NTNU

Norwegian University of
Science and Technology

Identifying clothing articles by use of iterative classification

Simon André Johnsen Blindheim

Master of Science in Cybernetics and Robotics
Submission date: June 2018
Supervisor: Anastasios Lekkas, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Problem description

This thesis will address different aspects related to integrating computer vision and machine learning with a novel planning algorithm for controlling a physical manipulator in a simplified classification environment. The main objectives are listed below:

1. Detect initially disfigured clothing items by utilizing computer vision techniques.
2. Identify and calculate grasping points in space on ambiguously shaped clothing.
3. Unfold wrinkled or folded clothing items using a robotic manipulator.
4. Correctly classify spread out clothing items by use of a convolutional neural network.

Preface

This thesis aims to investigate how a convolutional neural network can be used to iteratively classify disfigured clothing items through the use of image preprocessing and physical interaction by a robotic manipulator, and concludes my Master of Science in Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU).

The inspiration for this work was sparked by my interest in the future of robotics within common households, i. e. the desire to one day have humanoid robots at home performing many of the every day tasks we find boring and tiresome today. The equipment provided by NTNU for use in this thesis includes a standard stationary computer and work station for software implementation purposes, a Kinect v1 camera for Xbox 360, and a Dynamixel manipulator. Software packages used with the Python 3.6 environment include the OpenKinect, OpenCV and Keras (Tensorflow) packages, as well as the Convolutional Neural Network module from my own specialization project and a separate manipulator control module independently developed by Øyvind Harding Gulbrandsen for his thesis work.

I would like to express my sincere gratitude to my parents and friends for their continous support throughout my years at NTNU, and particularly PhD candidate Joakim Tafjord for his help with proof reading this thesis. Furthermore, I want to especially thank my supervisor Anastasios Lekkas for all his encouraging guidance, as well as for the tremendous impact he's made on my choice of career path moving forward.

Trondheim, June 2018

Simon Blindheim

Abstract

Recent developments in the areas of computer vision and robotics have produced impressive systems capable of increasingly difficult tasks, but actually combining these fields together still has a lot of unexplored potential. In a few years time, it is not unlikely that a unified system utilizing the newest advances in both robotic vision and movement can be integrated onto a biped and two-armed android robot assigned to tidy our households.

Supervised machine learning is commonly applied in order to give machines the ability to assign labels to input data. Given a set of categories, the machine is tasked with identifying which class new observations belong to. This thesis focuses on the use of a previously implemented and trained convolutional neural network to iteratively classify disfigured clothing articles, with the help of a robotic manipulator and a combination of classic computer vision techniques. The clothing in question is observed initially deformed, i. e. wrinkled or partly folded on a table, and the proposed algorithm reads and processes image and depth sensor inputs in order to plan the appropriate moves which the manipulator will carry out.

The suggested final approach can to a satisfying degree classify and separate up to three different clothing items after a given number of iterations. As the robotic manipulator regrettably became inoperable under development, the direction of this work was eventually redirected to investigate the challenges encountered while processing and classifying sensor inputs alongside the intended manipulator control. The proposed scheme utilizes well-known computer vision techniques in order to plan the desired grasping points and movement trajectory for the manipulator, which in turn moves the iterative classification algorithm forward until a desired prediction goal is met.

The results produced in this work may be further considered for use with similar settings, such as housekeeping robots tasked with identifying and sorting or folding clothing in real environments.

Sammendrag

Norwegian translation of the above abstract

Nyere utvikling innen datasyn og robotikk har gjort det mulig å produsere stadig mer komplekse systemer som kan utføre mer avanserte og utfordrende oppgaver, men å kombinere disse feltene har fortsatt mye uutforsket potensial. I nær framtid er det ikke usannsynlig at et enhetlig system som utnytter de nyeste fremskrittene innen begge områdene samlet kan integreres på en tobeint og toarmet menneskeliknende robot, som for eksempel er gitt i oppgave å ta seg av rydding og vasking i hjemmet.

Menneskelig veiledet maskinlæring gir maskiner evnen til å kategorisere data. Maskinen får tildelt en samling med kategoriserte eksempler av det som skal klassifiseres, og den får utfra dette i oppgave å bestemme hvilke kategorier nye observasjoner tilhører. Denne oppgaven fokuserer på å utnytte et tidligere implementert og trent konvolusjonært nevralt nettverk for å iterativt klassifisere diffuse bylter med klær, ved hjelp av en robotmanipulator og en kombinasjon av klassiske datasynteknikker. Klesplaggene er i utgangspunktet deformert, som i rynket eller delvis innbrettet på et bord, og algoritmen leser og behandler bilde- og dybdesensordata for å planlegge passende kommandoer som manipulatoren så vil utføre.

Den foreslåtte tilnærmingen kan i en tilfredsstillende grad klassifisere og skille opptil tre forskjellige klesplagg etter et visst antall iterasjoner. Ettersom robotmanipulatoren desverre ble skadet under utvikling av kontrollmodulen, fokuserer denne avhandlingen heller på å undersøke utfordringene som oppstår under bearbeiding og klassifisering av sensordata på sin ferd fram til denne. Metoden benytter kjente datasynteknikker for å beregne foretrukne gripepunkter og tilhørende manipulatorbevegelser, som videre driver den iterative klassifikasjonsalgoritmen framover inntil det ønskede målet er tilfredsstillende oppfylt.

Resultatene som framkommer i dette arbeidet kan om ønskelig studeres videre for bruk med tilsvarende oppsett, som for eksempel framtidens renholdsroboter i alminnelige husstander med formål om å identifisere og videre sortere eller brette klær i den virkelige verden.

Contents

List of Figures

List of Abbreviations

ANN Artificial Neural Network

CNN Convolutional Neural Network

ML Machine Learning

DL Deep Learning

CV2 OpenCV Python module

Chapter 1

Introduction

The purpose of this thesis is to explore how convolutional neural networks can be used as an iterative classification tool together with classical image processing methods and trajectory planning of a physical robotic manipulator. This work thus focuses on combining the different field areas of computer vision, supervised machine learning and robotics in order to produce a complete system able to solve a specific task.

Figure 1.1: Imaginative illustration of an intelligent laundry machine [1]

1.1 Motivation

Automation of mundane and/or repetitive tasks has since the conception of programmable computers in the early 19th century [2] been an important motivation for many, and the inspiration for this thesis and similar projects [3] [4] [5] is no exception. Classical examples of such tasks are washing, drying and folding of clothes. Some may find these kinds of chores dull or too time-consuming by our modern standards for everyday work, and are by many undesired in our busy daily lives. By designing, constructing and programming robots to carry out these tasks for us, we can focus our attention on other things that may matter more.

Figure 1.2: The Honda Asimo robot [6]

Humanoid or android robots are a specific class of robots which are intended to mimic the shape and abilities human beings are capable of, usually having two arms, two legs, a torso and a head. The now famous Asimo humanoid robot [7] is an excellent example of such a robot, which in some distant future may be able to execute these complex tasks we humans perceive as straightforward and easy to perform. This thesis aims to investigate a few ways such a robotic system utilizing imaging sensors may be able to observe and recognize clothing items visually.

1.1.1 Similar works

Advanced robotics for domestic applications have arguably gained some momentum during the last decade [8]. As such, the particular task of folding clothes has been attempted to be solved by an increasing number of parties in recent years. Some examples are mentioned in the following sections.

Laundroid

The Laundroid [9] is a newly developed and commercialized clothing folding system resembling the form of a big wardrobe. The system is claimed to be using image recognition and robotic manipulators behind closed doors to detect the items to be folded, and produces stacks of completely folded garments within its shelves.

FoldiMate

The FoldiMate [10] system is a smaller and slightly different take on the same challenge, also patented and commercialized. It has multiple clothing hangers on its front, onto which the user hangs up the clothing to be folded. After being transported into its interior one by one, the items are folded by a simple and pre-determined folding process using simple mechanical actions. The result is similarly to the Laundroid also a tightly packed stack of clothing left on a small tray for easy extraction by humans.

