NTNU
Norwegian University of
Science and Technology

Tactile-sensitive robotic grasping of food compliant objects with deep learning as a learning policy

Alexander Martin Olofsson

Master of Science in Computer Science

Submission date:	July 2017
Supervisor:	Agnar Aamodt, IDI
Co-supervisor:	Ekrem Misimi, SINTEF Ocean

Norwegian University of Science and Technology
Department of Computer Science

Abstract

This thesis outlines work done in generating models used to control a robotic arm and tactile sensitive gripper for successfully grasping compliant food objects, using colour and depth images as input. The compliant food object used were salads, as they are both easy to work with and require delicate handling. The focus is to determine if it is possible to achieve significant results using just images as input, as well as comparing two different approaches for generating models.

Two different approaches were utilised when generating the models. The first model, Support Vector Regression, required features to be extracted from the images before being input into the model. The second model, Convolutional Neural Networks, accepts an image as input. Both models were taught how to respond to various salad positioning through learning-from-demonstration, using a dataset of successful grasps previously gathered. Once the models had been generated, they were compared by tasked with grasping a set of new salads.

Preface

This thesis is written as the final part of my master's degree in computer science at the Department of Computer Science at the Norwegian University of Science an Technology. It was carried out during the Spring of 2017 in collaboration with SINTEF Ocean as a continuation of work done during the previous Autumn.

I would like to thank my supervisor at NTNU, Agnar Aamodt, as well as my supervisor at SINTEf, Ekrem Misimi for all their help during the project. A special thanks to Aleksander Børresen Eilertsen, Elling Rud Øye and Jonatan Sjølund Dyrstad for their help when things did not want to work as expected, or when I needed a break. Their projects were varied and interesting, and having something else to think about for a few minutes was helpful before returning to my own work. I would also like to mention the MARO lab at SINTEF Ocean, where I spent many hours grasping, dropping and moving salad.

Alexander Martin Olofsson

Trondheim, 2017

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1. Motivation

In the summer of 2016, I had a summer internship at SINTEF Ocean, formerly SINTEF Fisheries and Aquaculture. During this internship, I was given the opportunity of combining machine learning, with robotics, machine vision and automation. Having previously only worked in simulated environments and the on software, it was an exciting new experience to be able to use some hardware.

iProcess Projcet: *WP3 Flexible Processing Automation*
This task for the summer was part of a work-package within the *iProcess project*, http://iprocessproject.com/.

> *iProcess's main objective is to develop novel concepts and methods for flexible and sustainable food processing in Norway – that are to cope with small volume series and high biological variation of the existing raw materials – to enable increased raw material utilisation for food products and to increase profitability.*[14]

The work-package I was to be apart of was called *WP3 Flexible Processing Automation*. With the focus being on *3D vision for localisation, recognition of raw material and dense visual servoing based on depth maps and 3D point clouds.* Along with another student, I was set the task of building a system that would combine a robotic arm and manipulator, a RGB-D camera, and a teleoperation controller. This system would then be used to grasp objects through teleoperation, recording each successful grasp as a combination of images and robotic sensory data. These examples would then be use to generate machine learning models, where vision data from the RGB-D images would be used to make grasp predictions.

This work was continued upon during the autumn, in a project assignment. During this time alterations were made to the camera placement in order to improve upon camera-robot object-in-hand calibration accuracy. This meant that a new dataset had to be generated, as well as prediction models. This Master thesis is an even further continuation on this work, focusing on generating prediction models that are to be used with the aforementioned system.

1.2. Hypotheses

Hypothesis 1 *Given data gathered from a RGB-Depth (RGB-D) image it is possible to estimate a 6-degree of freedom (DOF) gripper pose for grasping.*

In Hypothesis 1 the aim is to prove that using RGB-D images, it is possible to estimate a 6-DOF grasp. Various features are therefore extracted from the images, and used as input in a machine learning model. This model will then output an estimated 6-DOF. Once this has been estimated, and the gripper re-positioned, it should be possible to activate the gripper, closing it around the object. Once the object is securely held in place, it should be possible to move the gripper to another location, while maintaining a secure grip on the object, releasing the object once it has arrived.

Hypothesis 2 *A 6-degree of freedom (DOF) gripper estimation is more accurate with RGB-Depth (RGB-D) images as input to a Convolutional Neural Network (CNN) architecture then a Support Vector Regression (SVR).*

In Hypothesis 2 we attempt to prove that it is possible to achieve greater accuracy when using images as an input to a CNN, as opposed to first extracting features before feeding them through a SVR model. The reasoning behind this is that a CNN model is better able to determine what features it requires, instead of being told to use a pre-determined set of features.

1.3. The system

Using different models of prediction, the goal of the system is to successfully grasp salad using a tactile gripper. The system relies on RGB-Depth (RGB-D) images from a camera overlooking the grasp area as well as tactile feedback from the gripper once the salad has been found.

In order to estimate where the gripper should be positioned in order to grasp the salad, each model has to output a 6-degree of freedom (DOF). This is a combination of positional and orientational vectors that tell the robot where it should position the gripper, and how it should be rotated. The models should also attempt to predict how the gripper should close onto the salad. This is possible due to the tactile sensitivity of each gripper finger. Being sensitive enough to measure the slightest contact, they ensure that the salad is not crushed when being picked up, only gripped tightly enough to ensure a secure grasp.

The models being used to generate these parameters either accept data extracted from depth and/or colour images, or the images themselves. In cases where the images themselves are not acceptable input, features have to be extracted from the images by another system. These features include the position of the salad, its orientation, as well as length, width and height at predefined points. With a robust system capable of repeatedly locating and extracting such features, a model is able to predict corresponding grips.

The first models to be attempted were generated using Support Vector Regression (SVR). These models accepted extracted features as input, and in return predicted a 6-DOF. Various variations of this model were generated, in an attempt to its optimum.

Once successful grasps had been achieved using SVR it was time train CNN models. These models are the more complex, as they do not use the extracted features, instead they make use of actual images. The model itself is responsible for figuring out what features in the images it finds important, learning what filters are needed in order to generate accurate predictions. Various variations were built, depending on input and parameters.

Due to the hardware required to run the system, a description and link to a video showing several of the main features has been included in appendix A. An explanation of the zipped code file is available in appendix E.

2. Theory and Background

This chapter is an meant to be an introduction for readers that have not fully grasped what machine learning is, as well as introducing them to concepts such as Support Vector Regression (SVR), Neural Network (NN) and Convolutional Neural Network (CNN).

2.1. Machine Learning

Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.
- Arthur Samuel, 1959[16]

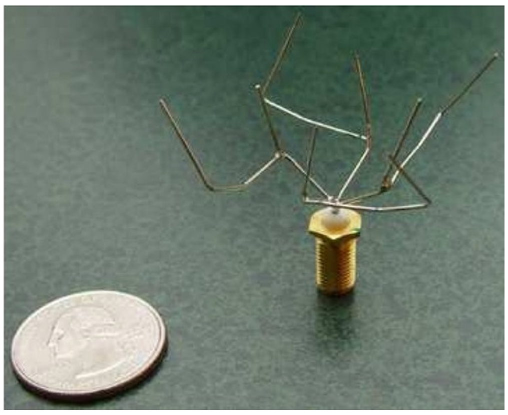
In 1959 Arthur Samuel defined the term Machine Learning as the field of study that gives computers the ability to learn without being explicitly programmed. Samuel was one of the early pioneers within this field, being the first to write a program that learned how to play checkers. Each players positioning of pieces was given a score, and simulations were run on possible valid moves. A mini-max search strategy was then used to determine what move would *maximise* the its own score, while at the same time *minimising* its opponents. The program would also remember previous moves, utilising this knowledge in its decision making, allowing past experiences and plays to influence its games.

> *Well posed Learning Problem: A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.*
>
> - Tom M. Mitchell, 1997[15]

Checkers was just the beginning, the use of Machine Learning in solving problems has increased over the past decades, becoming more and more instrumental in our digital lives. In some tasks such as email filtering, image classification or Optical Character Recognition(OCR) it has proved difficult for humans to design and program optimal algorithms. Instead Machine Learning is applied, along with a large amount of training data, to allow an optimal algorithm to be generated. If we refer to the definition by Tom M. Mitchell, and use email filtering as an example. The task T would be determining if an email is SPAM or not. Performance P would be how often it correctly classifies emails, an increase meaning the system is learning correctly. Experience E would be a set of previously categorised emails, and any correctly classified email can later be included into this set, further increasing the success of the system.

These models will work well for the task specified, but trying to understand how they work is not always clear. The Machine Learning algorithms will often make associations and choices that are not obvious to a human, but to a machine and its system they will make perfect sense. Two unconventional solutions can be seen in NASAs attempt at designing a spaceship antenna[13], and Adrian Thompson in his attempt at microchip design[17]. Here they made use of evolutionary algorithms. Each generation would produce a set of designs, which would be ranked according to performance, the best being chosen for "reproduction". A new generation was thereafter generated as a combination of two "parents" as well as a certain amount of mutation. Initially designs were random, but subsequent generations would slowly optimise according to performance measures. In NASAs case, this resulted in an antenna that looks nothing like what you would expect an antenna to look like. Instead it resembles a spider on its back, with its legs pointing in random directions as seen in figure 2.1. The optimal microchip would make use of only thirty-seven of its one-hundred logic gates, some not physically connected to anything at all, others connected in feedback loops. The disconnected logic gates were still crucial in the chips design, influencing performance through electromagnetic fields in neighbouring gates.

These are two examples were machine learning has been used, and the resulting solutions while performing optimally, are vastly different to what a human would produce.


(a)

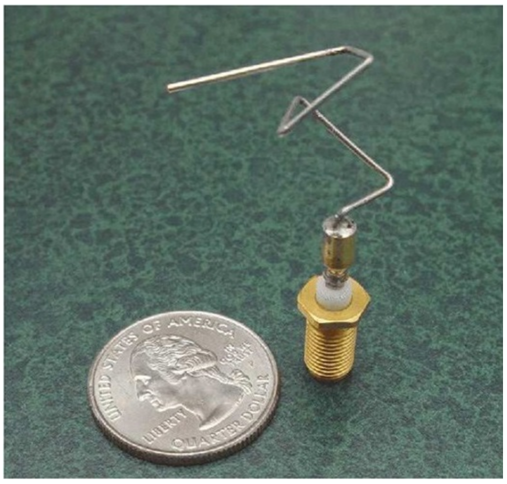

(b)

Figure 2.1: Photographs of prototype evolved antennas. (a) the best evolved antenna for the initial gain pattern, (b) the best evolved antenna for revised specifications.

2.1.1. Unsupervised, Supervised and Reinforcement Learning

Machine Learning tasks are often separated into three learning categories. This depends on what the goal of the system is and what kind of data is being worked on, whether it is labelled or not.

Unsupervised Learning
In unsupervised learning the data is not labelled. The goal is to discover patterns or relationships within the data. This allows for the data to be clustered into groups, with the possibility of placing new data within existing groups.

Supervised Learning
In supervised learning the model is trained on data with labels. This means that it is possible for the model to approximate a label for similar label-

less data. There are two kinds of supervised learning, either *Classification* or *Regression*. Classification problems are problems where the goal is to classify the input into one of a finite number of classes. While in regression problems, there are no classes, instead the goal is a continuous value.

Reinforcement Learning
Reinforcement Learning is somewhat different to supervised learning, as there is no optimal solution. Instead the machine decides what action to take in order to maximise its performance. If the decision is good, it is positively reinforced, while a bad decision will be penalised. As such the system will learn to make correct choices, in order to achieve the goal.

2.1.2. Data Splitting

When generating supervised training models, it is important to be aware of the fact that such models can focus to much on the trained data, becoming *Overfit*. This means that the model has not learned general concepts in order to make predictions, instead it has learned specifics to the training data. Although it achieves high accuracy when predicting on the training data, new and previously unseen data, will not achieve the same accuracy. This can be compared to remembering a solution in mathematics, you know that $2 \times 3 = 6$, as you have seen it before. But when asked to what $2 \times 7 = ?$ is, you have no idea, as you only remember question-solution pairs, having no idea how multiplication is done.

In order to combat this overfitting, data is often split into a *training set* and *validation set*. When doing this split, it is important that both sets are representative of the original data. The ratio for this split is not set in stone, and depending on your data, training-validation splits of 90-10, 80-20 or 75-25 are all commonly used.

Training Set The training set will be used to train the model, and is therefore the largest. This data is labelled, with the optimal output. By sending this data through the model, and comparing the models predicted output with the optimal one, it is possible to calculate an error. This is then used to refine the model, and minimise said error.

Validation Set Once the model has had some training, it is possible to test how it responds to unknown data. The validation set is sent through the model as during training, although now the error is not used to refine the model. Instead it is used to indicate the models accuracy, signalising if it is general enough to produce accurate predictions, or has overfit on the training data. By tracking the validation accuracy through training, it is possible to determine when the model has reached its peak, as it will start to decline, indicating that the model is beginning to overfit. Once the accuracy has started to decline, it is time to end training, as the model is at its best.

2.1.3. Data Augmentation

In some instances, it is not feasible or possible to gather a massive data-set. In cases such as this, it is possible to make use of various methods in order to increase the size of the original data. This can either be done before training, at the cost of extra disk-space being used, or simultaneously as the data is being loaded. When augmenting data, most common methods focus on augmenting in ways that increase the robustness of the model. If the size of an object has no implication on the output, it is possible to augment the data, by adding copies where the same object has been re-sized.

2.2. Support Vector Machines

The Support Vector Machine is a form of supervised learning algorithm, developed by Vladimir N. Vapnic and his colleagues in 1995[12], capable of both classification and regression. Mainly when referring to a classification problem, Support Vector Machine is used, while regression tasks make use of Support Vector Regression.

The Support Vector Machine algorithm works by plotting each data instance within a n-dimensional space, n being the number of variables or features of the data. Now that the data has been plotted within the n-dimensional space, it is time to locate the hyper-plane that best divides the data. This can be seen in figure 2.2 where H_3 is the hyper-plane with the larges margin between classes.

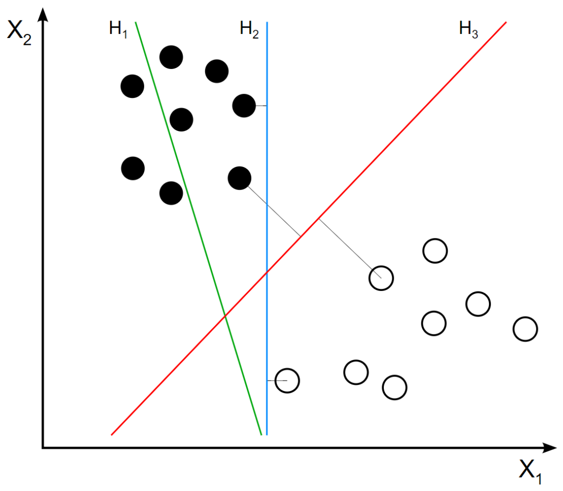


Figure 2.2: Figure showing hyper-planes in 2D. H_1 does not separate the classes, H_2 separates with small margin, while H_3 separates with maximum margin

2.3. Deep Learning

Deep Learning is a field within Machine Learning that makes use of models inspired by the structure of the brain. These models are therefore often referred to as neural networks, where artificial neurons take on the roll of axons, and weighted connections take on the roll of synapses between them. Each neuron is capable of transmitting a signal, the signal strength dependant on both activation function and the combined incoming signal to it. Neurons are often organised into layers, with the outputs of one layer, becoming the inputs of subsequent layers as seen in figure 2.3. This means that signals can traverse through the layers, activating neurons, eventually generating an output.

