input	Next state	Output
Robots_not_connected	RT == True	Robot Home	RP = Home
Robots_Home	I == False GL == False RC == True	Perception	GL RP = Image
	RT == False	Robots_not_connected	System failure
	GL == True	Gripper_Ready	RP = ready GT
Perception	RT == False	Robots_not_connected	System failure
	GL = valid	Robot_Home	GL RP = Home
Gripper_Ready	RC & RT == False	Robots_not_connected	System failure

	GT == false	Robots_Home	GL = False
			RP = home

	GT = True	Grasping	V G
Grasping	V == true G == true	Placing	P V
	RT == False	Robots_not_connected	System failure

	V == Error
	G = Error
		Robots_Home	VG = false
			VG = false
			RP = Home

Placing	RT == False	Robots_not_connected	System failure
	V == false	Robots_Home	Clear inputs

Table A.1: The table shows all possible states and the transition between them, and the outputs resulting from each input. The first column lists the possible states of the machine, while second column lists the a set of possible of inputs at this state. The third column lists the next state in the system, and is dependant of the inputs of the current state. The last column shows the outputs from the current state.

Global State Variables	Abbreviation	Comments
Robot Connection	RC	master computers connection to the PLC
Camera Connection	CC	master computers connection with the camera
Robot Pose	RP	Current robot pose
Perception	I	Outcome of image acquisition and processing
Vacuum	V	Vacuum on off status
Grasp Location	GL	Status on grasp location computation
Grasp Trajectory	GT	Status on grasp trajectory computation
Grasp	G	Outcome of grasp execution
Place	P	Outcome of place execution

Table A.2: Explanation of inputs and outputs of the state machine.