(a) Laundroid [9] (b) Foldimate [10]

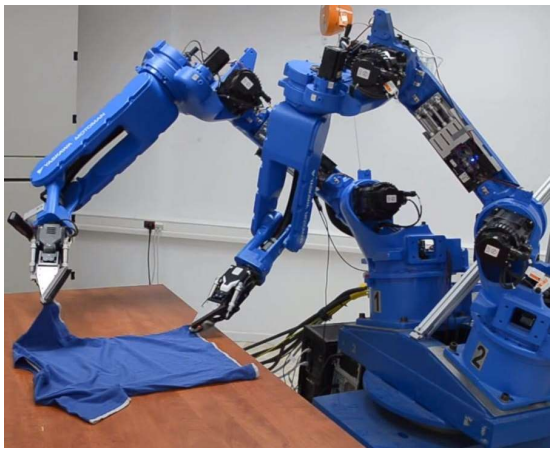
Figure 1.3: Examples of commercially available folding robots

CTU Prague, CERTH Thessaloniki and University of Oxford

More relevant to this project and unlike the previous examples, the work of a group of researchers at CTU Prague and CERTH Thessaloniki [11] [12] is based on a large industrial-sized robot with two arms and stereo cameras. This robot is able to pick up a piece of clothing lying on a table, and with the help of gravity, can repeatedly look at it and change its grip until it believes it has found the top two corners of the identified clothing in question. Lastly, the garment is spread out back on the table, and is subsequently folded into its final state.

UC Berkeley

The work lead by Pieter Abbeel et al. at UC Berkeley [13] [14] [15] is also more closely related to the work presented in this work, and is based on an open source humanoid robot called PR2, produced by Willow Garage [16]. This robot is built to function in real life environments, and has a smoothly designed human-like appearance with its two arms, a head and a large stereo camera as eyes. The unofficially but aptly named robot BRETT (Berkeley Robot for the Elimination of Tedious Tasks) has through the years become better and better at identifying and folding clothes. The latest advance on its capabilities was to incorporate deep learning and reinforcement learning into the mix, which enabled it to perform even more complex tasks based on the new feedback from the implemented neural networks.



(a) CTU Prague / CERTH Thessaloniki [11]

(b) UC Berkeley [13]

Figure 1.4: Examples of folding robots in development

1.2 Limitations

1.2.1 The manipulator

The most influential and deciding factor in regards to application and setup design througout this work was the size of the physical robotic manipulator itself. Though the initial approach was intended to be able to handle more general cases in real-life environments for clothing classification, the course was significantly altered after it was clear that the small manipulator available for this thesis was not capable of managing the more general use case.

1.2.2 The imaging sensor

The format and quality of input sensor data was also a primary concern while attempting the methods considered in this thesis. Two different cameras combining a color image and depth sensor were considered and later tested. The latter of those two was decided upon, and the Microsoft Kinect v1 camera [17] [18] was thus chosen for use with the manipulator.

1.2.3 Training and testing data

The challenge in finding sufficiently large databases with acceptable quality available for training CNN's is - as it was during the author's specialization project, from which the CNN module is adapted from - still present in this work as well. Accumulating enough data is often a tedious and lengthy part of the system development process in itself, and puts the entire classification performance under considerable limitations.

The image samples the network take as input to train on are manually downloaded from the internet in large quantities. The subsequent clothing to be tested upon are personally provided by the author, and thus the amount of achievable variation in the test set is likewise limited by the few available clothing items.

1.3 The approach

1. Construct a suitable physical setup and implement an image processing algorithm to detect clothing articles. The physical setup is handbuilt and custom-designed for use with the available manipulator and imaging sensors, and the proposed scheme utilizes well-known methods for feature extraction and detection.

2. Implement a simple program which takes the processed imaging as inputs and produces points in space as output, for which the manipulator would move to. A novel step-by-step procedure handles the extracted input information to plan the movement of the manipulator modeled in world coordinates.

3. Apply the thesis work of Øyvind Harding Gulbrandsen to control a Dynamixel-based manipulator for grasping and moving the clothing. *Though this was the original intent througout the work period, this objective was not completed due to the robotic manipulator becoming inoperable under development.*

4. Apply the specialization project work of the author to classify clothing articles by use of a Convolutional Neural Network. This module is adapted and modified to fit the use cases discussed in this thesis.

1.4 Overview

The theory behind some of the methods used in this thesis will be presented in Chapter 2, namely a few important concepts in the fields of computer vision, supervised (deep) machine learning, convolutional neural networks and robotics.

Section 3.1 lists the software and hardware used, and the designs and implementations of the two proposed approaches are detailed in Sections 3.2 and 3.3.

Next, the final results of each approach are presented in Chapter 4.

Chapter 5 discusses development concerns and their impact on the decisions made throughout this thesis, as well as a few suggestions for potential future work.

Lastly, Chapter 6 offers a closing remark summing up the main results, and an appendix demonstrating the steps needed to initialize the OpenKinect module is lastly added as easily accessible installation instructions for this work.

Chapter 2

Background

This thesis work deals with a relatively sizable combination of different knowledge areas, and consequently the background of each field is given but a brief introduction and overview of some of its core topics. The reader is thus expected to be familiar with the concepts of Computer Vision, Machine Learning and Convolutional Neural Networks, as well as some of the fundamentals of Robotics Modeling and Control.

2.1 Computer vision

2.1.1 Image processing

Manipulating images by geometric or intensity transformations are two of the main themes in image processing [19]. The different transformations and enhancements used in this thesis are presented next, in which most of them utilize OpenCV functions found in their respective documentation pages [20]. Most of the underlying introductory concepts regarding the digital representation of coloured and grayscale digital images as well as structuring elements (kernels) will be assumed known by the reader, as well as some of the most common and basic image processing functions found in the literature today.

Standard functions

Many straightforward image processing functions are used directly from within the CV2 library, and are considered unnecessary to be explained in detail here. These fairly simple functions include the following:

1. **Resizing**: Adjusts the size of the image by naming a tuple with pixel sizes.

2. **Cropping**: Extracts a subimage of the original image by explicitly specifying the window size and position to be targeted.

3. **Grayscale**: Converts a three-channeled colored image into a grayscale image, only containing single pixel intensity values between 0 and 255.

4. **Inverting**: In this thesis, this function converts the colors of a black-and-white image, making black pixels white and vice versa.

5. **Smoothing**: Applies a Gaussian blur to the image, it's intensity adjusted by changing the size of the kernel weighting the pixel values together.

The rotation transform

Rotating an image is performed by first obtaining the two-dimensional rotation matrix around an angle, aquired by the use of *cv2.getRotationMatrix* with the wanted angle and center of rotation as inputs. This matrix is then used to calculate new image intensity values by use of the *cv2.warpAffine*, which returns the new rotated image. See the OpenCV Tutorials documentation [20] for example usages and more information.

Perspective warp

Similarly to the transform above, the perspective warp in OpenCV utilizes the *cv2.warpAffine* function to produce the new image given its transformation matrix and area definition inputs. In order to get the skew matrix needed for the perspective skew transformation, one may use *cv2.getAffineTransform*, which accepts two lists of coordinate points. The indexed positions defined in the first list will be linearly moved to the positions defined in the second list by the corresponding indeces. Applying this spatial change of coordinates on all pixels results in a new perspective skewed image. See the OpenCV Tutorials documentation [20] for example usages and more information.

Gamma correction

Adjusting each individual intensity value in images across one or several channels is a powerful tool to modify images into suitable representations for further processing. The gamma correction method changes each individual pixel intensity in a grayscale image to the output of the following equation:

$$V_{out} = A * V_{in}^{\gamma} \tag{2.1}$$

Commonly A is set equal to 1, and setting gamma between 0 and 1 increases the brightness of the original image. Inversely, setting gamma to a value larger than 1 results in a darker image.

Spatial intensity adjustment

Simply adjusting each pixel's individual intensity value based on the relative position of the pixel is also a fairly common technique used in image processing. In this work, a self-implemented function such as this gives each pixel a value equal to one third, plus a third of the percentage of its relative horizontal position and a third of the percentage of its relative vertical position, times the original intensity value. As an example, this will set the optional corner pixel to one third of its original intensity, and the opposite corner will retain its full original intensity value. Examples using both these image enhancement funtcions and others mentioned in this chapter will be presented and further discussed in Chapter 3 Implementation.