For classification problems, an example will activate certain neurons in the first layer, referred to as the input layer. This will cause signals to propagate through the network, eventually activating a single neuron in the final layer. In this final layer, referred to as the output layer, every neuron would represent a unique class, signifying that the example should be classified as such.

With faster and better hardware being constantly developed and released,

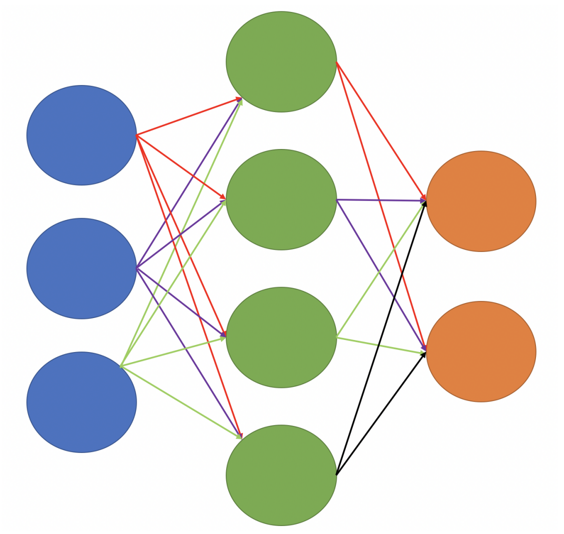


Figure 2.3: A fully connected neural network. Blue nodes represent input neurons, green nodes represent hidden neurons and red nodes represent the output neurons.

Deep Learning has been steadily increasing in popularity and success. With more powerful components, the size of networks can be increased, as well as the time spent training models decreased. This means Deep Learning is being applied to more and more problems. As long as the data is digitised, be it sound, images, text or weather data, to name a few, it is possible to classify it, or make predictions from it. With the performance of such systems only increasing in quality over time.

2.3.1. Neural Networks

As mentioned earlier Neural Networks are models formed through a graph of neurons. These neurons are connected to each other through learnable weights w. In addition to these weights, each neuron has a learnable bias b. When a neuron receives a set of inputs x, it calculates its output y as shown in equation 2.1. f is the activation function, it makes sure that all outputs from neurons are kept within specific ranges, depending on what function is chosen.

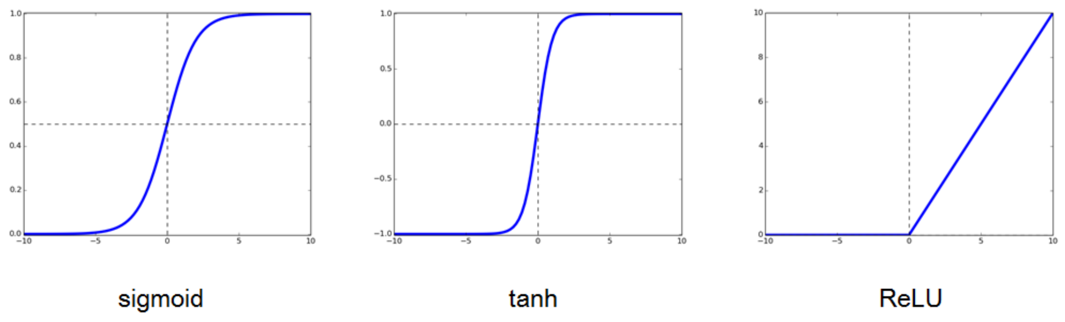


Figure 2.4: The three most common activation functions within Deep Learning. From left to right, Sigmoid, Tanh and ReLU

$$y = f(\sum_i w_i \bullet x_i + b) \tag{2.1}$$

Activation Functions

The activation functions job is to take an input, and with the help of a mathematical expression, make sure that the output is within a certain range. Presently the three most common activation functions can be seen in equations 2.2, 2.3 and 2.4.

$$\textbf{Sigmoid} \qquad f(x) = \frac{1}{1 + e^{-1}} \tag{2.2}$$

$$\textbf{Tanh} \qquad f(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.3}$$

$$\textbf{ReLU} \qquad f(x) = max(x, 0) \tag{2.4}$$

The Sigmoid activation function ensures that the output is somewhere between $(0, 1)$. The Tanh activation function allows for a negative output, keeping it within the range $(-1, 1)$. The ReLU activation function sets the output as any positive number, $[0, \inf)$. These functions are also visible in figure 2.4

Training a Neural Network

There are several steps involved when training a Neural Network. First the network is given some training data, resulting in a prediction *y*. This prediction is then used along with the correct output *y'* to calculate an error *E* as seen in equation 2.5. The error is calculated by taking half the square of the Euclidean distance between the prediction and goal. The reason for halving it is to cancel the exponent when differentiating in equation 2.6.

$$E = \frac{1}{2}|y - y'|^2 \tag{2.5}$$

$$\frac{\partial E}{\partial y'} = (y' - y) \tag{2.6}$$

The goal of training is to minimise this error value. This is achieved through a method called *back-propagation*. E is therefore partially derived as seen in equation 2.6 in regard to each neuron. The resulting error is then propagated back through the network, updating the models weights. This causes the models performance to slowly increase after each weight update, as well as lowering the error value.

Epochs and Batches
When training a network the same data is often repeatedly fed through the model. For every time the all the training data is fed through once, an *epoch* is completed, and the next one can begin.

Each epoch is usually split into several *batches* as the training data is usually to large to be able to fed through the model at the same time. After each batch, back-propagation is performed, increasing the performance of the model. It is important to shuffle the training data after each epoch, before splitting into batches, in order to reduce the chance of batch overfitting.

2.3.2. Convolutional Neural Networks

Convolutional Neural Networks are a subclass of Neural Networks relying on many of the same mechanisms to function. One notable difference between the two kinds, is that a Convolutional Neural Network accepts a matrix of

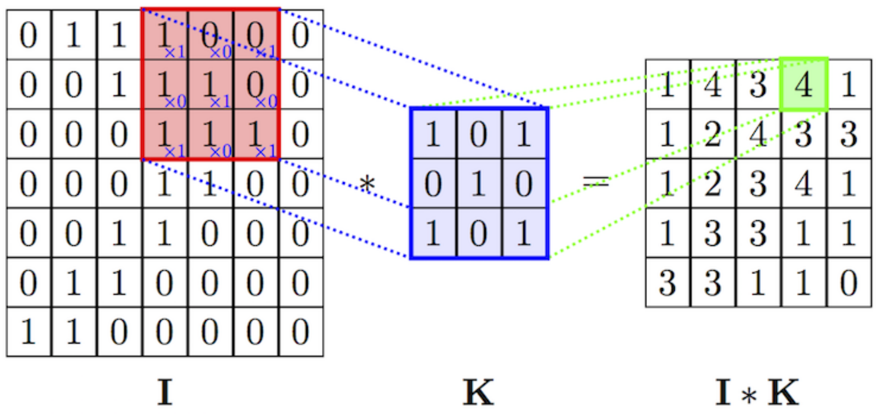


Figure 2.5: A figure depicting the activation map $I{*}K$ calculated using equation 2.7. I is the input matrix, while K is the current filter.[19]

numbers as its input, instead of a list. This means it is possible for the network to accept an image as its input, as images are just values in a matrix. When dealing with Convolutional Neural Networks, there are three main kinds of layers to be aware of, the Convolutional Layer, the Pooling layer, and the Fully Connected layer.

Convolutional Layer

The Convolutional Layer is the strength behind Convolutional Neural Networks. These layers are made up of multiple learnable filters. Filters are smaller sets of weights that can learn how to detect different features withing the image. Figure 2.5 shows us how a single filter K is convolved over the input matrix I to produce the 2D activation map $I * K$, using equation 2.7.

$$(I * K)_{xy} = \sum_{i=1}^{h} \sum_{j=1}^{w} K_{ij} \cdot I_{x+i-1,y+j-1} \tag{2.7}$$

Since a Convolutional Layer is made up of multiple filters, it produces multiple activation maps that are stacked along the depth dimension. This also means that different filters can learn different features. Some might become edge detectors, while others detect colour. This way, the network is itself in

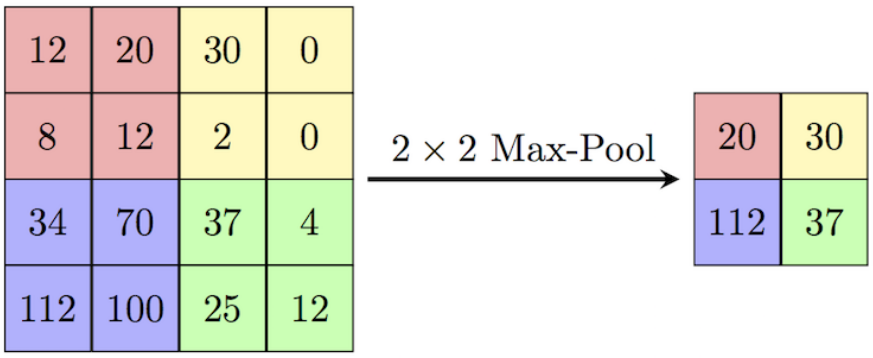


Figure 2.6: A Figure showing 2 × 2 MaxPooling with stride 2, on a 4 × 4 matrix.[19]

charge of determining what is needed in order to make accurate predictions depending on the required output.

Pooling Layer

Pooling layers are used in order to reduce the size of layers, thus reducing the number of parameters in the network. The most common pooling method is *MaxPooling* with a filter size of 2 × 2 and a stride of 2. This can bee seen in figure 2.6. The resulting matrix has been reduced to a quarter of its original size, halving in height and width.

Fully Connected Layer

The Fully Connected Layer is as the name implies a layer where each node is connected to every node in the previous layer. This is the same as in a normal Neural Network. In order for a Pooling or Convolutional layer to be connected to a Fully Connected Layer, it has to be flattened into one dimension.

2.4. Previous Work

2.4.1. Rig setup and system

In order to gather the required images as well as robotic and hand data, a custom rig setup was constructed during a summer internship in 2016 with the help of Linn Danielsen Evjemo. The rig that was constructed during the summer has since then been improved upon during the autumn, while writing a project assignment. Improvements include re-positioning of the camera in order to capture better images, as well as improved calibration procedures for increased precision.

The rig consists of a Denso VS-087 on a fixed platform, with attached cardboard grasping area. 1 meter above the grasping area, a Microsoft Kinect2 RGB-D camera is positioned to capture both depth and colour images of any object placed within its field of view. A RaspberryPi-3 (rPi-3) single board computer is also attached to the aforementioned rig, this computer is used solely to interface the robotic gripper with the windows machine over a custom TCP/IP connection. The robotic gripper is a ReFlex TakkTile hand from RightHand robotics. In order to remotely control the robotic arm and gripper the STEM system from Sixense was utilised, it makes use of magnetic fields to track controllers in relation to a base station.

During the summer internship in 2016, a program written in LabVIEW was created for controlling the different parts. LabVIEW was chosen as libraries for controlling both the robot, camera and STEM controller were already available. Unfortunately attempts to connect the gripper using the Robotic Operating System (ROS) directly to LabVIEW were not successful, instead a rPi-3 was set up as a server with custom TCP/IP commands being sent over wifi. Feedback from the tactile sensors in the gripper are numerically displayed on the screen while no haptic feedback is provided to the operator.

The program allowed for either manual commands to be issued through the computer, or for an individual to control both robot and gripper using a STEM controller. The controller method was utilised when gathering examples by having the robot shadow any movements made by the controller. The operator is therefore able to easily position the gripper correctly for

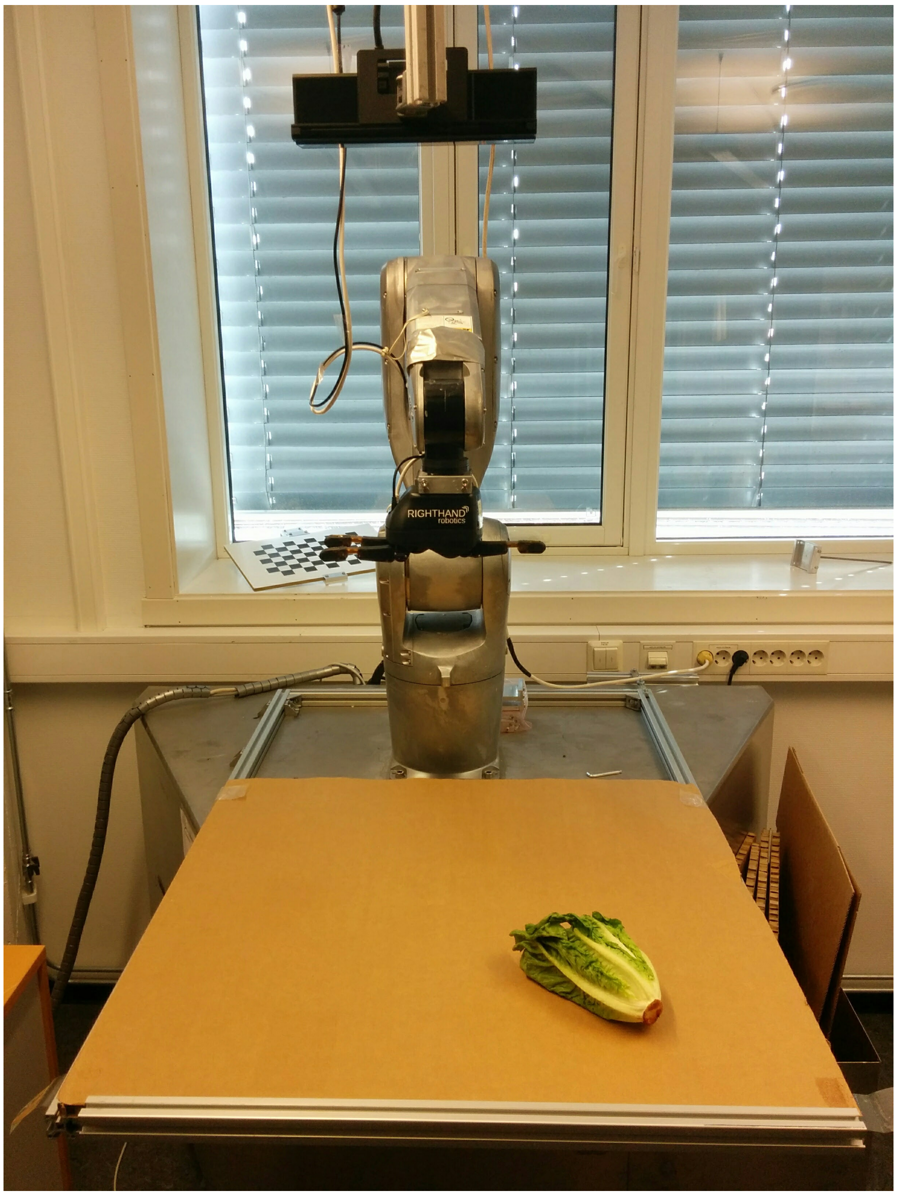


Figure 2.7: This figure shows the current setup. The gripper is attached to the robot, with the camera attached to the ceiling, aimed perpendicularly towards a grasping area.

grasping of an object, as any movement of the controller was directly translated to the gripper in near real time. The buttons on the controller are also programmed to allow procedures to be run. This reduces the need for mouse interaction with the program, as commands like *save images* and *open/close gripper* can be activated remotely.