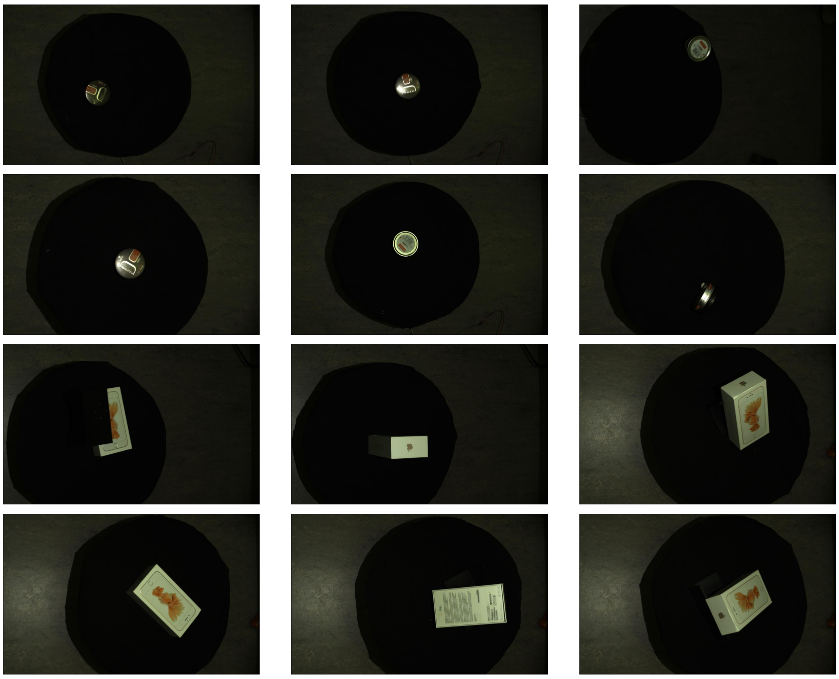


Figure A.3: Auto Labeled images for training

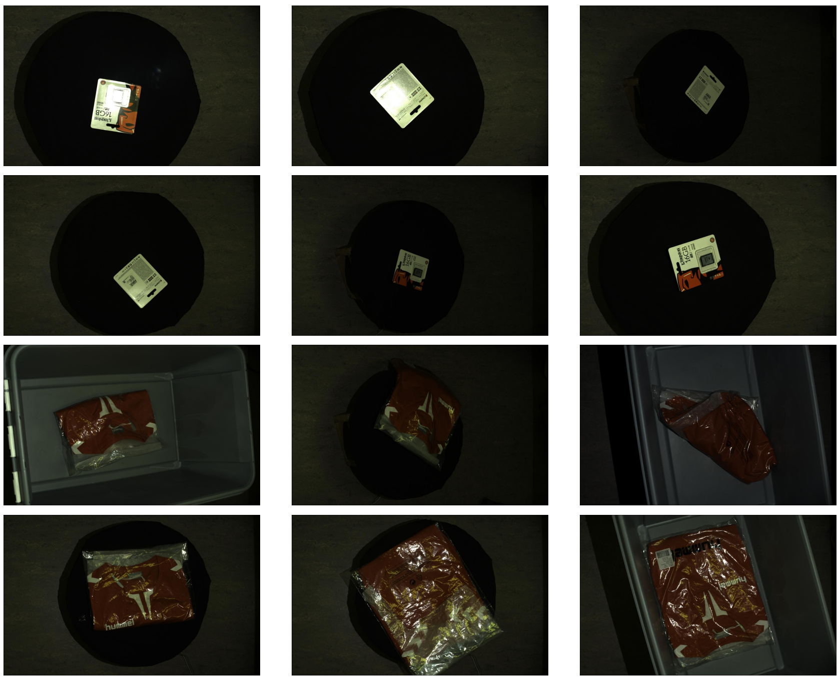


Figure A.4: Auto Labeled images for training

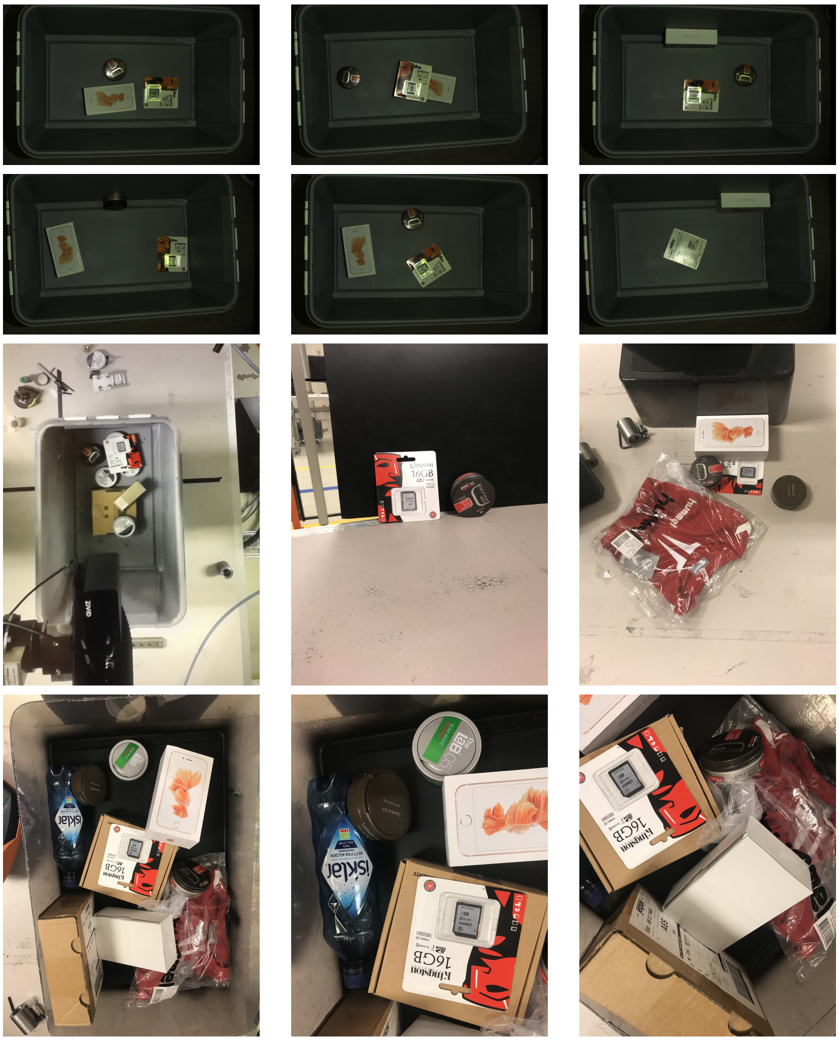