Dilation

Edge dilation, applied by the use of *cv2.dilate*, is a simple method used on binary images - for instance Canny edge maps [21] presented in Chapter 2.1.3 The Canny edge detector. This function runs a moving kernel along foreground (white) image pixels in an image and 'expands' the contour by setting the neighboring background (black) pixels inside the kernel to also become white foreground pixels. The size of the kernel used may be every odd integer from 3 and above, and it can be applied repeatedly to further enlarge or swell up edge features in an image.

(a) A binary image (b) The enlarged dilated body

Figure 2.1: Example images showing before and after dilation is applied [22].

Histogram equalization

Images with a low range of pixel intensity values are often difficult for human eyes to discern details within. As one of several possible solutions to this problem, one may use histogram equalization to improve the contrast of said images. This method evenly spreads the intensity values present in a grayscale image across the whole possible range of colors from 0 (black) to 255 (white). Every pixel is assigned a new appropriate intensity value based on the position of the original vale on the color scale, determined by the image's pixel intensity histogram. This method is applied by the use of the *cv2.equalizeHist* function, see its documentation [23] for further reading on this topic.

(a) The original and somewhat faded image with a small range of pixel intensity values.

(b) The enhanced image as a result of equalization, with new intensity values evenly spread across the whole grayscale range of colors based on the original contrast of the image.

Figure 2.2: Example images of histogram equalization [23].

2.1.2 Image segmentation

Another subconcept related to the field of computer vision is the act of segmenting an image [24] into isolated parts with clear edges separating them, and is a subject recognized as one of the main attempted challenges in this thesis. Identifying different instances of objects in an image is the cornerstone of object detection, and is in some circumstances often extremely difficult even for human eyes. The border of where one specific item ends and where the background behind it or the outline of other objects starts may even not be clearly defined or shown in the image at all, further making it an increasingly challenging task.

This process of extracting the foreground and background as well as object instances information out of images may be achieved for instance by collectively using several different image enhancements and transformation schemes together. The methods to be used and further discussed in this thesis are namely the **noise removal**, **foreground extraction**, **border closing** and **region growing** techniques, as well as other self-implemented functions. These will however be presented in detail in Chapter 3 Implementation, as they were designed and developed specifically for the application considered in this work.

Figure 2.3: Examples of semantic image segmentation. Each nature image is followed by a few semantic segmentations at different levels. In general, each image is segmented into a small set of meaningful segments with considerable sizes [25].

2.1.3 The Canny edge detector

The now famous Canny edge algorithm [21] is a flexible and often adequate method for detecting visible edges in an image. This makes it a useful tool for quickly producing immediate feature extraction results with small efforts, for further application of more complex computer vision techniques. The algorithm is comprised of five subsequent steps presented below, and is applied simply by the use of the *cv2.Canny* function in this work:

1. The Gaussian filter is applied to smooth the image.
2. The intensity gradients in the image are calculated.
3. Non-maximum suppression is applied.
4. Double thresholding is applied to separate the remaining gradients into weak and strong edges.
5. Edges are tracked by hysteresis, suppressing all weak edges not connected to strong edges.

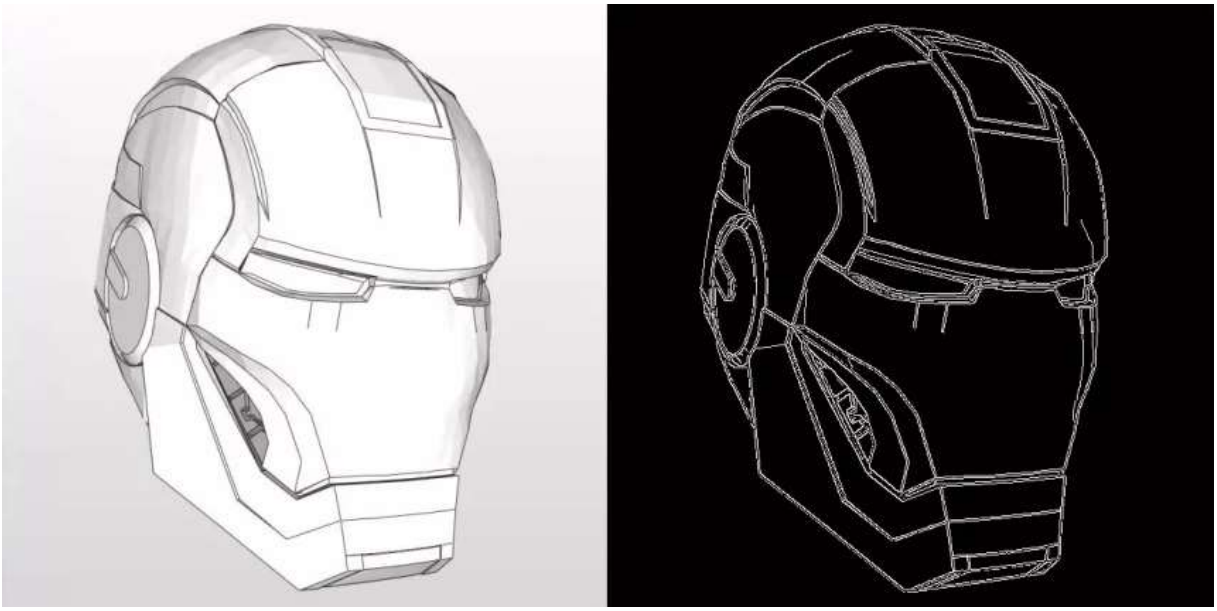


Figure 2.4: The original image is shown to the left, and the resulting Canny edge map is shown to the right. [26]

2.2 Supervised machine learning

The term machine learning is today accepted as the field concerned with giving computers the ability to learn without being explicitly programmed [27], essentially giving machines the ability to predict or decide probable outcomes in certain circumstances based on its observable inputs [28]. Machine learning algorithms have internal model settings and weights called *hyperparameters*, and the goal is to determine the nature and value of these parameters in order to maximize learning. Additionally, as the name suggests, supervised machine learning requires human supervision in the form of carefully labeled input data in order to incite the learning process of the ML algorithms [29].

2.2.1 Object detection

This concept has a wide range of different applications in every day life, ranging from anti-collision systems in vehicles to face-recognition used in anti-theft systems. There exists a number of diverse techniques to handle various use cases, both general methods and highly focused hard-coded approaches based on each use case. Image segmentation is as previously noted one such method for detecting objects in an image. However, one may also consider to attempt to "teach" machines the ability to distinguish patterns when given normal images directly as inputs without the use of other preprocessing techniques, hence why the act of detecting objects is in this thesis considered to be part of the machine learning topic. This particular approach will be further investigated in Chapter 3.2 The Sliding Window approach, and the more classical object detection methods using image processing will be revisited in Chapter 3.3 The Iterative Single Item approach.

2.2.2 Feature extraction

Another concept often associated with the fields of image processing and computer vision is to extract features from an image, as was indeed the primary intent behind the bulk of the previously presented image processing methods from Chapter 2.1.1 Image processing. This ties directly into the computer vision objective of differentiating useful or interesting information from irrelevant properties or noise, in order to further infer appropriate deductions about the elements depicted in images.

However, feature extraction is also the underlying practice happening behind the scenes when a ML algorithm is applied directly onto images with no additional preprocessing, as mentioned in Chapter 2.2.1 Object detection. Extracting features directly from images by the use of machine learning models and algorithms is in this work applied through the use of neural networks, and will be thoroughly addressed in the following sections related to classification and convolutional neural networks.

Figure 2.5: Example use of the local Hough transform for lane detection [30], a popular feature extraction method for detecting lines or circular features in images not discussed in this work.

2.2.3 Classification

The classification, or labeling, of observed inputs into fitting categories when labeled data is available, is a classical example of supervised learning. The already labeled data is called training data, and is provided to the machine in such a way that it can execute a learning algorithm in order to infer statistical connections between the data samples. The task is to produce a function that maps its inputs into the desirable outputs, which in this thesis is structured as a neural network of weights that will generate the predicted labels of clothing items.