3. Methodology

This chapter explains the in a bit more detail how the system for grasping objects works, including what parts were used. It also describes calibration and how the dataset used in generating models is made up. The chapter finally explains how features are extracted from the images, as well as how both SVR and CNN models are built.

3.1. Software and Hardware

3.1.1. Software

The system for connecting together the different hardware was written in LabView, as libraries were readily available for all but one piece of equipment. The models on the other hand were generated using either Matlab or Python. In order for the programs to transfer data between them, the data was either written to *.csv* spreadsheets, a *.txt* file, or custom TCP/IP connections.

Labview
LabVIEW[3] is a system-design platform and development environment developed by National Instruments. LabVIEW uses a graphical dataflow language called *G* for visual programming. Its editor has both a block diagram and front panel window. The front panel works as the programs GUI, while the block diagram is where everything is connected.

Matlab and Python
Matlab[4] and Python[5] are two programming languages, while Matlab is a proprietary language and requires a license to use, Python is open-source. Matlab is also the name of the development environment used for its programming. Python on the other hand can be programmed in any text editor,

although an IDE will simplify programming. Python supports moduels and packages, encouraging code reuse.

Tensorflow
Tensorflow[11] is Googles response to Deep Learning, and made open-source in 2015. It was developed by the Google Brain Team within Google's Machine Intelligence research organisation for the purposes of conducting machine learning and deep neural networks research. Tensorflow is able to run on both CPU and GPU chips, runs on both CPU and GPUs, allowing for parallelization to increase computational power. Tensorflow is available as a module that can be imported into python.

3.1.2. Hardware

Main Workstation
Almost all programming was written and executed from a workstation running Windows 8.1. This machine was equipped with a *Intel i7-4790K @4.00GHz* CPU, and a *Nvidia GeForce GTX 970* GPU, as well as 16GB of RAM.

Denso VS-087
The robotic arm used for grasping the salad, as well as holding the calibration board is a Denso VS-087 robotic arm[1]. This robot has 6-axis, and a reach of 905mm from its centre. The arm is mounted on a base station placing it, 1.2meters above the floor. In order to control the robot, a teaching pendant can be used, or commands can be sent from a connected computer over ethernet.

Kinect 2 RGB-D camera
Images are recorded using the Microsoft Kinect 2.0[2] camera. The Kinect 2.0 camera boasts a 1080p colour camera, as well as an active IR camera with a resolution of 512x424. The Kinect 2.0 is able to generate a depth image using the *Time of Flight* principle on the IR images. With the help of a built in IR projector, it can measure the time it takes the light to travel to an object and bounce back. This is then used to produce a depth image.

RHand Robotics: ReFlex TakkTile gripper
The gripper utilised for picking up the salad was built by RightHand Robotics[8].

The ReFlex TakkTile[7] gripper has three fingers, one fixed, and two that can be rotated to alter their pose. Each finger is equipped with nine tactile sensors and is able to bend at a finger joint. It is controlled over ethernet using theRobotic Operating System (ROS).

Raspberry Pi 3
The Raspberry Pi 3[6] is a small cheap microcomputer, running a Linux distribution. The reason for making use of this device is that we were unable to connect the gripper to LabView using ROS[9]. Therefore a workaround had to be found, a simple server on the RPI3 therefore controls the gripper, with commands being transmitted over TCP/IP from a client on the windows computer.

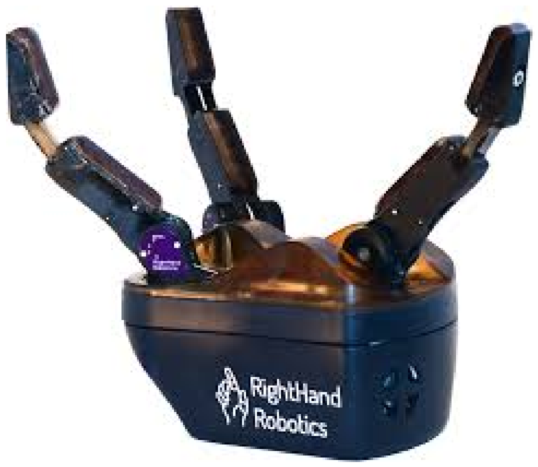
Sixense, STEM System
In order to easily control the robot and gripper, the STEM system[10] from Sixense is used. This system makes use of an A/C electromagnetic field in order to track its controllers. Each controller is equipped with a joystick and several buttons. This allows for wireless feedback of not only the controllers position and orientation, but can also be used to trigger various events.

3.2. Object-in-hand Calibration

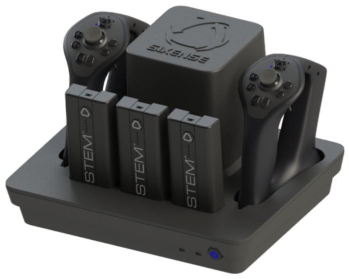
In order to be able to make use of data extracted from the Kinect camera, it is important to be able to transform between the camera and robot coordinate spaces. The method used to achieve such a transform is called object-in-hand calibration[18]. This calibration is done by attaching a planar calibration board fitted with a printed chessboard pattern to the robot as seen in figure C.1. This chessboard pattern is used to define a coordinate system, where the Z-axis is perpendicular to the board, while the X- and Y-axis run along the chequered squares. A toolpoint is defined with its point in the intersection between the two top left black squares, as seen in table D.2a. By gathering a set of images and corresponding robot positions, it is possible to calculate a transformation matrix. This is because both the image and robot position contain enough information to allow for the same spacial plane to be located. This can then be used to extrapolate where the camera is positioned in relation to the robot.

(a) Denso VS-087

(b) Kinect 2



(c) ReFlex TakkTile hand



(d) STEM system

Figure 3.1: A figure showing the main hardware components of the system. a) Denso VS-087, b) Kinect 2, c) ReFlex Takktile hand, d) STEM system

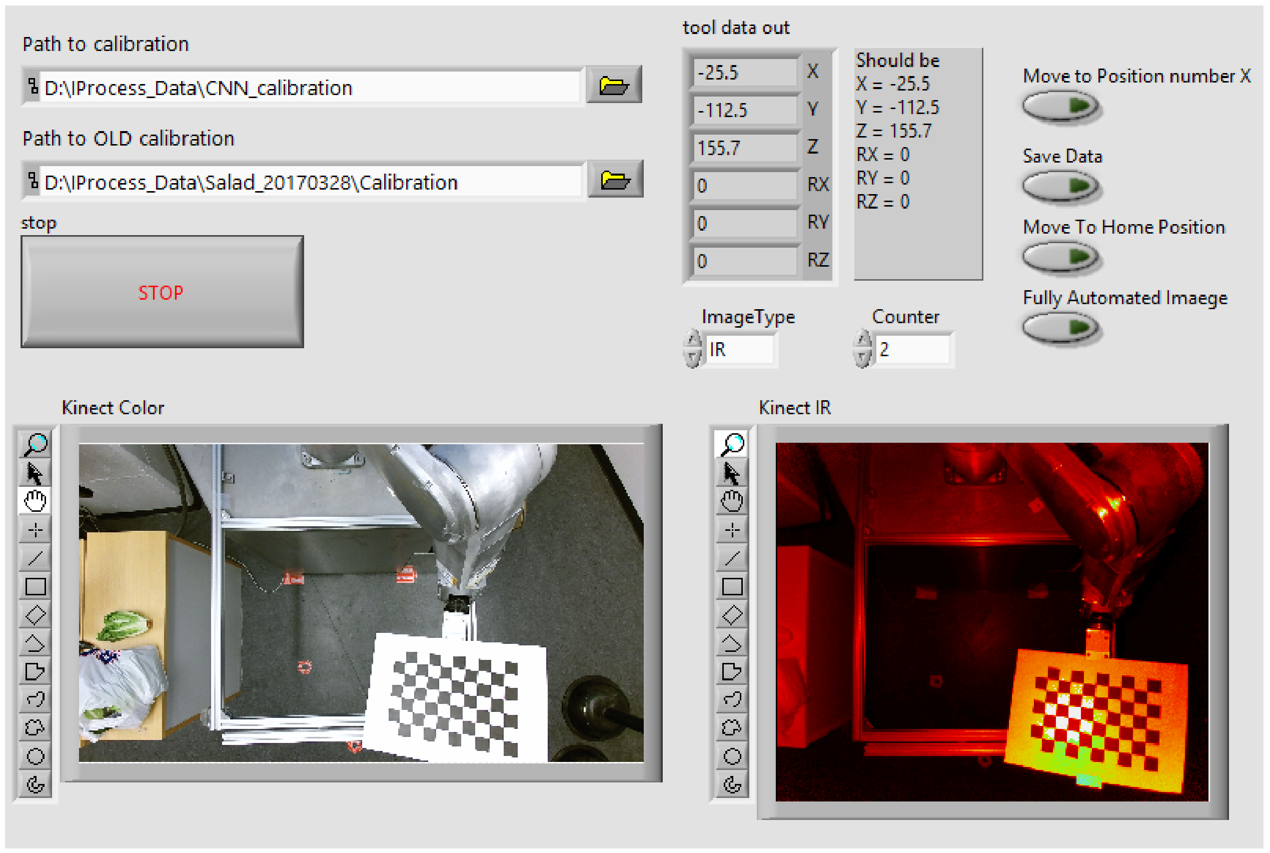


Figure 3.2: The program used to gather calibration images, it uses positions from a previous calibration in order to speed up the process.

3.3. Data-set

In order to generate models that would allow for grasp prediction, a set of examples have to be produced. For each example, there exists a set of images, and data gathered from the grasp. As well as gathering and generating the example set, it is important to be able to convert between the robots and the images coordinates. A calibration is therefore performed ahead of gathering any examples, or making any predictions. It is important for the calibration to be accurate as a low accuracy in calibration, will extend to the accuracy of the models.

Due to the capabilities of the Kinect2 camera, it is possible to save colour, depth and IR images, referred to as I_{RGB}, I_D and I_{IR}. For every example gathered, all three types of images are saved at different states of the grasp:

- $I^{background}$, taken before gathering any other images, used for normalisation of subsequent images.
- I^{init}, first image with an object, the gripper is in its home position.
- I^{grasp}, the gripper has grasped the object.
- $I^{release}$, the gripper has moved the object to its goal area, and is about to release object.
- I^{home}, the gripper is back in its home position, the object is in the home area.

Simultaneously as the I^{grasp} images are saved, various parameters for both the robot and hand are saved. They are recorded to a *pos.txt* file, and include the position of the gripper, as well as its orientation. In addition to data gathered from the robotic arm, tactile data from the hand is recorded, as well as the pose and position of fingers are saved.

3.3.1. Gathering

Generating data-sets was done at two separate instances, once during the project assignment, and a once during the writing of this thesis. The initial data set consists of 325 examples from 6 salads, evenly spread around the grasping area, as seen in figure B.1 in the appendix. After having used the first dataset to train SVR models and attempt to generate CNN models, it was decided that having a larger set of data could be beneficial. Therefore a second set of data was gathered. This dataset was not as large as the initial one, only making use of 4 salads, and supplying 200 distinct examples. Their distribution can be seen in figure B.2. Combining these two sets of data, allowed for a example pool of 525, whose distribution is seen in figure B.3.

Quality of the data Since the models were going to be trained on the data, it is important to make sure that all grasps were successful. This is because it would be detrimental to train a model on bad examples, causing it to learn that unsuccessful grasp are also good.

3.3.2. Data Augmentation

When working with machine learning the data used for training is sometimes not enough for the task, this can be rectified by augmenting with computer generated data, based on the original.

Translating Coordinate Space
By translating both the extracted data and the recorded data by the same vector it is possible to augment the current data. This is possible because both the extracted data, and recorded gripper data contain one set of coordinates that tell where the object and gripper are respectively.

$$gripper = G_{X,Y,Z} \qquad extracted = E_{X,Y,Z} \qquad random = R_{X,Y,Z}$$

$$new\_gripper = G_{X,Y,Z} + R_{X,Y,Z} \tag{3.1}$$

$$new\_extracted = E_{X,Y,Z} + R_{X,Y,Z} \tag{3.2}$$

By generating a random 3D vector, and translating both coordinates, but keeping all the other data, a new example is created. This example will have the same orientation for both gripper and object as the original, but will be in a new position.

Flipping the Gripper
By rotating the gripper 180 degrees it is still possible to grasp objects due to

the grippers semi-symmetrical build. Although the gripper has two fingers on one side, and one on the other, as long as the two fingers stay parallel, there is nothing that dictates which way to pick up the object.

In order to achieve this rotation of the gripper, the output values have to be altered in such a way to realign the gripper correctly. The gripper is positioned using a 6-DOF vector. Equation 3.3 shows how a 6-DOF is rotated 180 degrees, while still maintaining the same position.

$$\begin{aligned} original &= \begin{bmatrix} x & y & z & Rx & Ry & Rz \end{bmatrix} \\ flipped &= \begin{bmatrix} x & y & z & -Rx & Ry & Rz+180 \end{bmatrix} \end{aligned} \tag{3.3}$$

Rotating relative data
The models generated with the CNN only make use of a cropped image of the object, predicting a relative 6-DOF. This means that the prediction is relative to the centre of the image, and does not have to take into account where the object is located within the grasping area. Since the relative data is oriented around the centre of the image, it is possible to rotate both image and relative data to create new examples.

$$\begin{aligned} original &= \begin{bmatrix} x & y & z & Rx & Ry & Rz \end{bmatrix} \\ rot90 &= \begin{bmatrix} -y & x & z & Rx & Ry & Rz-90 \end{bmatrix} \end{aligned} \tag{3.4}$$

By rotating the data in increments of 90 it is possible to quadruple the number of examples. Since all the cropped images have the same height and width, rotating them by 90 degrees does not pose any issues with the input to the CNN. The relative 6-DOF can be rotated by 90 degrees as shown in equation 3.4. Repeating up to three times for a 90, 180 and 270 rotation.

3.4. Feature Extraction

Due to how Support Vector Regression (SVR) work, using the raw image data as an input would not an option. This meant an algorithm for locating and extracting certain features from the images had to be implemented. This algorithm had to be robust enough to allow for the biological diversity of salad while still getting accurate readings.