Figure A.5: Manual Labeled Training images

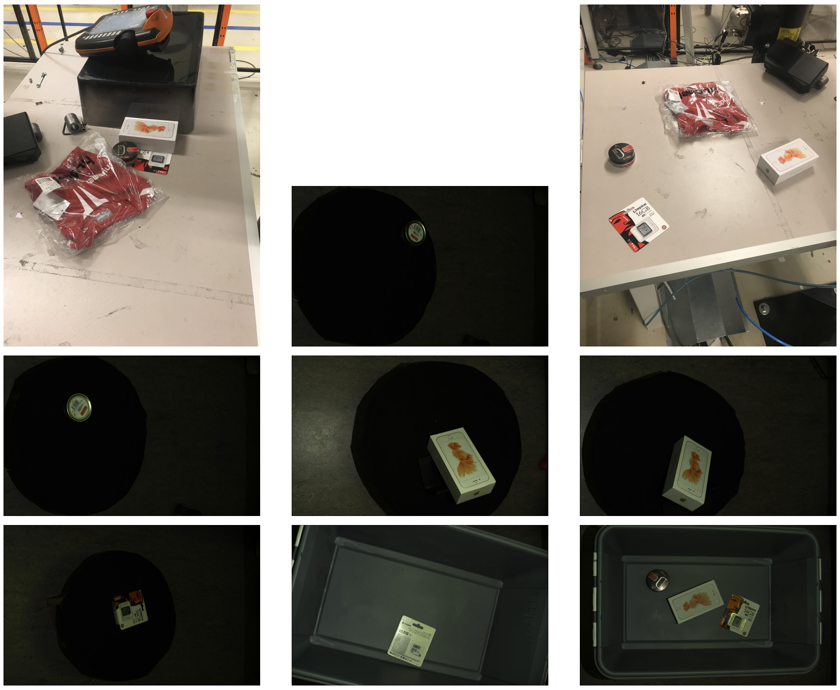


Figure A.6: Test and validation training dataset consists of 20 % of the manual and auto labeled images.