The act of generating internal parameters or weights for a classifier and re-adjusting them over time, is referred to as the actual learning part of the approach. Most classifiers initiate its weights randomly. The machine then applies its estimated function onto its available training data, and checks its output against the desired (labeled) output. If there are disrepancies or errors between the produced and the desired output, the machine carries out a learning algorithm which uses these errors to adjust its hyperparameters berfore it tries again. In order to measure its own performance while training, the training dataset is split into a validation set and a training set. This validation set is kept separate from the training data and is only used to cross-validate the generalization error while training the classifier. After training for a period of time, the classifier is then fed another new and unobserved set of independent data called the test data, in order to predict which class or category the test data sample(s) belong to. The outputs predicted on the test data may further be analyzed by a human operator in order to measure its success rate or performance.

The learning algorithm

One of the most used learning algorithms found in the literature today is the *stochastic gradient descent* [28]. This algorithm is defined by a loss or error function that is used to generate the rate of how much the internal weight parameters should be adjusted each iteration. The learning coefficient multiplied by the error value is called the learning rate, and is set constant or dynamicly during run-time to control the rate of which the weights adapt to training and validation datasets. Moving backwards from the error of the output to the errors between each individual weighted input is called *backpropagation*, and is one of the key concepts for learning algorithms today.

2.2.4 The Convolutional Neural Network

Network layers

Artificial neural networks are feedforward networks connecting its input layer-wise to internal activation functions via weighted connections. Each layer of nodes may have different numbers of nodes and connections between them. Densely or fully connected layers are named so because they are connected to all of the nodes in the subsequent layer after it. Each layer extracts information (features) from the layer below and produces an output with a higher level of abstraction to the layer above. In the case of this thesis, the lower levels of representation describe the contours on clothes. Subsequent layers abstract these contours into smooth edges, lines and corners, and is structured into sleeves, collars, pockets, prints, texts and other familiar clothing attributes further up the hierarchy. Finally, the output from the top layer is a single type of clothing, which is the final classification objective.

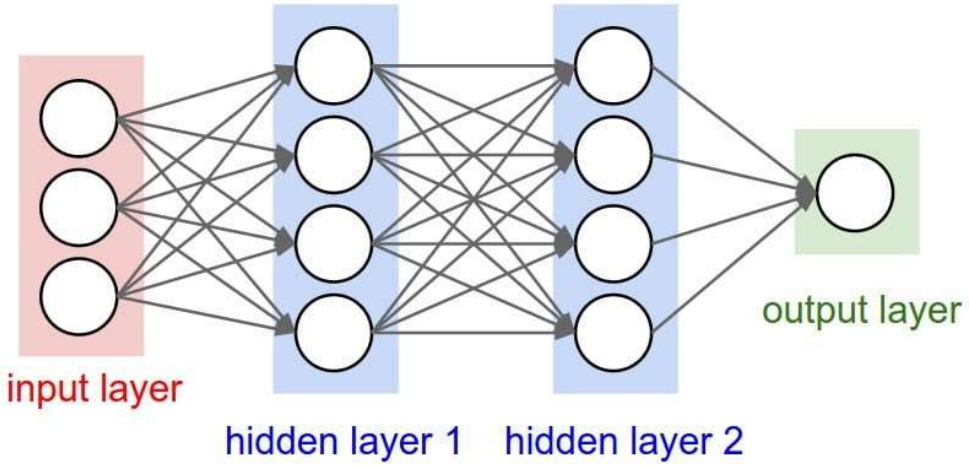


Figure 2.6: A simplified illustration of the layer structure of an artificual neural network. All of the different types of layers in between the input and output layers are often called hidden layers, as they are not 'observed' from the outside. [31]

In this thesis, the neural network has a special kind of layer as its input and second layer, called the convolutional layer. This layer is biologically inspired, in the sense that a moving filter kernel is applied to each input value in a 2D-structured map, and produces an overlapping output fed to the next layer. This can be viewed as smaller clusters of artificial neurons that make up reception fields with overlapping inputs and outputs using convolution operations. Input neurons in CNNs have three dimensions (width, height, and depth), shared

(replicated) weights, and is locally (not completely) connected. These properties wastly reduce the computational power needed for training and testing the network, which makes the approach convenient for this type of application.

In addition to the standard dense and convolutional layers, there are also several more types of layers mentioned in this thesis. Namely, the so-called hidden, max-pooling, dropout and "flattened" layers. A hidden layer is just the common term for all network layers not visible directly by the outside, i. e. all layers except the input and output layers. The flattened layer is simply a 1 by n vector containing all of the outputs from every node in the previous layer.

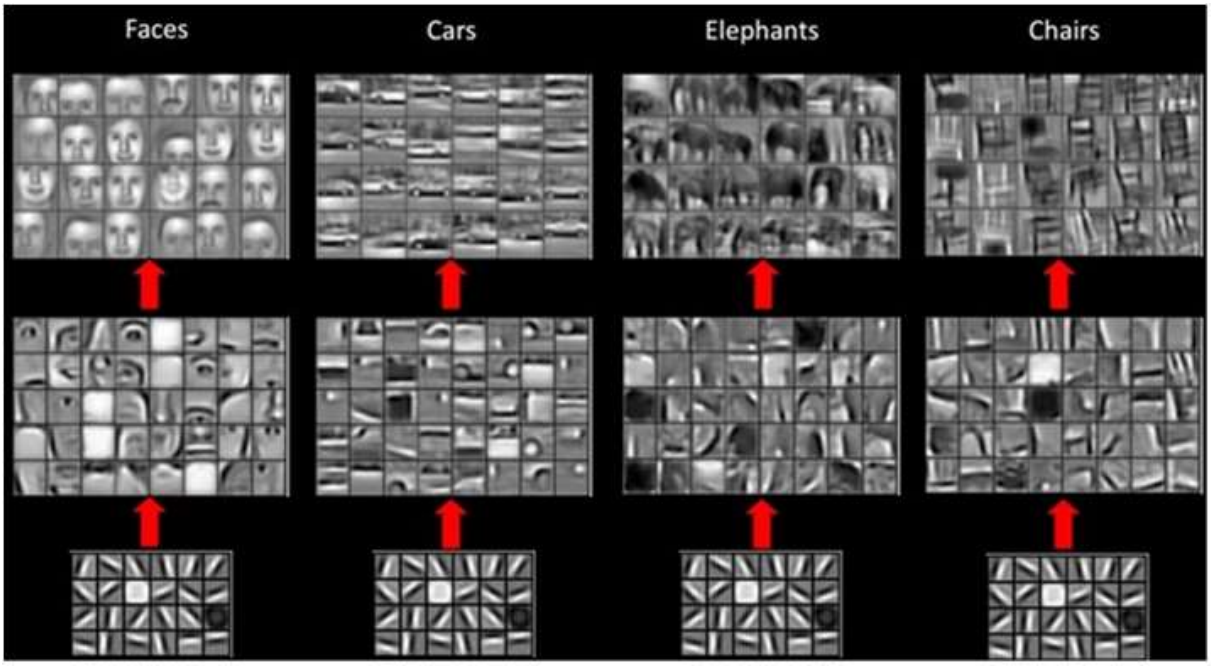


Figure 2.7: Simple edge features are identified in the bottom convolutional layers by the applied moving filter kernels, and are repeatedly "combined" by the above layer filters to represent more complex image features. The top layer lastly produces a simple class label, given its input weights filtered through each subsequent layer. [32]

Max-pooling layers are in this thesis included to down-sample the feature representations obtained in each convolutional layer, and may avoid some potential overfitting by increasing the feature abstraction. As well as possibly increasing the performance of the network model, its inclusion can also help reduce computation time and costs significantly. The large reduction in layer sizes gained from using this approach is generally shown to improve overall efficiency, and may be crucial for larger network structures to avoid significant overfitting.

Dropout layers also serve as an anti-overfitting measure. By simply setting a random number of layer weights equal to zero between feed forward computations, the network is forced to increase its redundancy as an arbitrary selection of weights will not always be active at each iteration. Thus, while back-propagating in order to adjust its weights during the training phase, the inactive activations will be "left out" of the learning process and other weights would be appropriately increased or decreased in their place. This somewhat counterintuitive but simple process may in that way help to reduce overfitting by a significant amount [33].