3.4.1. Locating the Object

There are many methods for detecting and locating objects within an image, one method is to look for discrepancies between two images. By subtracting one image from another, as seen in 3.5, any pixel value that is very similar in both images will approach zero. In colour images this means black, while in depth images this means there is zero change in perceived height between the images.

$$I_D^{norm} = I_D^{init} - I_D^{background} \tag{3.5}$$

It is therefore possible, using a pair of depth images and normalisation, to locate any objects introduced into one of the images. Although there will always be some minor discrepancies between images, the bigger the object, the easier it is to locate. The pixel values will now represent the height of the object perceived from the camera, instead of its distance from the camera. It is also possible to calculate the centre of the object, using only values greater than zero. Any pixels within the image, have to be transformed from camera to robot coordinates if they are to be located with the robot.

3.4.2. Determining Rotation

Due to the varying shape and dimension of the salad, creating a algorithm that could robustly align objects proved difficult. In order to calculate the orientation, the line with the lowest moment of inertia through the Center of Mass was calculated. This angle can be calculated using equation 3.6.

$$\theta = \frac{1}{2}\arctan\left(\frac{2I_{xy}}{I_{xx} - I_{yy}}\right) \tag{3.6}$$

This allowed for all the objects to be rotated so that they were horizontal, allowing length, width and height measurements to be taken in a robust and controlled manner. No models where trained using this method as the angle was, $0 \leq \theta \leq 180$, and therefore could not determine whether what way the salad was facing.

3.4.3. Highest Point Rotation

Early versions of the Support Vector Regression (SVR) system only made use of the depth images. Experimentation into different methods for determining orientation based on shape or curves did not give satisfactory results, due to the irregular shape of salads. The most promising solution basically determined what side of the salad had the highest point, rotating the salad an additional 180 degrees if this point was not to the left.

The Highest Point Rotation method had an accuracy of 88% on the initial data of 325 salads. 288 were correctly oriented, while the remaining 37 were rotated incorrectly.

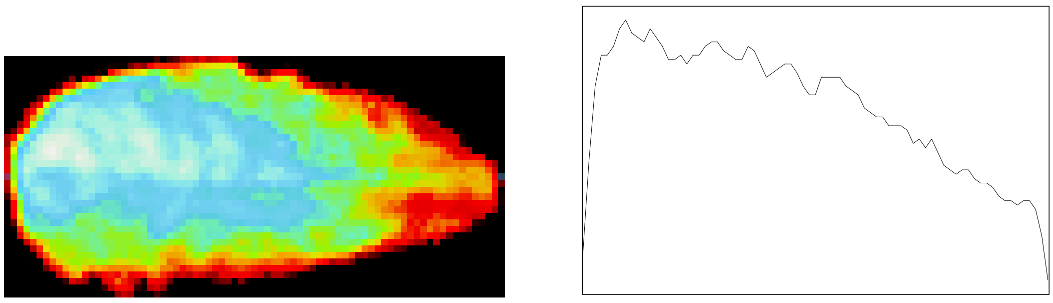

(a) Cropped depth image (b) Maximum pixel values in each column of figure 3.3a

Figure 3.3: Figure showing a cropped depth image, as well as its corresponding column height map.

3.4.4. RGB-D Rotation

By utilising the colour images alongside the depth images, it is possible to create a more robust algorithm for determining the orientation of a rotated object. Due to the different resolutions of the colour and depth images, as well as the position of both lenses, a look-up table is needed to correctly assign a colour to a depth pixel. In figure 3.4, the colour image 3.4a and depth image 3.4b are combined to produce the coloured depth image 3.4c.

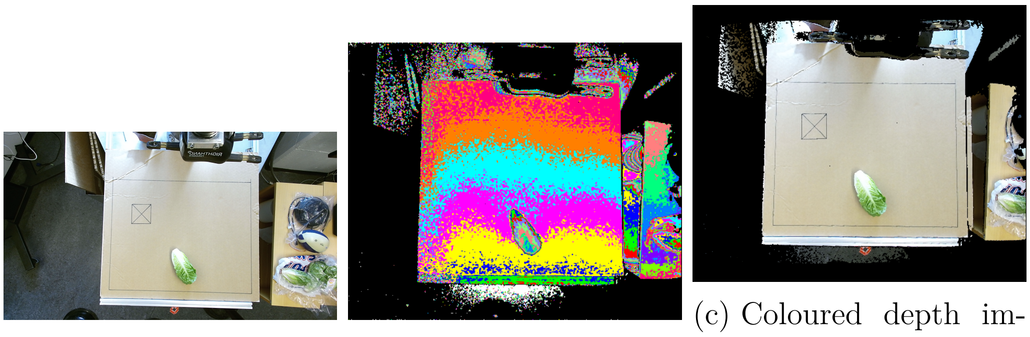

(a) Colour image (b) Depth image (c) Coloured depth image

Figure 3.4: A set of images showing the original and combined images when mapping colour to a depth image

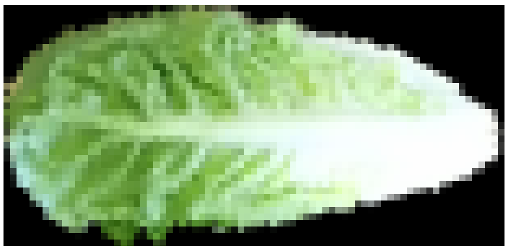
With this new colour image, any pixel in the depth image, can be assigned a colour. This means that by extracting a colour channel, calculating the mean of each column, and plotting a graph, the intensity of the colour can be seen. These values can then be used in the same way that the height data was used, locating the highest value and determining what side it is on.

Since the salad is mostly green and white, the blue colour channel seemed to produce the most distinct results, although both green and red could have been used.

3.4.5. Data Extracted

Knowing what features to gather when converting an image to a set of values is not easy. Depending on what task is being solved, it is up to the user to know what is needed, and figure out how to robustly extract this data. This differs from a Convolutional Neural Network, where it is the model that figures out how and what features it needs, in order to solve the task.

As seen in Figure 3.6, the salad has been rotated so that it is horizontal, with the leaves to the left, and its roots to the right. The centre of the salad is denoted **a**, its location calculated through centre of mass. The points **b,**


(a) Cropped masked depth image with colours

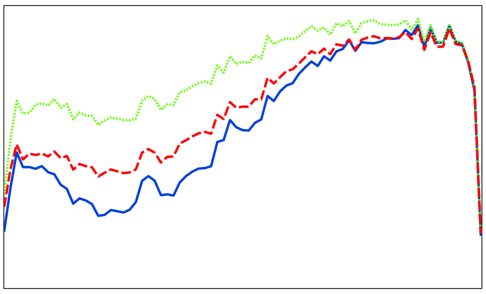

(b) Channels extracted from colour image

Figure 3.5: This figure contains an example of a cropped coloured depth image, as well as a graph of the highest RGB data point in each column.

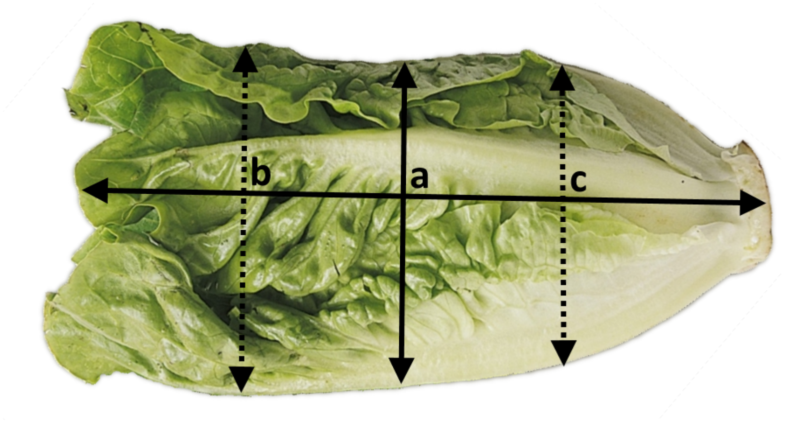


Figure 3.6: A figure showing the locations of features on a salad

c are calculated to be midway between **a**, and either side of the salad, on the same horizontal line.

$$\begin{aligned} \mathbf{pos} &= \begin{pmatrix} X_a & Y_a & Z_a \end{pmatrix} \\ \mathbf{rot} &= \begin{pmatrix} \cos\theta & \sin\theta \end{pmatrix} \\ Features &= \begin{pmatrix} \mathbf{pos} & H_a & L_a & W_a & \mathbf{rot} & H_b & W_b & H_c & W_c \end{pmatrix} \end{aligned} \tag{3.7}$$

In equation 3.7 we can see what features are recorded for each example. Z_a is the pixel value of $I_D^{init}(X_a, Y_a)$. H, L and W stand for Height, Length and

Width. Height is the pixel value of a normalised depth image, $I_D^{norm}(X, Y)$. Length and Width is the sum of non-zero pixels along a row or column, respectively. The rotation is recorded as the cosine and sine of the orientation, calculated using equation 3.6.

3.4.6. Transforming Features to robot coordinate

It is important to remember to transform the extracted features coordinates from the cameras coordinate space, and into the robots. If this step is overlooked, a new camera placement will cause the whole model to stop producing useful results, as they are dependant on a specific camera position. This is why a calibration was performed before gathering any data, as well as before making any predictions, to ensure that any error is not from movement in camera or robot.

3.5. Support Vector Regression (SVR)

The code for generating the SVR modell is written in Matlab, but the model it self can be exported and saved as a parameter file. This allows for other programs to make use of the model, and run predictions given the correct input.

A program for LabView was acquired that is able to accept the parameter file as input as well as a set of input values. This program is then able to output the correct prediction according to the model. It is also possible to make predictions using Matlab, but due to the rest of the robot control being in LabView, it is simpler to predict by importing the model.

The input values are the extracted features from the images, as seen in equation 3.7. In each example, the dataset has had its features extracted, and saved to a spreadsheet, that in turn can be imported into Matlab for training.

$$\begin{aligned}
\mathbf{pos} &= \begin{pmatrix} X_a & Y_a & Z_a \end{pmatrix} \\
\mathbf{rot} &= \begin{pmatrix} \cos\theta & \sin\theta \end{pmatrix} \\
Features &= \begin{pmatrix} \mathbf{pos} & H_a & L_a & W_a & \mathbf{rot} & H_b & W_b & H_c & W_c \end{pmatrix}
\end{aligned} \tag{3.7}$$

The goal of the model is to make predictions that allow for a 6DOF as well as the correct pose and finger pressure for the gripper. This means that the model has to know that it is these values it should predict. A second spreadsheet is therefore constructed, containing this relevant data.

$$\begin{aligned}
\mathbf{pos} &= \begin{pmatrix} X & Y & Z \end{pmatrix} \\
\mathbf{rot} &= \begin{pmatrix} O_1x & O_1y & O_1z & Ax & Ay & Az & O_2x & O_2y & O_2z \end{pmatrix} \\
\mathbf{pressure} &= \begin{pmatrix} finger_1 & finger_2 & finger_3 \end{pmatrix} \\
Prediction &= \begin{pmatrix} \mathbf{pos} & \mathbf{rot} & figure & pose & \mathbf{pressure} \end{pmatrix}
\end{aligned} \tag{3.8}$$

The model should therefore take the features as seen in equation 3.7 as input, and after mapping them into a high dimensional space, make a prediction in the form of equation 3.8.

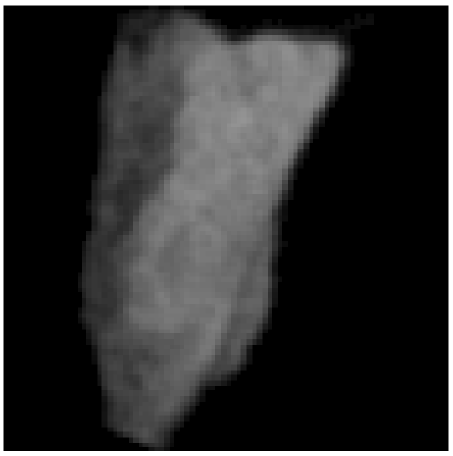
Feature Augmentation
In order to increase the size of the dataset, augmentation for the SVR models only used augmentation through translation. Both inputs and labels for training data had their coordinates translated with the same amount.

3.6. Convolutional Neural Network (CNN)

3.6.1. Preparing the images

In order for the model not become overly complex, it is possible to simplify what it has to predict. Instead of letting the model take in the whole image as its input, either depth (figure 3.4b), or coloured depth (figure 3.4c), it is possible to crop the image.

By locating the the salad using many of the same functions as in feature extraction from section 3.4, it is possible to crop the image to only this. The salad is then placed within a square of equal height and width, with the background set to 0. This results in images as seen in figure 3.7. By also recording the coordinates of the centre of this image, it is possible to calculate a relative 6-DOF to the salad, instead of a 6-DOF in relation to the robot.


(a) Cropped depth image of salad used in CNN models

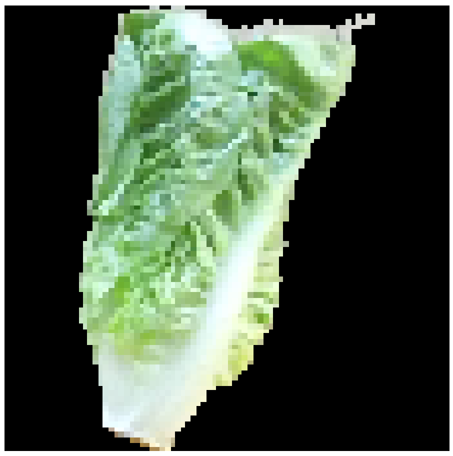

(b) Cropped coloured depth image of salad used in CNN models

Figure 3.7: This figure contains both the depth and coloured depth of a salad in preparation of it used as input in a CNN model.

3.6.2. Training a model

The code for generating a CNN model is written in python using Tensorflow. In order to make loading and working with the data easier several helper classes were written. In addition to these classes, code was written that allowed the trained model to be saved to file, as well as loading these files back into the system when only predicting.

Loading Data for training

In order to train any models the labelled training data has to be fetched and loaded into the program. This is done using a class *CNNData*. This class makes use of two other classes, *RelativeData* and *CSVData*. Together, they are able to load the images and labels into memory. By basing all data manipulation around one class, it is possible to keep the main program simple.

Input Images When loading the data it is possible to decide what set of images the model is to rely on, this can either be only the depth images, only the colour images, or a combination of both. The depth images are only one channel, while the colour images are three channels. By combining these two, it is possible to produce an image with four channels, containing information from both types.

Removing inconsistencies
The *CNNData* class checks for inconsistencies in the dataset when loading. These are examples where the relative gripper is much further away from the salad than it should be.

Splitting into training and validation sets
The *CNNData* class splits the remaining data into two classes, one containing training data, and one for validation. This split can be seen in table 3.1. The split is also able to make sure that both *Salad-1* and *Salad-2* have their examples evenly spread between them.

Augmenting the data
Once the data as been split, it is possible to augment it in order to increase the number of examples. This can either be by rotating the gripper in the relative data 180 degrees, while keeping the same image. Or it can be

Table 3.1: Table showing the training-validation split using a 75-25 ratio

	Total	*Salad-1*	*Salad-2*
Total	521	321	200
Train	390	240	150
Validation	131	82	49

rotating both image and relative data by 90 degrees. Using both methods it is possible for the dataset to become 8 times the original size.