A.3 TensorBoard Training Graphs

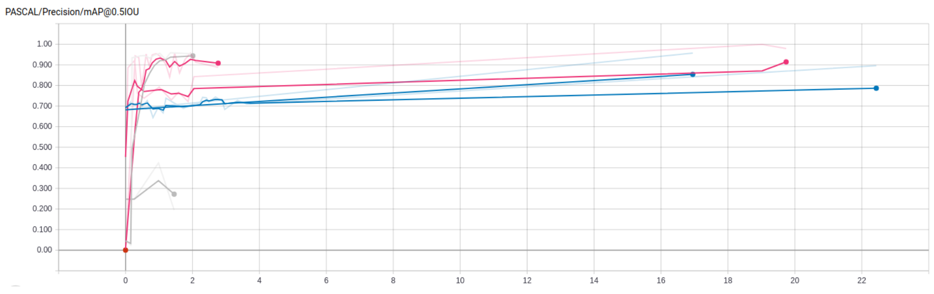


Figure A.7: Example of accuracy during training with different hyperparameters

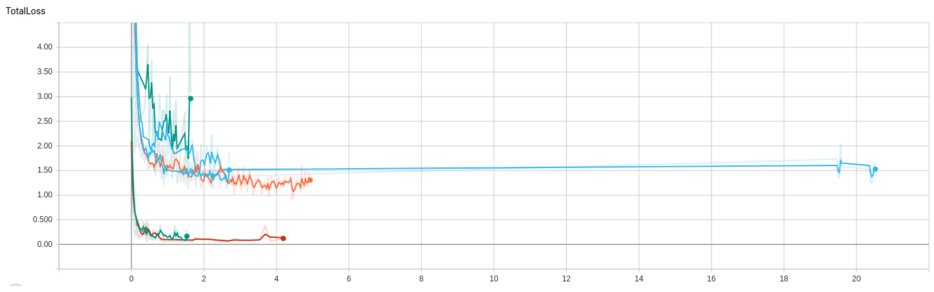


Figure A.8: Example of total loss during training with different hyperparameters

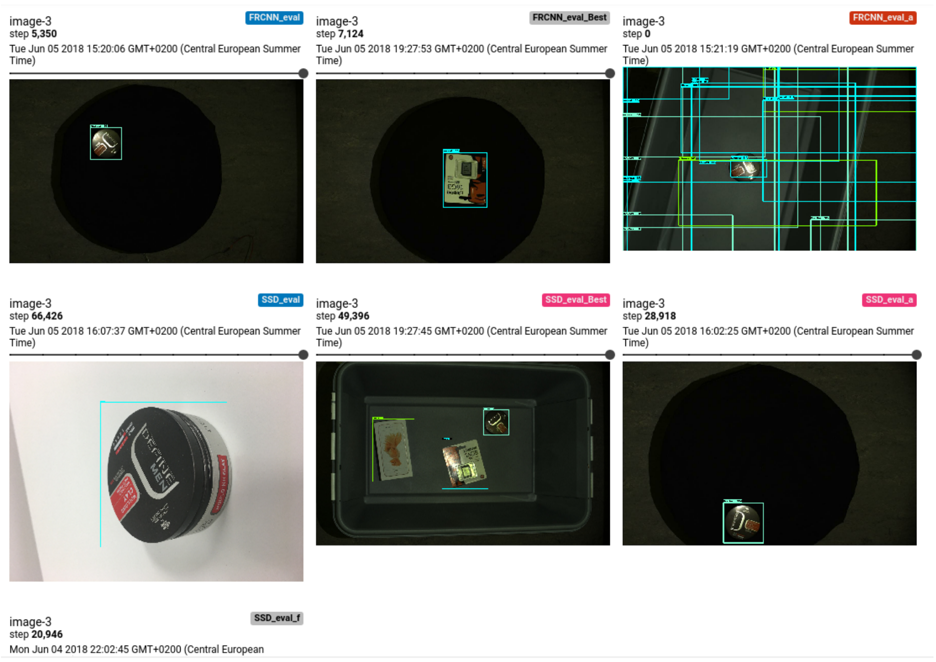


image-3 FRCNN_eval
step 5,350
Tue Jun 05 2018 15:20:06 GMT+0200 (Central European Summer Time)

image-3 FRCNN_eval_Best
step 7,124
Tue Jun 05 2018 19:27:53 GMT+0200 (Central European Summer Time)

image-3 FRCNN_eval_a
step 0
Tue Jun 05 2018 15:21:19 GMT+0200 (Central European Summer Time)

image-3 SSD_eval
step 66,426
Tue Jun 05 2018 16:07:37 GMT+0200 (Central European Summer Time)

image-3 SSD_eval_Best
step 49,396
Tue Jun 05 2018 19:27:45 GMT+0200 (Central European Summer Time)

image-3 SSD_eval_a
step 28,918
Tue Jun 05 2018 16:02:25 GMT+0200 (Central European Summer Time)

image-3 SSD_eval_f
step 20,946
Mon Jun 04 2018 22:02:45 GMT+0200 (Central European

Figure A.9: Example of predictions during training with different hyperparameters.

A.4 Digital Appendix

A digital appendix was submitted with the thesis, which contained the most relevant software:

- **DemonstrationVideoMaster.mp4**
  Demonstration video while bin picking with object detection.
- **Readme.txt**
  Readme file.
- **BinPick_main.py**
  Main code used while running the system.
- **BinPickCapture.cpp**
  Image acquisition for the Zivid camera.
- **GetRGBDim.py**
  Extraction of the relevant channels to create an RGB and depth image.
- **OD.py**
  Object detection module.