2.3 Robotics

Although the robotic manipulator was rendered inoperable under development and no replacement parts were available prior to the completion of this work, the concepts employed with regards to the application of the planning algorithm to physical hardware are still fundamentally important throughout this work. This section attempts to address some of the key points behind robotics modeling and control. The concepts mentioned in the following subsections are courteously adapted from select chapters in *Robot Modeling and Control* by Spong et al. [34]

2.3.1 Modeling and representations

Link angles and lengths

In the field of robotics, precise modeling of the physical relationships between components in an environment is of paramount importance and may itself often be one of the most challenging tasks during development. For rigid and distinctly interconnected bodies like manipulator arms, however, representing the different links and joints is somewhat simplified. For a manipulator containing only revolute joints, points and rotation angles around particular axes represents the variable parameters of each joint and specifies the lengths and orientation between these points in space. Thus, one may quite easily completely characterize a simple model for the manipulator from its base origin to its end effector. One such representation is the commonly-used Denavit–Hartenberg convention [35], shown in Figure 2.8.

Coordinate systems and frame transformations

In order to be able to use an established robot model and its parameters, one is also in the need of one or several sets of coordinate system to correctly relate the parameters to each other. These coordinate frames will also need to be defined relative to each other, and this is also one of the objectives of the previously mentioned DH convention [35]. The first and main frame of reference is often called the *world coordinates* of an environment, and its orientation and origin is decided upon with the intention to sensibly set the relative coordinates

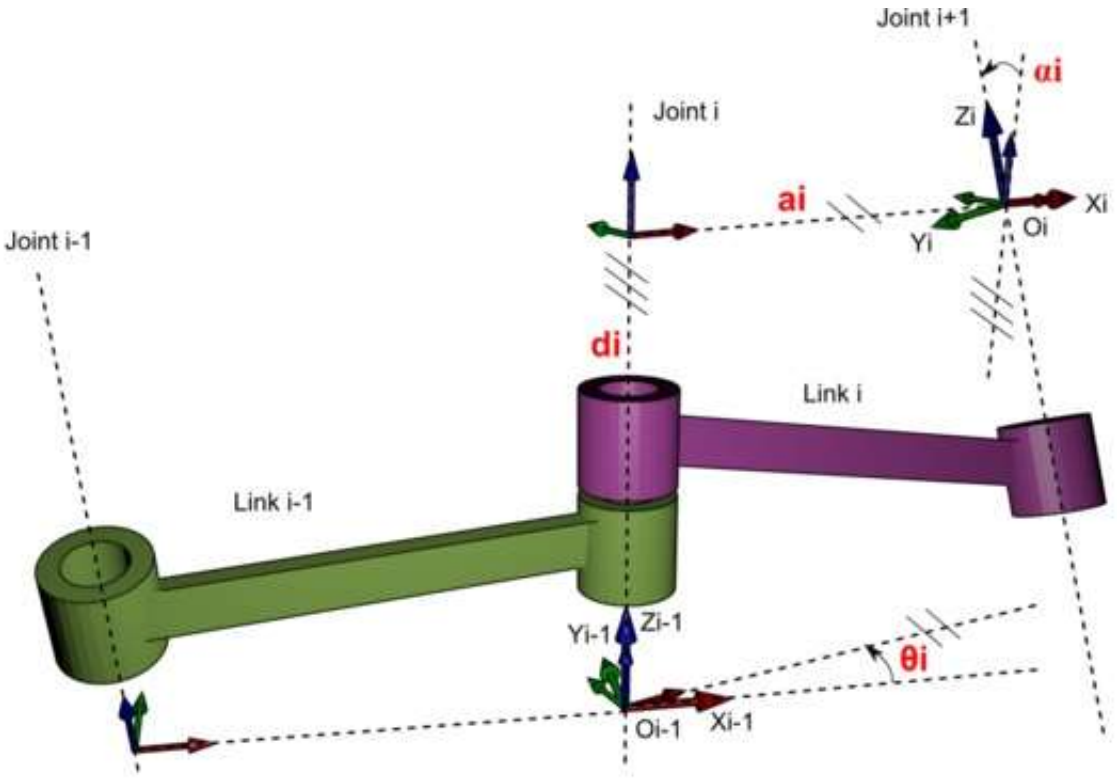


Figure 2.8: An illustration of the four different parameters and their respective link coordinate systems used in the Denavit-Hartenberg convention. [36]

of robotic manipulators, vehicles or other relevant objects given its context.

Next, subsequent frames of reference are given as rotation and translation matrices between points of reference in space, for each respective manipulator link or rigid body present in the environment. In this text, the world coordinates is the only reference frame actually considered. The manipulator control module is intended to imitate a so-called black box input output system, exclusively taking in only the desired end effector world coordinates as inputs. The resulting manipulator end effector orientation and position are thus assumed known at all times, as an open-loop scheme in which there's no sensory feedback given to the planning module. This particular well-known and common problem in the vast field of robotics is further discussed in Chapter 5 Discussion.

2.3.2 Inverse kinematics

Following the modeling of a manipulator's link and joint parameters, the reference frame and thus the orientation and the origin of its end effector is often the main concern in a given application. Applying the translation and rotation matrices subsequently after one another, one obtains the total transformation matrices given in relation to any and every manipulator linkages. This process of extracting the resulting coordinates of any joint from any other starting point in space, is called forward kinematics and is a fairly straightforwardly solvable problem related to robot dynamics.

However, the inverse problem is substantially more difficult. Given a desired end effector position and orientation, one is thus tasked with calculating the new joint parameters of every joint in order to achieve the correct joint control. This is aptly called inverse kinematics and is often one of the main challenges encounter in robotics, as the problem in some circumstances may have infinitely many solutions. The individual joint parameters can, depending on the type and size of the manipulator, often be controlled in several completely different ways in order to produce the same end effector coordinates. See Figure 2.9 for such an example. The mathematics behind these calculations and solutions will not be detailed in this work, and is appropriately left to the author of the manipulator control module.

2.3.3 Path and trajectory planning

Although merely performing the control movements of manipulator joints after the joint parameters are found seems simple enough, even this problem is an extensive one. Closely related to the inverse kinematics challenge, the manipulator controller essentially needs to calculate and execute countless of infinitesimal movements in every joint from its initial to its targeted configuration. This process is computationally expensive, and remains one of the toughest problems in real-time robotics control today. As for the other relevant aspects of robotics presented in this chapter, the topics of path and trajectory planning and control are also not investigated in more detail in this thesis.

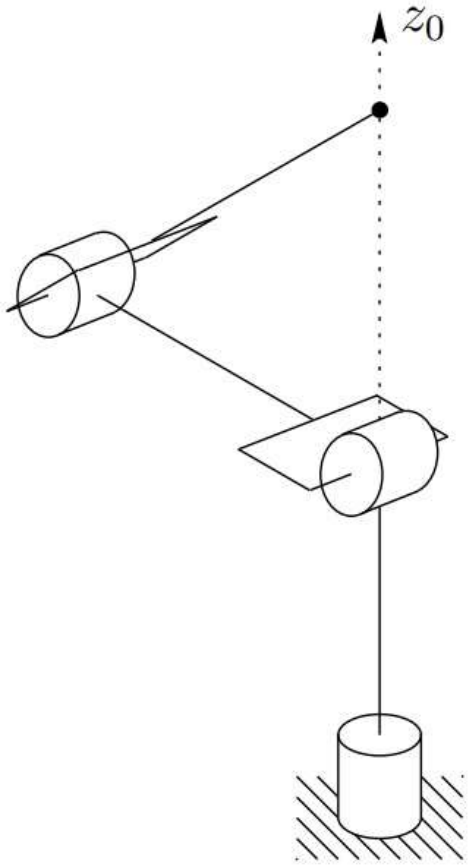


Figure 2.9: This illustration from the Inverse Kinematics chapter in Spong et al. [34] shows how a simple 3-linked elbow manipulator may encounter the problem of a singular configuration, for which the end effector wrist position has infinitely many solutions.