Training
Now that the data has been loaded, it is run through the model in smaller batches. This is because it is mostly impossible to fit both the CNN graph and all the training data in the GPU memory at the same time. Instead the training data is randomised and divided into batches. After each batch has been run through the model, a loss is calculated, and the model updated in order to become better.

3.6.3. Saving and Restoring CNN models

Since some models might take several hours or even days before they have any success is is important to be able to save these models, so that they can be restored and utilised later. This is possible as Tensorflow has implemented a method where the graph along with its weights are stored. This data can then be used to restore the graph along with the trained weights, allowing it to make new predictions.

4. Experimentation and Results

This chapter focuses on the results gathered when comparing different SVR and CNN models, as well as how they measure up against each-other. The chapter also includes a analysis of the results in regards to the hypotheses.

4.1. Support Vector Regression (SVR)

Here we have the regression results when training SVR models on the *Salad-1* dataset. Each model is trained on the complete data, as well as on data that has been augmented. When augmenting, each example has its position moved 15 times, allowing the dataset to grow to sixteen times its original size. The augmentation is done by translating the position of both object and grip by the same vector. This was done with two different standard deviations, 50 and 200mm, giving a close augmentation, as well as an augmentation with examples better spread around. Models were also trained on examples where inconsistent examples was removed, reducing the number of total examples from 325 to 303.

Highest Point Orientation

The features used in this model have been extracted using the highest point flip method. As was mentioned earlier, this method did not correctly flip all salads, but still has a high accuracy when it comes to determining what is the head and what is the root of the salad.

Table 4.1: A table showing the SVR regression results using highest point flip feature extraction

Augment	Aug_STD	Consistent	Average	STD
0		False	**0.555478**	**0.312140**
0		True	0.535396	0.319572
15	200	False	0.329148	0.374905
15	200	True	0.409707	0.340042
15	50	False	0.379686	0.368258
15	50	True	0.414064	0.339041

In Table 4.1 we can see some results from running different combinations of augmentation with data gathered using the highest point flip method. Results show that augmenting the data does not increase the accuracy of the models, this might be due to the small example size and how all it does is overfit the model to some examples. It is also worth noting that removing inconsistent examples does seem to mean that this model performs better.

RGB-D Flipped

Since the highest point flip method did not manage to correctly rotate all salads with as high an accuracy as hoped, a different method was found. This method uses the colour image, mapping it to the depth image. This allows the program to distinguish between head and root of the salad as there is more green in the head than in the root.

Models were again generated for with and without augmentation, this time using features extracted using the new rotation method. As we can see, this managed to increase the average accuracy when not removing inconsistent data, while lowering the accuracy when also only using consistent data, in a weird contradiction to the previous models.

Table 4.2: A table showing the SVR regression results using the RGB-D flip feature extraction

Augment	Aug_STD	Consistent	Average	STD
0		False	**0.599968**	**0.310034**
0		True	0.545194	0.337891
15	200	False	0.384907	0.348562
15	200	True	0.353944	0.375388
15	50	False	0.428394	0.345957
15	50	True	0.391338	0.368840

SVR test 1, 31.03.2017

In order to gauge the accuracy of the SVR, the 4 salads used for expanding the dataset the 28. march were grasped using the best RGB-D flip method from Table 4.2. This was the un-augmented model, without removing any inconsistent values. Since these salads had not been used in any way when generating the model, this would be an indicator of whether or not the model would work or if it had become to saturated on the 6 salads used to train it.

6 grasps were performed on each salad, using the SVR model mentioned above. The salad was placed randomly within the grasping area. As it turned out, the pressure predicted by the model was not useful when grasping the salads. Its predicted pressure was not high enough to allow for a successful grasp and therefore a pressure of 20 on all sensors on all fingers was used. This value was chosen as it had had repeated success when practicing on an American footbal.

The results from the test can be seen int table 4.3, with an explanation as to what caused the grasp to fail in table 4.4. Each row of the table is a new salad, the order of the salad chosen does not coincide with the order used when generating the data-set on the 28. march.

Some of the grasps did not go as expected, therefore reasons for their failure were noted. The number indicates what grasp has been commented on, the asterisk indicates that the salad slipped out of the grasp while it was being moved to the goal area.

Table 4.3: Table showing whether a grasp was a success or fail during a SVR test the 31st of March

	1	2	3	4	5	5	
1	T	T	T	T	F	T	5
2	T	T	T	F	T	F	4
3	T	T	T	T	F	T	5
4	F	T	T	T	F	F	3
							18
							0.75

Table 4.4: A table showing the explanation for grasp failures during the first SVR test.

Grasp	Reasoning
5	Bad orientation
10	Bad orientation
*12	Good orientation, Bad grip
*17	Good orientation, Bad grip
*19	Good orientation, Bad grip
23	Bad orientation
24	Bad orientation

4.2. Convolutional Neural Network (CNN)

When generating CNN models there are many parameter that can be tweaked. From the size of the input, to the shape of the model, with layers having many different parameters each. After having minor success with predicting the tactile pressure with the SVR model, this was not included in the labels to train. Instead only the relative positioning of the gripper was to be predicted as seen in equation 4.1.

$$\begin{pmatrix} X & Y & Z & cos(Rx) & sin(Rx) & cos(Ry) & sin(Ry) & cos(Rz) & sin(Rz) \end{pmatrix} \tag{4.1}$$

The reason behind splitting each rotation into both cosine and sine values is due to the fact that angles are measured around a circle, and two values can therefore be the same, like 0 and 360, or -180 and 180. By splitting the angle into its cosine and sine values, this problem is negated. Once a prediction is made, the angles are unconverted and the relative grip is added to the salad. This produces a 6DOF that a robot can then use to move the arm.

4.2.1. The CNN model

The graph that was used for training and testing was generated to accept an image of 128 by 128 pixels. This image would be processed by four convolutional layers, the first two having 32 filters, the third 64, while the final convolutional layer made use of 128 filters. Following the second, third and fourth convolutional layer a max pooling layer was added. This caused the input dimensions to shrink, going from 128 to 32 pixels in both directions. When flattened this produced a staggering 32768 nodes, that were all fully connected to 4096 nodes that again were connected to the final 9 output nodes.

4.2.2. Training data

In order to find out what was the best model, different combinations of unagumented and augmented models were generated. The augmentation was either flipping the gripper 180 degrees, or rotating both image and gripper by 90 degree increments.

As we can see in table 4.5, the un-augmented training data performs the best using a four channel RGB-D image. The reason for the standard deviation within the data being so high is also discovered when we look at table 4.6. This table shows the regression results for each part of the 6-DOF, for the same runs as in table 4.5. Here we see that the X and Y coordinates are very accurate, with Rz having moderate success.

In figure 4.1 we see the regression plots for one of the runs. The y-axis represents the predication, while the x-axis represents the examples label. In the plots it is possible to see how the results are spread round. The coordinates are the top row, with the bottom row being rotation. The aim

Table 4.5: A table showing the average regression results for CNN models trained on various augmentation and image combinations.

#	Image	Aug90	Aug180	Normalise	Epochs	Average	STD
1	RGB-D	False	False	False	10	0.547127	0.418405
2	RGB-D	False	False	False	10	0.575091	0.408996
3	RGB-D	False	False	False	10	0.560633	0.423185
4	RGB-D	False	False	True	10	**0.609176**	**0.384432**
5	RGB-D	False	False	True	10	0.501289	0.516242
6	RGB-D	False	False	True	10	0.554905	0.435091
7	RGB	False	True	False	10	0.403782	0.461314
8	RGB	False	True	True	10	0.357368	0.513892
9	Depth	True	False	False	10	0.436706	0.450869
10	Depth	True	False	True	10	0.514106	0.389266

Table 4.6: Table showing individual regression results for the same models as in 4.5, the best result in each category has been put in bold.

#	X	Y	Z	Rx	Ry	Rz
1	0.985395	0.991102	0.114043	0.241104	0.163577	0.787539
2	0.988921	0.992169	0.186544	0.397383	0.072893	0.812636
3	0.987894	0.995102	0.278466	0.345484	-0.028885	0.785739
4	0.976146	0.990856	0.248980	**0.464929**	0.114093	**0.860049**
5	0.979298	0.992551	-0.116591	0.401335	-0.093109	0.844250
6	0.987487	0.991815	**0.282702**	0.260258	-0.021736	0.828906
7	**0.989539**	**0.995787**	0.142755	0.059172	**0.217516**	0.017923
8	0.988976	0.995254	0.315264	-0.091795	0.032511	-0.096002
9	0.976673	0.989702	0.117371	0.040516	0.037187	0.458784
10	0.980378	0.988186	0.201988	0.196007	0.173589	0.544490

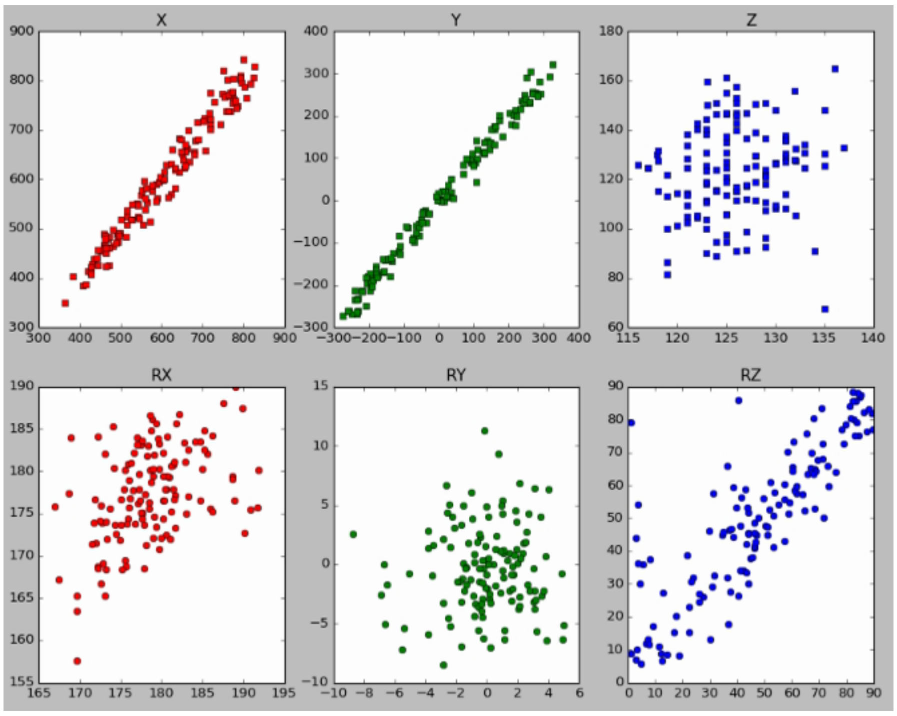


Figure 4.1: Figure showing regression plots on validation data for a CNN model. In each plot the correct value is along the x-axis, while the prediction is along the y-axis.

of the plots are to give a visual representation of how well a model can predict. If all models show plots as seen in X and Y, where the points are placed on a diagonal line, this indicates that prediction and correct label are very similar. Similarly if the points are scattered around, it means that the model does not predict very well.

4.3. Comparing the best models

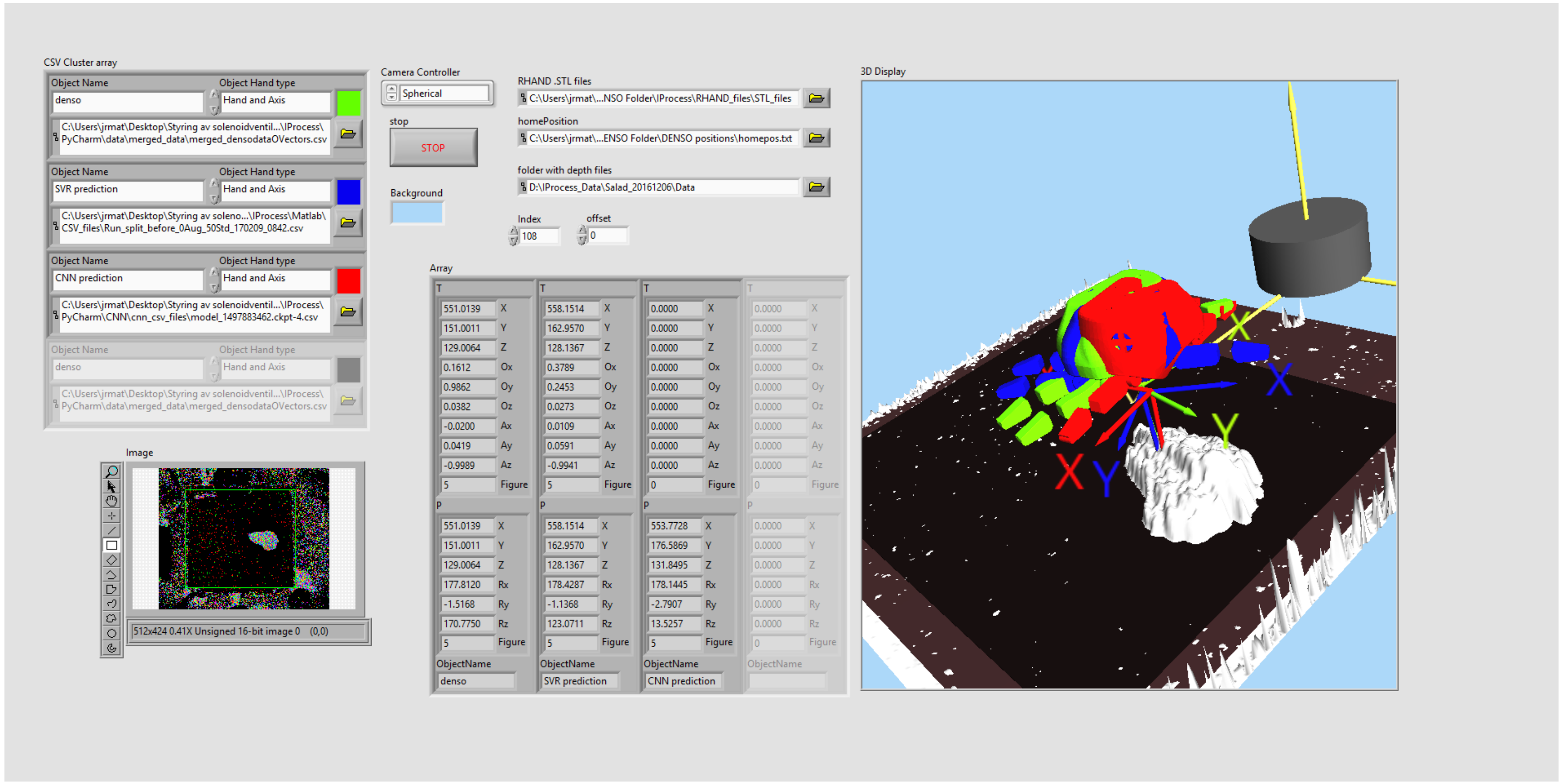
Once the models had been trained for both SVR and CNN it was time to see how the two types of models compared to each other. A program that simulated multiple gripper positions at the same time in a 3D environment using the same example was written. In figure 4.2 we see the main screen of the program. The green gripper shows the position gathered when generating the example, while blue and red show predictions based on the best models. The depth mesh showing the salad is also included, showing where it is placed prior to being grasped.

4.3.1. Virtual Comparison

Figure 4.3 shows various different grasp predictions. In figure 4.3b it is possible to see that the red gripper is flipped in relation to the green gripper, but still produces a working grasp. In 4.3d the green grasp from the dataset does not fit with the other grasps. This is because one or two mishaps were made during the gathering of examples, where the image and grasp are for two different salad positions, but saved as the same. Still the models managed to predict grips that allow for grasping, and although the models were based on different inputs and principles, both predictions are similar to each other.

4.3.2. Practical Test

It is also important to test the accuracy of models using real salads. Therefore 10 new salads were bought, these being used to make 5 predictions each, for both the best SVR model, and the best CNN model. This resulted in the tables 4.7 and 4.8, each describing the success of failure on the same 50 salad. The salad was randomly placed within the grasping area, with random orientation. Since only the SVR model predicted finger pressure, and this pressure was not useful for grasping salad, each finger was set to close until 20N as this was what had previously worked for grasping salad.



CSV Cluster array

Object Name	Object Hand type	File
denso	Hand and Axis	C:\Users\jrmat\Desktop\Styring av solenoidventil...\Process\PyCharm\data\merged_data\merged_densodataOVectors.csv
SVR prediction	Hand and Axis	C:\Users\jrmat\Desktop\Styring av soleno...\Process\Matlab\CSV_files\Run_split_before_0Aug_50Std_170209_0842.csv
CNN prediction	Hand and Axis	C:\Users\jrmat\Desktop\Styring av solenoidventil...\Process\PyCharm\CNN\cnn_csv_files\model_1497883462.ckpt-4.csv
denso	Hand and Axis	C:\Users\jrmat\Desktop\Styring av solenoidventil...\Process\PyCharm\data\merged_data\merged_densodataOVectors.csv

Camera Controller: Spherical

stop: STOP

Background

RHAND .STL files: C:\Users\jrmat\...NSO Folder\Process\RHAND_files\STL_files

homePosition: C:\Users\jrmat\...ENSO Folder\DENSO positions\homepos.txt

folder with depth files: D:\Process_Data\Salad_20161206\Data

Index: 108

offset: 0

Image

512x424 0.41X Unsigned 16-bit image 0 (0,0)

Array

T				
X	551.0139	558.1514	0.0000	0.0000
Y	151.0011	162.9570	0.0000	0.0000
Z	129.0064	128.1367	0.0000	0.0000
Ox	0.1612	0.3789	0.0000	0.0000
Oy	0.9862	0.2453	0.0000	0.0000
Oz	0.0382	0.0273	0.0000	0.0000
Ax	-0.0200	0.0109	0.0000	0.0000
Ay	0.0419	0.0591	0.0000	0.0000
Az	-0.9989	-0.9941	0.0000	0.0000
Figure	5	5	0	0
P				
X	551.0139	558.1514	553.7728	0.0000
Y	151.0011	162.9570	176.5869	0.0000
Z	129.0064	128.1367	131.8495	0.0000
Rx	177.8120	178.4287	178.1445	0.0000
Ry	-1.5168	-1.1368	-2.7907	0.0000
Rz	170.7750	123.0711	13.5257	0.0000
Figure	5	5	5	0
ObjectName	denso	SVR prediction	CNN prediction	

Figure 4.2: A Figure depicting the program used to compare models in virtual 3D. Green gripper for actual grasp in the dataset, Blue for grasp prediction using an SVR method, Red for a prediction using a CNN method.

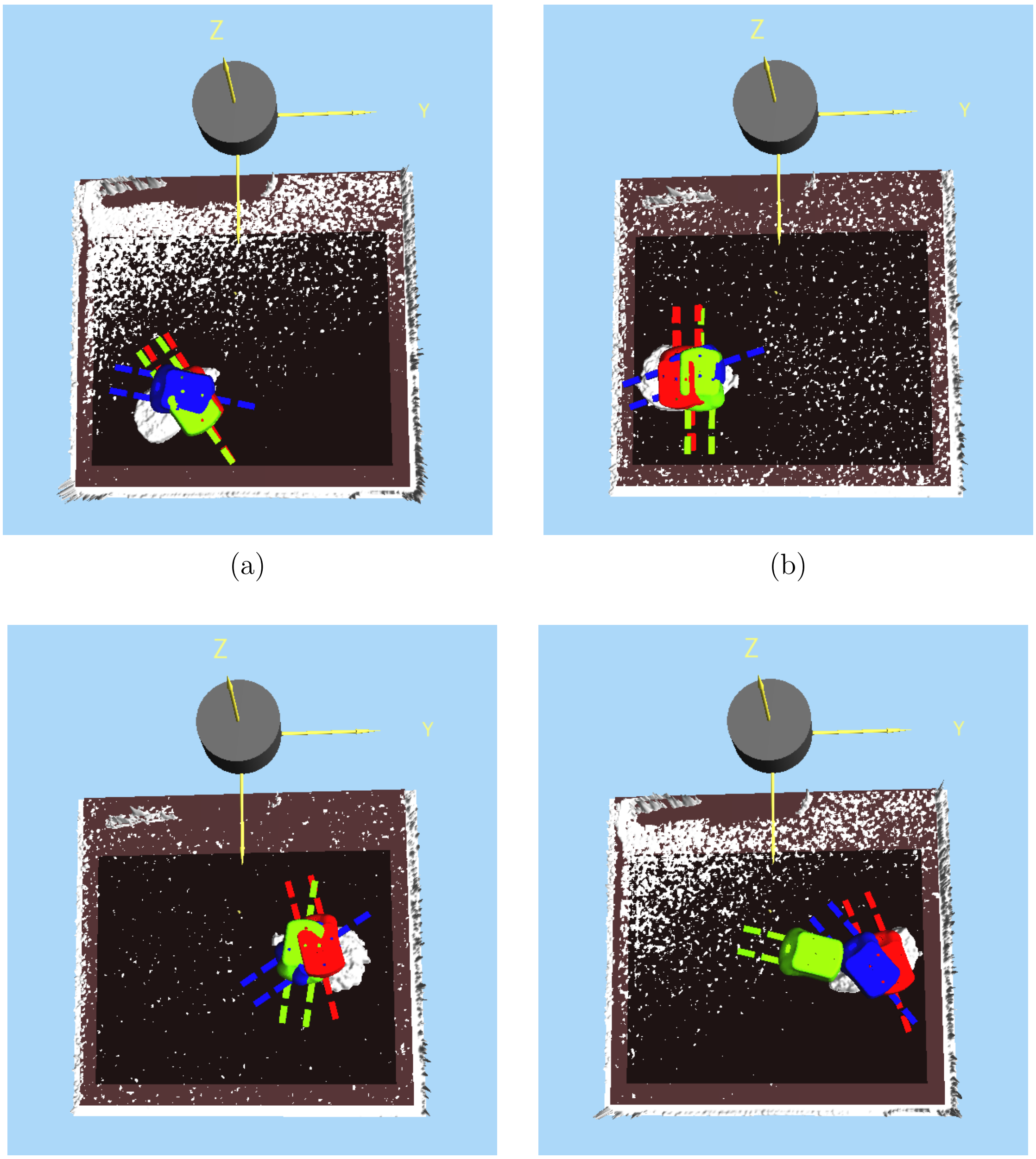


Figure 4.3: This figure depicts four different salads from the example data, and how the gripper is positioned using a SVR model (blue), a CNN model (red), as well as the original grasp (green).

Table 4.7: A table showing the success or failure of 50 grasps when predicting using the best SVR model. 10 salads were grasped 5 times each.

	1	2	3	4	5	
1	T	T	F	T	F	3
2	T	T	T	F	T	4
3	F	T	F	T	T	3
4	T	T	T	F	T	4
5	T	T	F	T	T	4
6	T	T	T	F	F	3
7	T	F	T	F	T	3
8	T	T	T	F	T	4
9	F	T	T	T	T	4
10	T	F	T	F	T	3
						35
						0.70

Table 4.8: A Table showing the success or failure of 50 grasps when predicting using the best CNN model. 10 salads were grasped 5 times each.

	1	2	3	4	5	
1	T	T	F	T	T	4
2	F	T	T	T	T	4
3	T	T	F	T	T	4
4	T	T	F	F	T	2
5	T	T	T	T	T	5
6	T	F	T	T	T	4
7	F	T	T	T	F	3
8	F	F	T	T	T	3
9	T	T	T	T	F	4
10	T	T	T	T	T	5
						39
						0.78

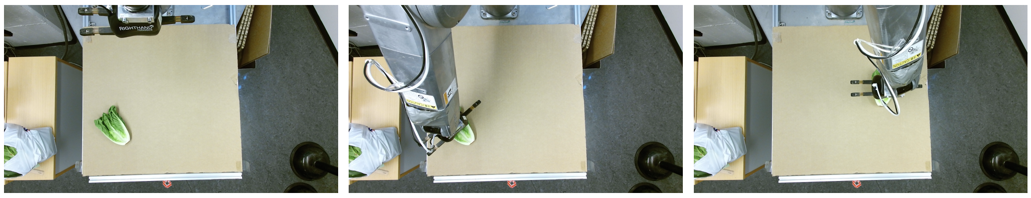

(a) Initial image (b) Grasping image (c) Release image

Figure 4.4: Example of successful grasp made using prediction from SVR.

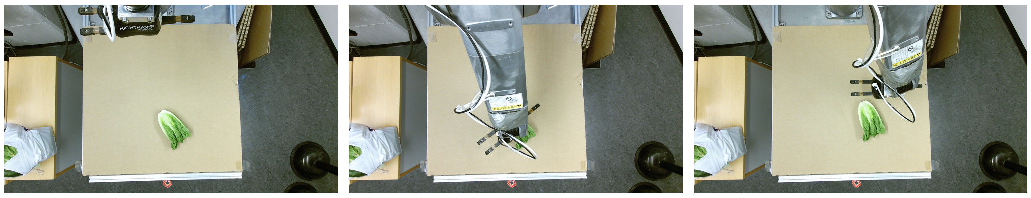

(a) Initial image (b) Grasping image (c) Release image

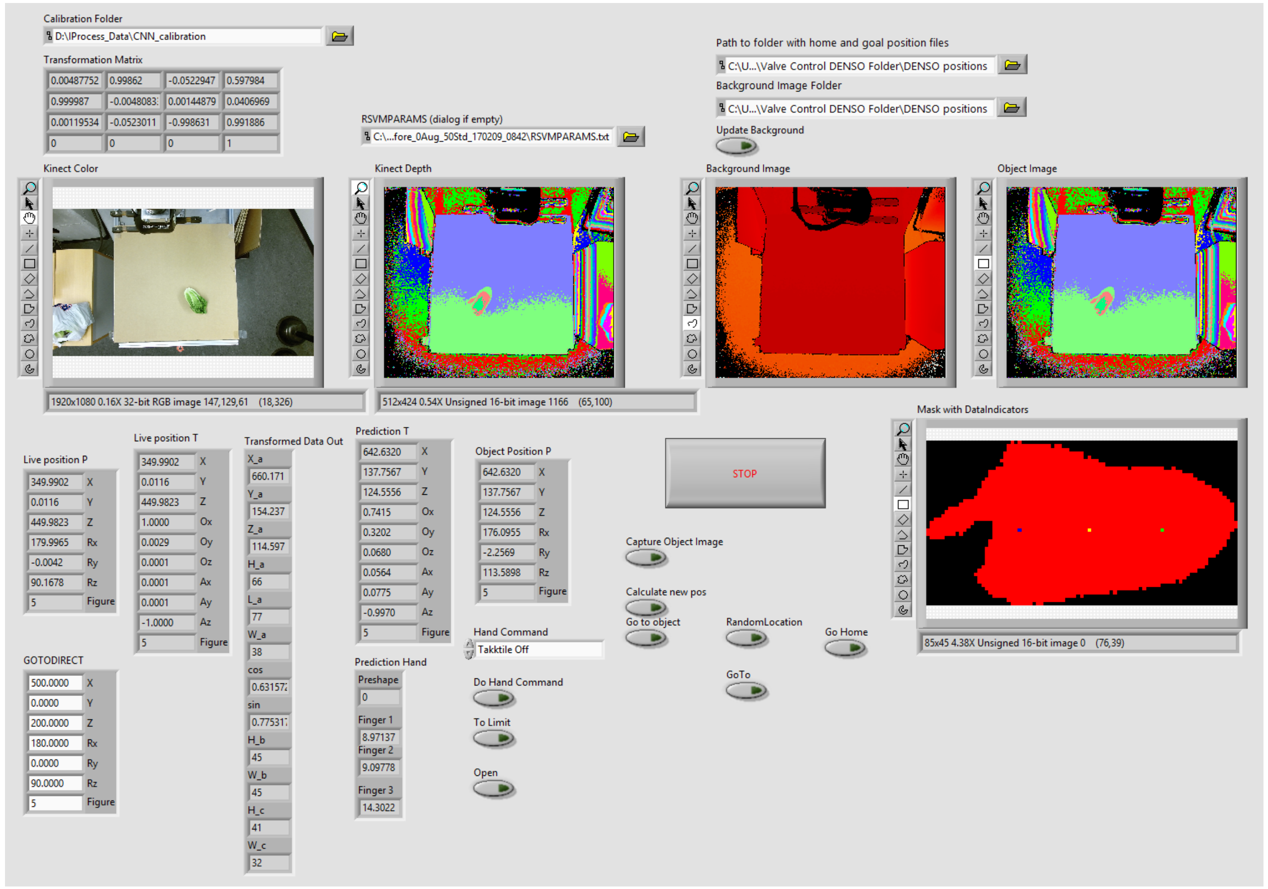
Figure 4.5: Example of failed grasp made using prediction from SVR. The salad was not grasped properly and slipped out during transit.

4.4. Analysis

As we can see in the 3D visualiser in figure 4.3 both methods are able to generate 6DOF pose predictions for grasping salad. When comparing the results gathered from grasping the same salads using both methods, we are able to see that they both manage to locate and pick up salads between 70% and 80% of the time. The SVR prediction managed to pick up and move 35 of the 50 salads to the goal. The CNN prediction method managed to move 39. With some of the failures being the result of sub-optimal gripping when the fingers did not close enough. Since each finger has several sensors along itself, there were some instances where one sensor pushed against the salad, causing the finger to halt. Once all fingers had stopped moving the resulting grip was weak, since it only had a loose grasp of the salad.