Chapter 3

Implementation

3.1 System structure and tools

3.1.1 Physical components

The imaging sensor

- **The ASUS Xtion Pro Live**: This camera was initially considered for use but was eventually laid aside due to consistent connectivity problems. The device was detected when plugged in to a computer but was otherwise completely unresponsive, and as such was ultimately assumed to be faulty.

- **The Microsoft Kinect v1**: The Kinect sensor for the Xbox 360 including a power adapter later replaced the Xtion Pro. The depth sensor along with a single color image provide a sufficiently satisfactory representation of the physical environment, which will be translated into world coordinates.

(a) The ASUS Xtion Pro Live

(b) The Microsoft Kinect v1 for Xbox 360

Figure 3.1: The imaging sensors considered

The robotic manipulator

- **MX-28 actuator**: Serves as the circular base of the robotic arm.
- **MX-106 actuators**: Two of these move the bottom and middle arms.
- **MX-64 actuators**: Used to move and rotate the forearm and wrist links.
- **AX-18A actuators**: A pair of these serve as the end effector gripper.

Due to an unfortunate accident during development of the Dynamixel module, the manipulator was internally damaged and some of the joint motors remained inoperable toward the end of the work period. Nevertheless, the computer vision and classification application is still developed with the manipulator in mind, and will assume it to be properly working each step of the algorithm. Manually performing the grasps and movements by hand between each iteration will act as a substitute for the manipulator.

Figure 3.2: The manipulator

3.1.2 Operating system and development platform

The main part of this thesis work is implemented on the **Ubuntu 18.06** operating system, using **Python 3.6** distributed through the **Anaconda3 5.0.1** package.

3.1.3 Python packages

The Convolutional Neural Network module

This module will be properly described in the Chapter 3.2.3 The CNN module, and is adapted from the author's specialization project.

The Dynamixel module

This module is solely the work of *Øyvind Harding Gulbrandsen*, but was unfortunately never used as a consequence of the accident mentioned above. It is fundamentally based on the Dynamixel SDK library [37] and provides an interface between the Dynamixel drivers for each motor and world coordinates, enabling users to specify points in space for which the manipulator will move to using inverse kinematics and trajectory planning.

OpenKinect

This package is named freenect for Python, and the only methods used were *sync_get_video* and *sync_get_depth*.

OpenCV

This package provides many of the computer vision methods used throughout this thesis.

Keras (Tensorflow)

The following features and methods were used from the Keras library:

- **keras.layers**: Dense, Dropout, Flatten, Conv2D, MaxPooling2D
- **keras.models**: Sequential
- **keras.utils**: np_utils

Others

In addition to the standard libraries included in Python 3.6, the following were also used:

- **Numpy**
- **Matplotlib**

3.2 The Sliding Window approach

3.2.1 The physical environment

During the first stages of the work period, considerable amounts of time was spent exploring different earlier works and promising approaches. The direction eventually chosen was however mostly inspired by the work of Pieter Abbeel et al. at UC Berkeley mentioned in Chapter 1.1.1 UC Berkeley. Though the robot developed by Abbeel et al. focuses on folding clothing from a fully spread out and natural state, this work aims to tackle the challenge which lies within determining the type of clothing from a disfigured patch of garments before one performs the actual act of folding it.

Thus, the initial environment would be for many a familiar situation: The robot would be looking down into a basket full of different clothing items to be sorted, straightened out and folded or hanged up before being put into a closet. Hence the task in this case would be to detect, isolate and sort the various clothing items into their respective categories. However, to simplify this environment to fit into a master's thesis work period, it was decided that there would only be a few clothing items present and they would be laying fully separated on a white flat surface.

Figure 3.3: Frustrated woman holding a heap of clothes [38]

3.2.2 The algorithm

Sliding windows and image pyramids

At first, the algorithm was proposed to be able to discern clothing stacked on top of each other by use of color segmentation. However, the primary task quickly became to enable the system to target specific areas of a larger picture in order to use the CNN, as it was already implemented as a separate module. This was decided to be accomplished by the so-called 'sliding window' approach [39].

This method extracts large numbers of smaller windows from an image with different sizes, often applied in order to detect specific items in a larger environment - as would be the case for this type of application. By specifying a window size smaller than the original image, one may extract overlapping smaller partitions by 'sliding' the window along the x- and y-directions of an image. One may also optionally adjust the step size for which the window moves each iteration, essentially altering the degree of overlap between the extracted image partitions.

As an example, one may specify the extracted window to be 1/4th of the size of the original image, and the step size may be set to a length of only 1/8th of the image size each step in both directions. This will thus produce a consistent overlap area of four times the original image area, with the exception of by the image borders. One often tends to empirically set these step values and window sizes customized to the specific application in question, to best fit the function it would serve with the least amount of time and resources spent extracting these smaller partitions.

Figure 3.4: The sliding window algorithm and the image pyramid performed on an image of a video game cover, courtesy of Adrian Rosebrock [39]

It is also common to extract partitions of different sizes for the same application, as it is not always known exactly how big something one's looking for actually is in a larger picture. One way to address this is to apply the sliding window method several times to the same image, but with different window sizes each time. Another approach is to use so-called image

pyramids [40]. This method applies Gaussian smoothing and downsizing to the original image several times to produce smaller and smaller sized images to which the sliding window method may be applied to. This serves as an alternative to changing the window size itself, and produces 'layers' of the image in different sizes. The top layers are the smallest representations, and the bottom layer is the original image - hence why it is called an image pyramid.

The combination of the image pyramid method and the sliding window produces stacks of partitions of the environment image, onto which one may use the standalone CNN module to predict if the partitions contain features similar to that of clothing present in the training database. Each of these partition predictions are then analyzed in relation to each other.

Estimating position coordinates

In order to make sense of the data produced by the sliding window predictions detailed above, one may try to extrapolate relations between the overlapping and/or connected image partitions by merging the prediction probabilites over larger subsections of the image. The merging could be a simple average of the predictions produced for every partition, but it could also be a more advanced statistical approach. Creating some sort of 3D plot showing the predictions across the x- and y-axes of the image would be helpful to pinpoint where the algorithm estimates a certain clothing to be perceived. The tallest spikes may then show a significant preference to one specific item, which one could further assume to showcase the position of such a clothing.

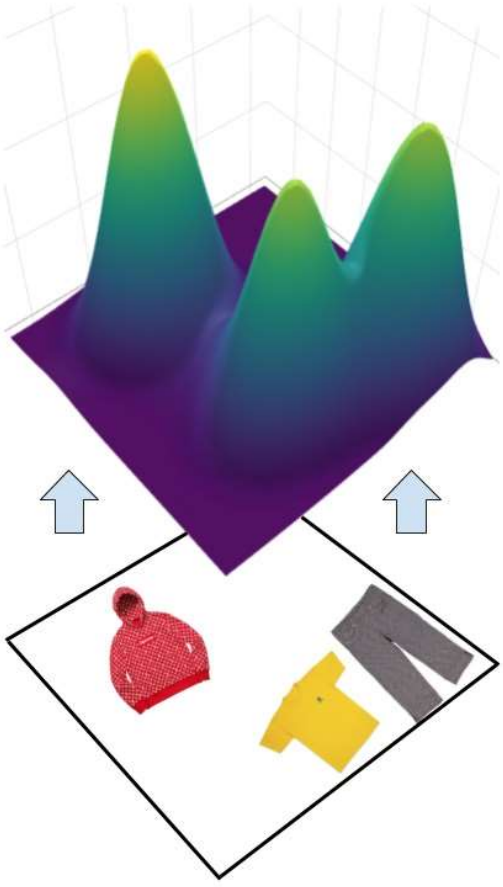


Figure 3.5: Example of the envisioned output of the prediction probability distributions of an environment containing three separated clothing items of different types. The simulated urface plot is retrieved from plot.ly [41]

Based on these overlapping estimated values, the algorithm could calculate an approximated position for the objects present in the image. These coordinates could furthermore be provided to the feature extraction and later planner modules, which together will produce more precise coordinates for which to guide the physical manipulator further on in the pipeline.