4.4.1. Hypothesis

In **Hypothesis 1** we stated that, *Given data gathered from a RGB-Depth (RGB-D) image it is possible to estimate a 6-degree of freedom (DOF) grip-*



Calibration Folder: D:\Process_Data\CNN_calibration

Transformation Matrix

0.00487752	0.99862	-0.0522947	0.597984
0.999987	-0.0048083	0.00144879	0.0406969
0.00119534	-0.0523011	-0.998631	0.991886
0	0	0	1

RSVMPARAMS (dialog if empty): C:\...fore_0Aug_50Std_170209_0842\RSVMPARAMS.txt

Path to folder with home and goal position files: C:\U...\Valve Control DENSO Folder\DENSO positions

Background Image Folder: C:\U...\Valve Control DENSO Folder\DENSO positions

1920x1080 0.16X 32-bit RGB image 147,129,61 (18,326)

512x424 0.54X Unsigned 16-bit image 1166 (65,100)

85x45 4.38X Unsigned 16-bit image 0 (76,39)

Live position P

349.9902	X
0.0116	Y
449.9823	Z
179.9965	Rx
-0.0042	Ry
90.1678	Rz
5	Figure

Live position T

349.9902	X
0.0116	Y
449.9823	Z
1.0000	Ox
0.0029	Oy
0.0001	Oz
0.0001	Ax
0.0001	Ay
-1.0000	Az
5	Figure

Transformed Data Out

X_a	660.171
Y_a	154.237
Z_a	114.597
H_a	66
L_a	77
W_a	38
cos	0.631572
sin	0.775317
H_b	45
W_b	45
H_c	41
W_c	32

Prediction T

642.6320	X
137.7567	Y
124.5556	Z
0.7415	Ox
0.3202	Oy
0.0680	Oz
0.0564	Ax
0.0775	Ay
-0.9970	Az
5	Figure

Object Position P

642.6320	X
137.7567	Y
124.5556	Z
176.0955	Rx
-2.2569	Ry
113.5898	Rz
5	Figure

GOTODIRECT

500.0000	X
0.0000	Y
200.0000	Z
180.0000	Rx
0.0000	Ry
90.0000	Rz
5	Figure

Prediction Hand

Preshape	0
Finger 1	8.97137
Finger 2	9.09778
Finger 3	14.3022

Hand Command: Takktile Off

Figure 4.6: Figure showing the program used to predict using SVR.

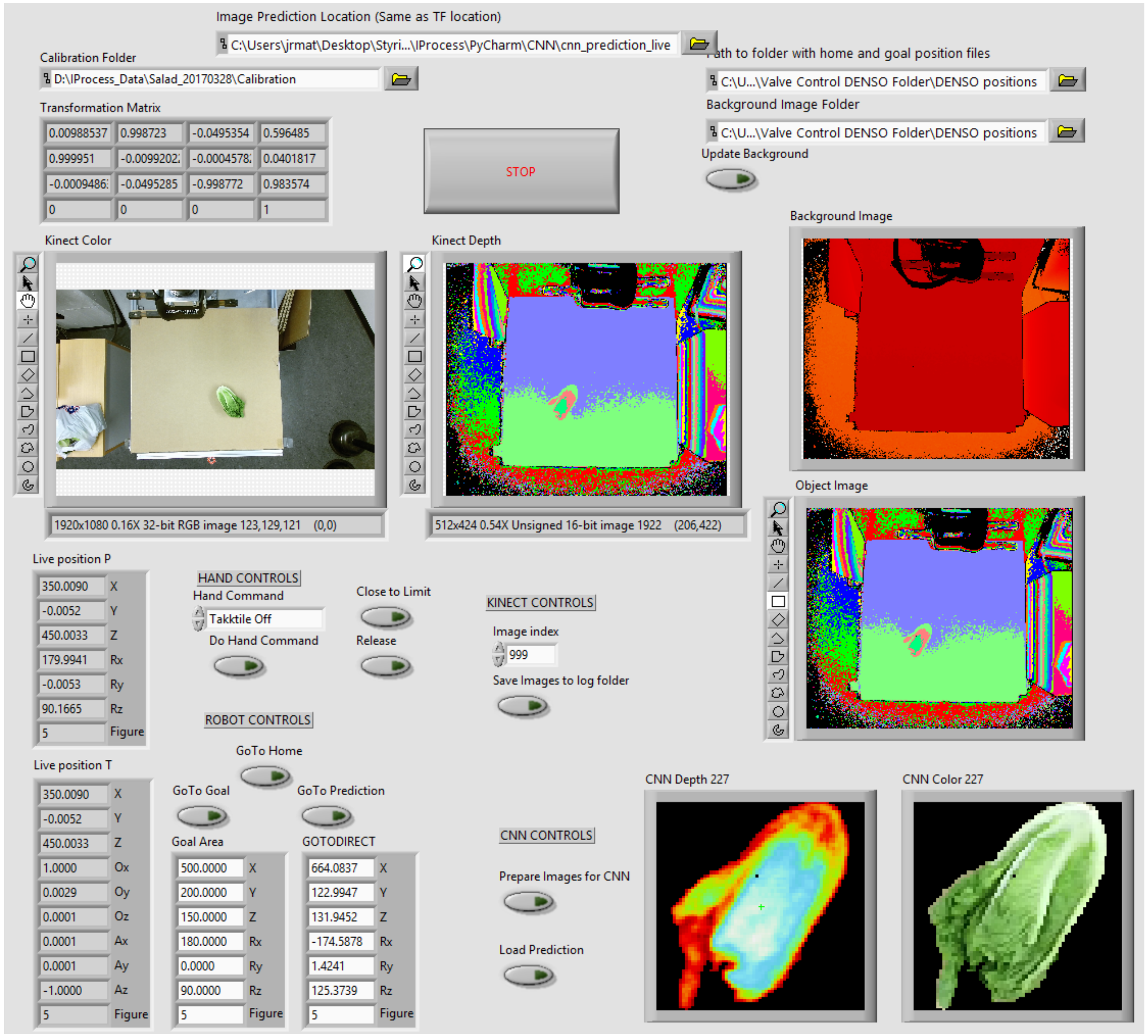


Image Prediction Location (Same as TF location)
C:\Users\jrmat\Desktop\Styri...\Process\PyCharm\CNN\cnn_prediction_live

Calibration Folder
D:\Process_Data\Salad_20170328\Calibration

Path to folder with home and goal position files
C:\U...\Valve Control DENSO Folder\DENSO positions

Background Image Folder
C:\U...\Valve Control DENSO Folder\DENSO positions

Update Background

Transformation Matrix

0.00988537	0.998723	-0.0495354	0.596485
0.999951	-0.0099202	-0.0004578	0.0401817
-0.0009486	-0.0495285	-0.998772	0.983574
0	0	0	1

STOP

Kinect Color

1920x1080 0.16X 32-bit RGB image 123,129,121 (0,0)

Kinect Depth

512x424 0.54X Unsigned 16-bit image 1922 (206,422)

Background Image

Object Image

Live position P

350.0090	X
-0.0052	Y
450.0033	Z
179.9941	Rx
-0.0053	Ry
90.1665	Rz
5	Figure

HAND CONTROLS
Hand Command
Takktile Off
Do Hand Command
Close to Limit
Release

KINECT CONTROLS
Image index
999
Save Images to log folder

ROBOT CONTROLS
GoTo Home
GoTo Goal
GoTo Prediction

Live position T

350.0090	X
-0.0052	Y
450.0033	Z
1.0000	Ox
0.0029	Oy
0.0001	Oz
0.0001	Ax
0.0001	Ay
-1.0000	Az
5	Figure

Goal Area

500.0000	X
200.0000	Y
150.0000	Z
180.0000	Rx
0.0000	Ry
90.0000	Rz
5	Figure

GOTODIRECT

664.0837	X
122.9947	Y
131.9452	Z
-174.5878	Rx
1.4241	Ry
125.3739	Rz
5	Figure

CNN CONTROLS
Prepare Images for CNN
Load Prediction

CNN Depth 227

CNN Color 227

Figure 4.7: Figure showing the program used to predict using CNN.

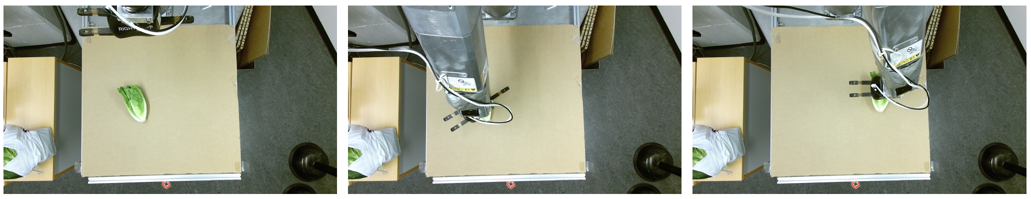

(a) Initial image (b) Grasping image (c) Release image

Figure 4.8: Example of successful grasp made using prediction from CNN.

per pose for grasping. This can now be confirmed as we were able to generate models that took data from both colour and depth image, with colour being mainly used for robustness. The features extracted from the images were able to be used in order to generate an SVR model, and using this model, it was possible to generate successful grasps.

Continuing with **Hypothesis 2**, *A 6-degree of freedom (DOF) gripper estimation is more accurate with RGB-Depth (RGB-D) images as input to a Convolutional Neural Network (CNN) architecture then a Support Vector Regression (SVR).* Once satisfied with the methods were the depth and colour images were analysed and various feature extracted from them it was time to make things a bit harder. After some testing it was determined that using the whole depth or coloured depth image would be more work than it was worth, as it included allot of excess and useless information. When instead the model only received the specific area surrounding the salad, and was set to predict a relative 6-DOF, the model could be made smaller, and therefore work faster.

By deciding to make use of only the cropped are area of the salad, the model did not have to first locate the salad, instead it was able to only focus on the main task, generating a 6-DOF.

4.4.2. Data augmentation

The best models for both SVR and CNN versions were the ones that were not augmented in any way, this is interesting and might be due to the fact that with such a small dataset, the augmentation does more damage then good. The goal of data augmentation is to increase the number of examples when generating a model in order for it to be able to predict any form of input within the given parameters. Therefore, since the sample size was so small, only 500 images, as opposed to the several tens of thousands that are used in some networks. Augmentation can have given to many theoretical inputs, that forced the model to train on stuff that would not occur. Leading to the model having to generalise to much, and therefor loosing accuracy when given validation data.

5. Conclusion

This chapter takes a look at conclusions in regards to the hypotheses, as well as key results and any future work that is possible from the results.

5.1. Conclusions in regard to hypotheses

Hypothesis 1 *Given data gathered from a RGB-Depth (RGB-D) image it is possible to estimate a 6-degree of freedom (DOF) gripper pose for grasping.*

In the end it was possible to estimate a 6-DOF pose for grasping salad using data extracted from the colour and depth images. The initial attempt using only depth images did not work as well as the attempts made using both colour and depth. This was due to the various shapes of the salad, and there not being enough information in just the depth images to repeatedly rotate the salad correctly. Once the data from the colour images was also used, the model was able to perform much better.

Hypothesis 2 *A 6-degree of freedom (DOF) gripper estimation is more accurate with RGB-Depth (RGB-D) images as input to a Convolutional Neural Network (CNN) architecture then a Support Vector Regression (SVR).*

By using the raw images, or parts of them it is possible to estimate a successful 6-DOF pose. This is good news as finding the correct features to use is not always as easy as it seems. Salads are rather simple, as they have an easy shape to work with, but for other objects, this might not always be as easy. Therefore being able to use some machine vision to locate the general area of the object, and then allow a CNN to learn the features it needs in order to make good predictions is the better option. This also means that time is not spent trying to figure out what features need to be chosen, as this is done within the program.

5.2. Key Results

The results show it is possible to generate models using RGB-D images, using either the images, or features extracted from them. Both types of models achieved an accuracy above 70&, when predicting the position of a gripper in order to make a grasp, with the CNN model achieving 78% accuracy. The CNN model also ended up being much quicker to generate than the SVR model.

The SVR model, using its 12 extracted features, is able to make a prediction on how to position the gripper, as well as how to rotate it so that it is able to close and grasp the salad. The SVR model also made predictions as to how tightly to close the fingers in order to make the grasp. These pressure predictions did not achieve the same results as the 6-DOF, their predictions being to low for any grasp. This meant the the pressure instead had to be hard coded, using a value gathered through trial and error. The reasoning behind this lack of success at predicting pressure, can be attributed to the teacher not having a method for telling how hard a grip is.

When generating the dataset, there is no force feedback, the only indicator that a grip was good was that the gripper was grasping the salad, as well as a readout on the pressure at each finger. These 9 pressures for each finger were then converted to an average value. Since the fingers could not uniformly distribute this pressure around the object, only a few of the sensors received a high pressure reading, causing the average to drop. It is also worth noting that salads should not be crushed when handled, so the grasp was not very hard to begin with, instead it was just enough to get a good grip.

The CNN model is extremely adept at generating its prediction. It does not take many epochs for it to produce good results. This means that training a model is done in less than 10 minutes, often less, as opposed to the 12 to 15 hours it takes to train the SVR models. This is good when testing various parameters, as new models can quickly be generated, with the resulting models either tested on salads, or compared to other models virtually.

5.3. Further Work

Currently the systems built for predicting 6-DOF pose estimates using the models generated only work on a static grasp area. A future goal is to be able to implement the system with a moving conveyor belt, as in a food processing plant. There the salad, or other object would constantly move along, and the system would have to take this into account when generating its 6-DOF. This improved model would also have to handle multiple salads at the same time, as they wouldn't be placed one and one on the conveyor, instead arriving in bunches and groupings.

One interesting future adaptation would be to see how much training data is needed in order to adapt the model, or train new models that could grasp multiple kinds of objects instead of only one type. With the SVR model, this would mean locating and generating the same features for a new object. The CNN models on the other hand only require the same image input, this means less time is spent figuring out how to gather the features.

When generating the training data, had there been a possibility for the trainer to receive some sort of force or haptic feedback, it might have been easier for the model to predict the finger pressure. Without any indicator of how hard the gripper is grasping the object, it is difficult to tell how hard it is actually gripping, and this uncertainty transfers into the model and predictions.

With the way the gripper is connected to the main program it is not that simple to control all aspects of the hand. The gripper uses ROS to connect to the rPi-3, by connecting it directly to the main program without this intermediary, it is possible to get more control over its movements. This would mean that it would be possible to gain control over individual fingers when closing towards a grasps, or receive feedback from the hand while executing a command.

Acronyms

CNN Convolutional Neural Network

DOF degree of freedom

IP Internet Protocol

LabVIEW Laboratory Virtual Instrument Engineering Workbench

NN Neural Network

RGB-D RGB-Depth

ROS Robotic Operating System

rPi-3 RaspberryPi-3

SVR Support Vector Regression

TCP Transmission Control Protocol

Bibliography

[1] Denso vs 087. https://www.densorobotics-europe.com/en/product/new-vs-087.

[2] Kinect v2. https://developer.microsoft.com/en-us/windows/kinect/hardware.

[3] Labview. http://www.ni.com/en-no/shop/labview.html.

[4] Matlab. https://se.mathworks.com/products/matlab.html.

[5] Python. https://www.python.org/.

[6] Raspberrypi. https://www.raspberrypi.org/.

[7] Reflex hand. http://www.labs.righthandrobotics.com/reflex-hand-1.

[8] Right hand robotics. http://www.righthandrobotics.com/.

[9] Robotic operating system (ros). http://www.ros.org/.

[10] Sixense stem motion tracker. http://sixense.com/wireless.

[11] Tensorflow. http://www.tensorflow.org/.

[12] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

[13] Hornby, G. S., Globus, A., Linden, D. S., and Lohn, J. D. (2006). Automated antenna design with evolutionary algorithms.

[14] iProcess. http://iprocessproject.com/.

[15] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition.

[16] Samuel, A. L. (2000). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 44(1.2):206–226.