Calculating manipulator paths

Now that there's been formed some notion as to where interesting items to be handled may be located, the manipulator needs to be able to identify grasping points on these items and determine how to move to and from these points. Exactly how the algorithm computes these grasping points was not addressed during this stage of development, but will however be thoroughly discussed in Chapter 3.3.6 The grasp point ranking algorithm. Lastly, the standalone manipulator module calculates the manipulator movement arcs and its corresponding joint angles to move the end effector from point A to point B.

3.2.3 The CNN module

A summary of the implementation details regarding the CNN module from the author's specialization project is presented below.

Gathering data samples

500 images of the clothing categories 'pants', 't-shirts' and 'hoodies' were manually downloaded from the internet, resulting in 1500 images in total. Experimentation with the Canny edge detector resulted in a decision to delete and replace images which proved to produce unsuitable edge maps, as a consequence of complex or strange image patterns or formats. Most white-colored t-shirts were removed, along with some of the images with too complex patterns. Suitable training set images was decided to not be allowed to contain more than one clothing item in the same picture, the item was to appear in its natural undistorted state, and the background was quite strictly chosen to be of white colour only. A simple self-written function compared every image sample to one another and checked if there were identical copies present. As such, each training data sample was ensured distinct, guaranteeing non-repeating input samples.

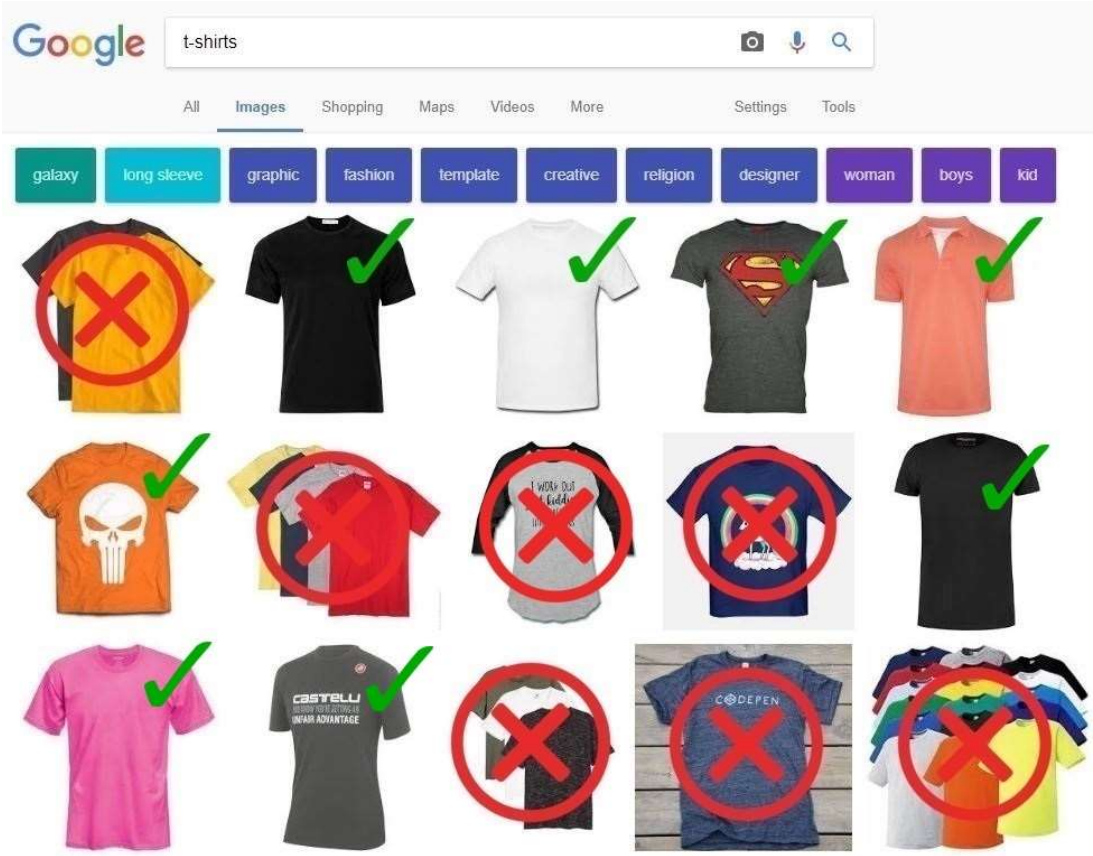


Figure 3.6: Examples of accepted and discarded images from a typical Google search

Generating training vectors

12 diverse images from each class were separated from the training data, chosen to be as challenging as possible for the neural network prediction algorithm within reasonable boundaries. Because of the fast and highly accurate training and testing results aquired very early in the project period, there was some doubt as to the legitimacy of the initial network training results. Ensuring that the results was 100% certain to be independent of earlier training sessions was achieved by taking pictures of a small selection of clothing articles from the author's own apartment, positioned on a white table in various neutral and distorted poses. As such, these newly self-produced images were guaranteed to not have been seen by the network or any other system before, consequently showing that the performance of the network was indeed legitimately genuine. Each training and testing image was loaded into memory, rescaled down to 28 by 28 pixels and converted to grayscale by utilizing the CV2 functions *imread*, *resize* and *cvtColor*. See the OpenCV documentation [42] for details regarding these functions.

(a) Left rotation (b) Right rotation (c) Upwards tilt warp (d) Downwards tilt warp

(e) Lighter brightness (f) Darker brightness (g) Bottom-left corner light source (h) Bottom-right corner light source

Figure 3.7: The 8 augmentation functions applied to the training data

Next, for each of the input images, several different image augmentations were applied to produce variations that were also added to the training data set, resulting in 4500 different images for each clothing category. These augmentations include two rotation functions, 'camera tilt' warping functions skewing the original image in two different ways, and four lighting intensity variation functions. See Figure 3.7 for example demonstrations of each image augmentation function, and a brief explanation to the theory behind them can be reviewed in Chapter 2.1.1 Image processing.

The originals along with their augmentations are then unraveled into standard 1 by n NumPy arrays and concatenated alongside their label number representing which folder and class they belong to. Next, the resulting feature vectors are written into two simple training and testing CSV files [43], producing a compact representation of the complete database easily stored and read by the network training module. Lastly, these 1 by n vectors are read into memory again and are transformed into the format which the network model accepts as input. See the Keras documentation [44] for more information.

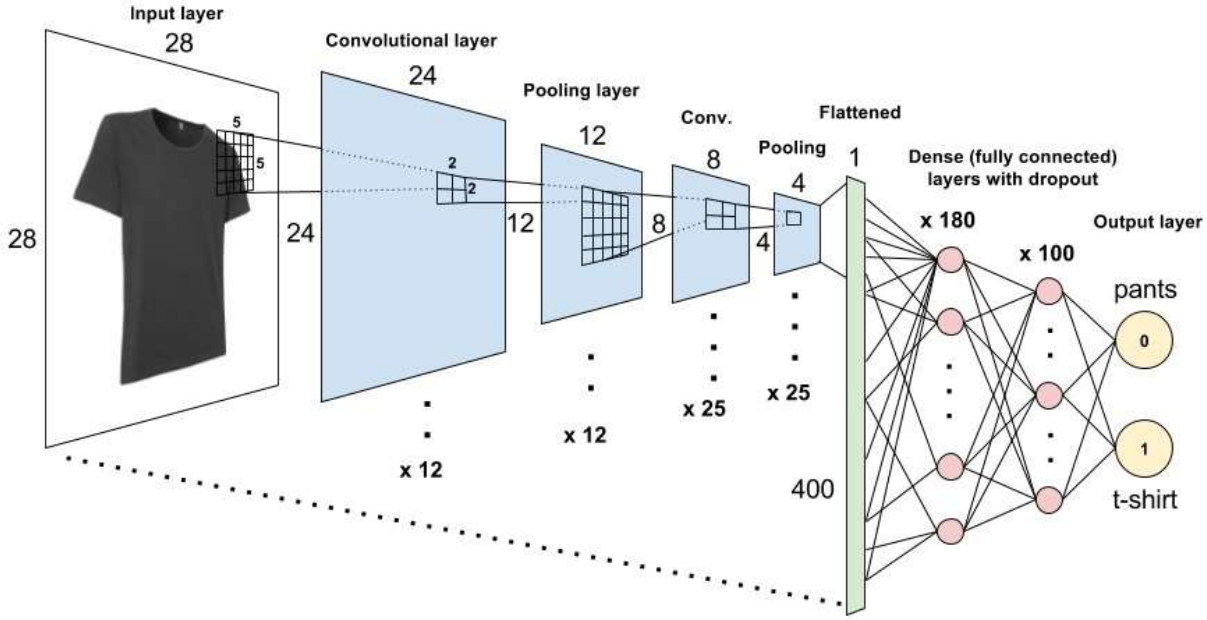


Figure 3.8: The LeNet5 network structure including dropout layers, for binary classification between two different clothing categories.