[17] Thompson, A. (1997). *An evolved circuit, intrinsic in silicon, entwined with physics*, pages 390–405. Springer Berlin Heidelberg, Berlin, Heidelberg.

[18] Tsai, R. Y. and Lenz, R. K. (1989). A new technique for fully autonomous and efficient 3d robotics hand/eye calibration. *IEEE Transactions on Robotics and Automation*, 5(3):345–358.

[19] Veličković, P. Deep learning for complete beginners: convolutional neural networks with keras. https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html.

A. Video

Due to the specific hardware dependencies of the model, a video was edited together showing various clips of the system. This video can be found on youtube by following the link: https://youtu.be/MX9HFz827ag.

In the final weeks before the assignment was due, the main workstation suffered a minor malfunction. We are unsure what has has, as everything still works, although the working theory is that the power supply has become damaged. This means that when running some programs, the whole system becomes very slow. It is still possible to train models, as well as make predictions, but teleoperation has become unfeasible. This is most likely due to how the program connecting the remote control, robot, camera and hand is constructed, making use of several loops. This means that when trying to teleoperate, the refresh rate drops to levels where it takes seconds for a loop to complete, instead of milliseconds.

Teleoperation
This clip was captured captured during the construction of the first dataset in December 2016. It show me holding a black controller. Movements made with it are telegraphed to the hand, allowing for an intuitive control. I move the gripper to the salad, close it using buttons on the controller, and then activate an automated sequence, that moves the gripper to a predetermined spot, opens the hand, before returning to its resting home position.

Generating a CNN model
This video shows what is printed to console when generating a CNN model, as well as the tensorboard. The tensorboard shows two graphs, Absolute Distance and Loss, with each datapoint indicating the results of one training batch. The Absolute Distance is the Eucledean Distance between the prediction and correct answer. While Loss is the sum of errors in each batch. Once the model has been trained, a regression plot is made for each variable. This allows us to see how well each was predicted, the goal being a

diagonal line indicating the prediction and label are in sync. It also allows for comparison between models.

Grasping using CNN
There are two clips showing the complete procedure for grasping a salad using CNN. As well as one giving a closeup of the grasp. In order to make a grasp, the salad is placed within the grasp area and images are recorded using the LabView program. These images are then processed, the depth image is given colour by mapping what colour pixels correspond to what depth pixels. The depth image is also used to crop out the salad, re-sizing it to fit within a square, with zeroes or black as the background for depth and colour images respectively. The location that the crop was made at is also recorded, as this will be used to convert the models "relative" 6DOF, into a 6DOF within the robots coordinate space. Once the depth and colour images have been cropped, a python proram is used to feed the data into a CNN model, and a "relative" 6DOF prediction is made. This in turn is converted into the correct 6DOF. Then it is back to the LabView program, where the prediction is loaded, and the robot moved. The clips showing the grasping and movement of salad are filmed simultaneously, and later synchronised with button presses.

Object-in-hand Calibration
This final clip shows how the calibration board has been attached to the robot, and is moved around to different positions. At each position the location of the board is recorded by saving an image, as well as the robots position. The toolpoint used for this calibration is located in the intersection between the two top leftmost black squares. With the checkboard pattern it is possible to locate this same point in the images taken, allowing for a transformation matrix to be calculated.

B. Salad Grasp Dataset

This appendix contains information about the salads used when generating two datasets of salad grasps.

B.1. *Salad-1* dataset, generated 06.12.2016

The first set of grasps was created from 6 salads, their weight and length being recorded and shown in table B.1. An initial 50 grasps were done on each head of salad. This resulted in a data set of 300 grasps.

An additional 25 grasps were made to fill in any areas where there was less coverage then desired. The salads used for these augmenting grasps where chosen at random. With these additional grasps, a complete data set of 325 examples was available for model training and testing.

Table B.1: Table showing the weight and length of each salad used during the generation of the *Salad-1* dataset.

	weight(g)	length(cm))
1	161.72	14.0
2	154.44	16.0
3	161.54	17.0
4	152.75	20.0
5	219.89	20.5
6	202.04	20.0

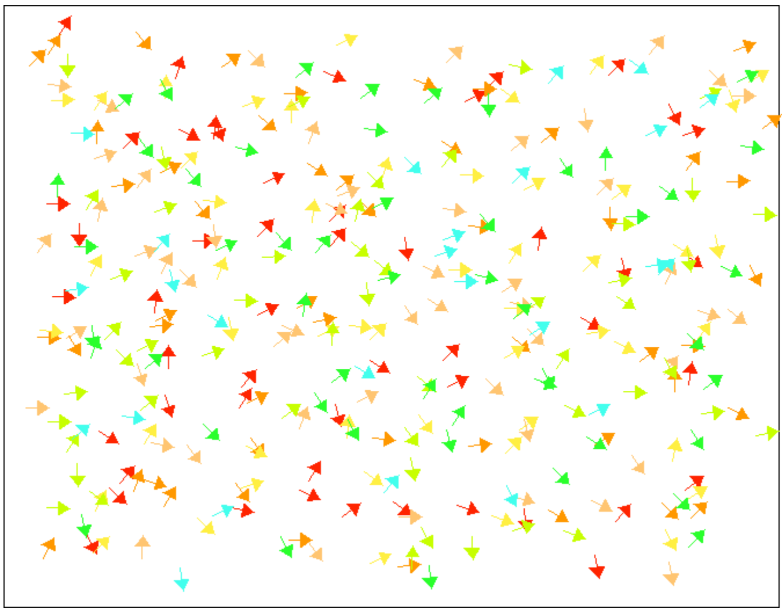


Figure B.1: Position and orientation of grasps when generating the *Salad-1* dataset, colours differentiate between salads

B.2. *Salad-2* dataset, generated 28.03.2017

In order to increase the size of the set of data available for model training and testing, an additional 4 salads were bought. The weight and length of each salad was recorded as shown in table B.2. These salads where used to generate 200 more grasps of 50 each. This allowed for the total number of examples to increase from 325 to 525. As well as increasing the variation of salads used, from 6 to 10.

Table B.2: Table showing the weight and length of each salad used during the generation of the *Salad-2* dataset.

	weight(g)	length(cm))
1	203.36	20.5
2	159.36	20.1
3	238.71	17.6
4	195.51	16.3

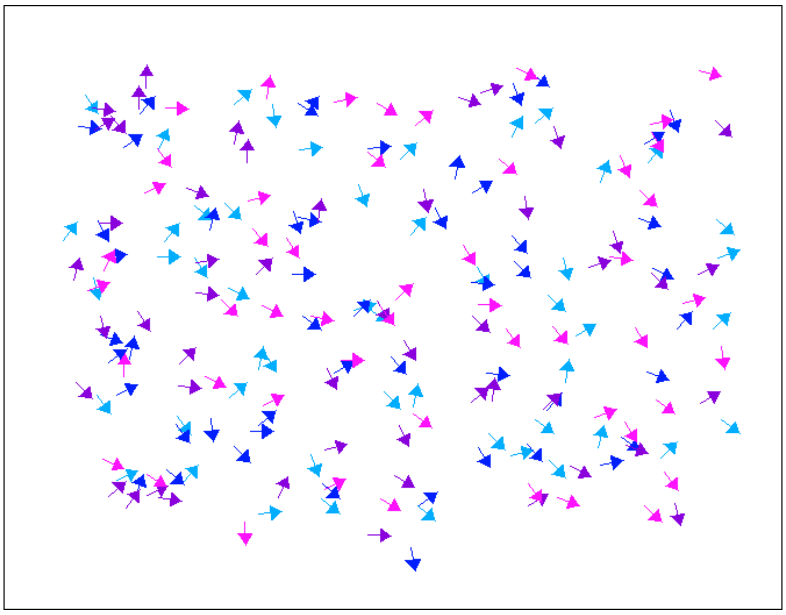


Figure B.2: Position and orientation of grasps when generating the *Salad-2* dataset, colours differentiate between salads

B.3. Merged dataset

Combining *Salad-1* and *Salad-2* into one large dataset produced a sample pool of 525 different grasps. The distribution of these grasps can be seen in figure B.3.

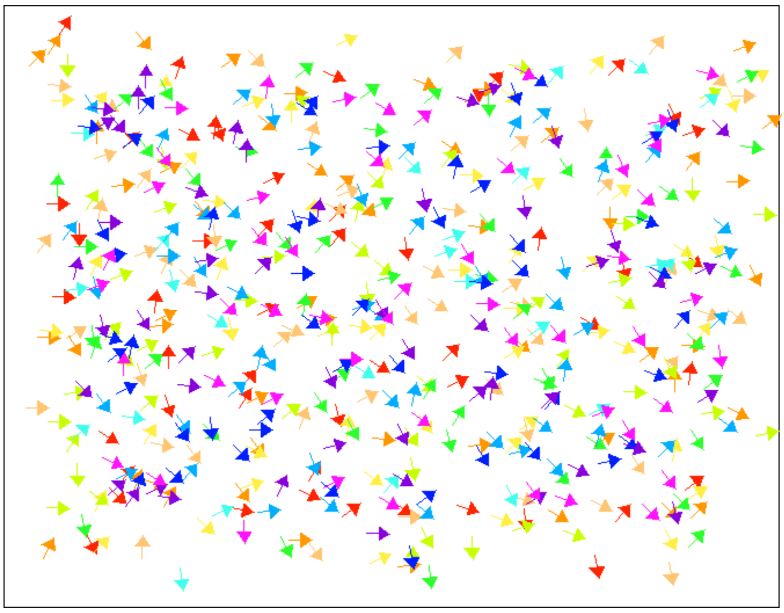


Figure B.3: Position and orientation of grasps in merged dataset, colours differentiate between salads

C. Transformation Matrices for Calibration

The transformation matrices from object-in-hand calibration. The matrices indicate where the lens of the Kinect2 IR camera is located in the Denso robots coordinate system.

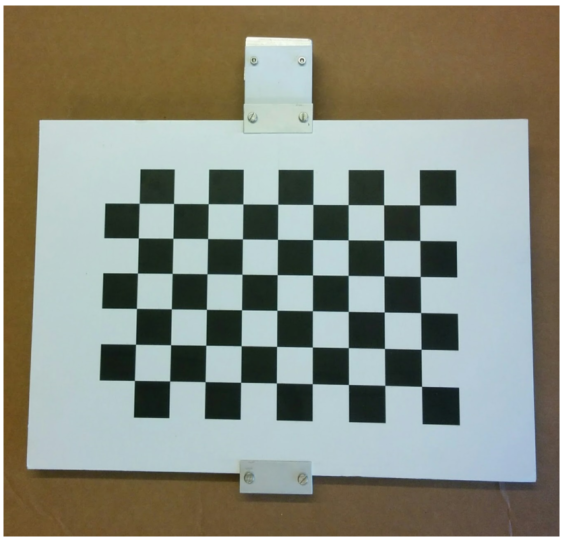


Figure C.1: The calibration board used when performing object-in-hand calibration

C.1. Calibration - 20161205

The calibration seen in table C.1 was performed the 5th of December 2016, in preparation of generating the *Salad-1* dataset.

Table C.1: Transformation matrix for Camera-Robot coordinate mapping. Calibration performed 2016.12.05

0.00237327	0.998577	-0.0532766	0.599985
0.999989	-0.00214866	0.00427278	0.0398296
0.00415222	-0.0532861	-0.998571	0.992112
0	0	0	1

Average Rotation Error (deg)	0.4651
Average Position Error (mm)	2.9511

C.2. Calibration - 20170328

The calibration seen in table C.2 was performed the 28th of March 2017, before the increased dataset *Salad-2* was generated.

Table C.2: Transformation matrix for Camera-Robot coordinate mapping. Calibration performed 2017.03.28

0.00988537	0.998723	-0.0495354	0.596485
0.999951	-0.00992022	-0.000457821	0.0401817
-0.000948639	-0.0495285	-0.998772	0.983574
0	0	0	1

Average Rotation Error (deg)	0.6173
Average Position Error (mm)	3.9686

C.3. Calibration - 20170620

The calibration seen in table C.3 was performed the 20th of June 2017 prior to testing a CNN model on 10 salads.

Table C.3: Transformation matrix for camera-robot coordinate mapping. Calibration performed 2017.06.20

0.00487752	0.99862	-0.0522947	0.597984
0.999987	-0.00480833	0.00144879	0.0406969
0.00119534	-0.0523011	-0.998631	0.991886
0	0	0	1

Average Rotation Error (deg)	0.4625
Average Position Error (mm)	3.4086

D. Tool-points

Tool-points are used to tell the robot where a predetermined point on an attachment is located in relation to the robots flange.

D.1. Calibration Tool-point

The tool-point in table D.2a is used when calibrating for object-in-hand. The position of this point, is the intersection between the two black squares in the top left corner, as seen in figure C.1

D.2. Gripper Tool-point

The tool-point in table D.2b is located 50mm beneath the centre of the grippers "palm". Since a quick connector is used to attach the gripper to the arm, the tool-point also negates an offset introduced due to it not being attached dead center.

Table D.1: Two tables describing the tool-points used for (a) calibration, and (b) grasping

	mm
X	-25.5
Y	-112.5
Z	155.7
Rx	0.0
Ry	0.0
Rz	0.0

(a)

	mm
X	-9.5
Y	-13.5
Z	145.5
Rx	0.0
Ry	0.0
Rz	45.0

(b)

E. Code

The zipped file containing the code contains many different files and folders. This is due to it containing all the LabView code required to run all the different programs, as well as the Matlab code to generate SVR and the pyhton code for generating and predicting CNN. This folder does not contain the dataset, being several gigabytes, although spreadsheets containing extracted data used when generating the models have been included.

E.1. LabView

LabView requires each sub-routine (SubVI) to be saved as a new file, this means that large programs may end up consisting of many files, the relative location of these files is also important for the program to run.

Gathering examples
The main program for controlling the complete system is called *RHand_-DENSO_KINECT_home_kinect_AXL.vi.* This program was used when generating the dataset, being able to control the robot using the STEM system as well as save images and data.

3D Visualisation
The program for visualising the different grasps over eachother was written in LabView, and can be found at "../iProcess/3D_Visualizer". There are several versions of the program, all of them begining with *RHAND_visualizer_*.

Automatic Grasping
When making predictions for either SVR or CNN the programs used are located in "../iProcess/Automatic". These programs connect to the Kinect, in order to capture the required image, then after having used a model,

move the gripper in order to grasp the salad, although they differ somewhat depending on what model they use.

Extracting Features
For the SVR models, features had to be extracted, this was first done using only the depth images, then later using colour as well. The SubVIs for doing this are located at "../iProcess/FeatureExtraction".

E.2. Matlab

The folder "../iProcess/Matlab/Code" contains the code needed to generate the SVR models. The folder "../iProcess/Matlab/Data" contains the data that the program uses in order to generate these models. This data has either been extracted from the images, or is gathered from the robot during a successful grasp

E.3. PyCharm

The folder "../iProcess/PyCharm/CNN" contains the code needed to run the CNN networks. This is both to generate the network, as well as make predictions using it. There is also a folder *cnn_prediction_live* for where data is saved when transfering it between LabView and the CNN model.