The network structure

It was decided that the quite simple LeNet-5 architecture [45] was to be used as the neural network structure, as it had been shown to produce quite good results for recent similar works. The sequential model [46] was initialized and all of the networks layers (including the dense, convolutional, flattened, max-pool and dropout layers) were added using the inbuilt model **Add** function according to the LeNet architecture, along with two additional dropout layers as detailed by Hinton et al [33]. The connections between the layers in the model are automatically handled by Keras, and as such the only parameters actually needed for an initial setup were the number of neurons for each layer, their activation function and their initialization mode. Values for these initial parameters were borrowed from a Keras LeNet tutorial [47].

Lastly the model was compiled, making it ready to fit training data and predict labels for previously unseen test data. The final structure of the network including its layer sizes and connections is shown in Figure 3.8. See the subsections of Chapter 2.2.4 The Convolutional Neural Network for closer details behind the convolution, max-pooling, flattened, dense, and dropout layers used in the network model.

Network training and prediction

The **network** module is initialized by first loading a database into memory using the previously detailed functions, and all of its settings are set depending on the type and size of the input data. Next, the network itself is created and initialized as described in Chapter 3.2.3 The network structure. By calling the **Network** methods simply named *train* and *predict*, one may easily train the network and predict labels for unknown test data.

The training itself is executed by the Keras *model.fit* function. This method takes in the lists of training data and labels, the *batch size*, the number of *epochs*, the *validation ratio*, and a boolean value *shuffle*, among other optional parameters. The batch size was set to 64, as it has been empirically shown to seem to have a general sweet spot around that value for this kind of problem. The size needs to be set such that the computations fit into memory, the training time until convergence is in a preferable range, and the generalization quality of the model is as desired [48].

The validation ratio was set to 80% training data and 20% validation data, as the model naturally showed tendencies to extreme overfitting due to the initial lack of cross validation in each iteration. As shown in Figure 3.9, the accuracies of the training and validation of network models should be somewhat close, and a large discreptancy between the two may indicate overfitting.

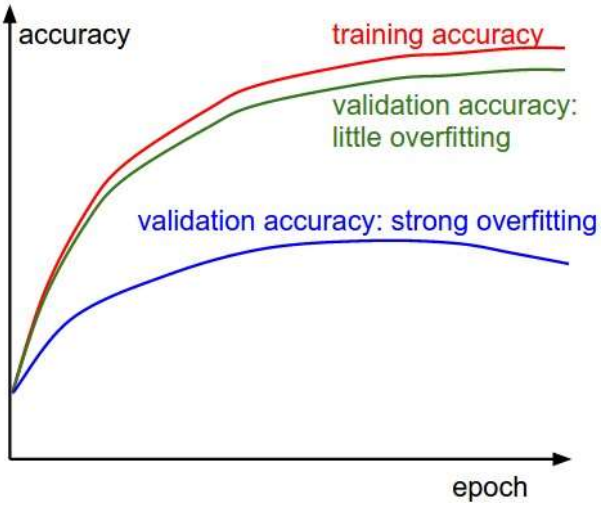


Figure 3.9: A graph showing the differences between training and validation accuracies in the case of overfitting [49].

The number of epochs was set to 10 during development, as the results from the author's specialization project (Figure 3.10) did not seem to show any discernible improvement past 10-20 training epochs while all of the augmentation functions were used on the training data. Also, the boolean value *shuffle* was by default set to **True** in order to shuffle the training data before each epoch. The Keras documentation [46] provides more in-depth details on these parameter settings.

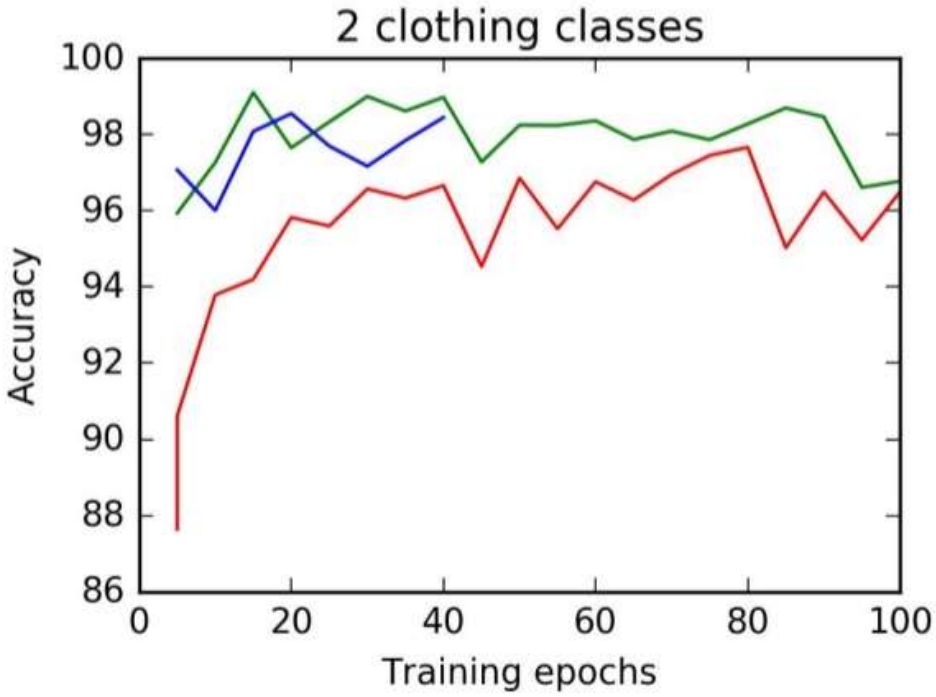


Figure 3.10: The red line shows the CNN's test certainty multiplied by its averaged test success rate against the number of epochs with 0 image augmentation functions in use. The green line shows the resulting performance including the 4 transformation augmentations, and the blue line has all of the 8 augmentations including the lighting variations enabled.

Prediction is then performed by the Keras *model.predict* function, which returns a list of prediction values for each test sample. This prediction list consists of values between 0 and 1, representing the distribution of certainty across all possible categorization classes for which the network predicts the sample belongs to. See the Keras documentation [46] for further information. Additionally, the highest percentage value for each classification attempt is compared to the user-defined test data labels in order to calculate the success rate and certainty of the prediction outputs.

3.2.4 Early classification performance

Presenting the results

The training and prediction results needs to be presented in a clear and simple representation. To achieve this, the original test images are read into memory, scaled down to 300 by 300 pixels and associated with each prediction result. Then, the percentages for each classification are overlayed onto a plain white background along with their corresponding class names using the OpenCV function *putText*. By concatenating the classification results text image to the original downscaled test image, one may easily read the prediction results for each sample. These images are then saved to disk by use of the OpenCV function *imwrite*, and may be viewed as a normal picture or added to text documents or pdf's as one pleases. See Figure 3.11.

(a) T-shirt (b) Pants

Figure 3.11: Examples of binary classification results

The 'background' training set

In addition to the CNN setup and training data described in the previous section, another classification category was added during this thesis work period. This was done as an attempt to handle the case in which the sliding window algorithm tries to predict labels for partitions of an image consisting only of so-called "background", i. e. partitions with no clothing items in them. As such, 500 images of different backgrounds were downloaded from Google Image search. Figure 3.12 shows a typical excerpt from this data set.

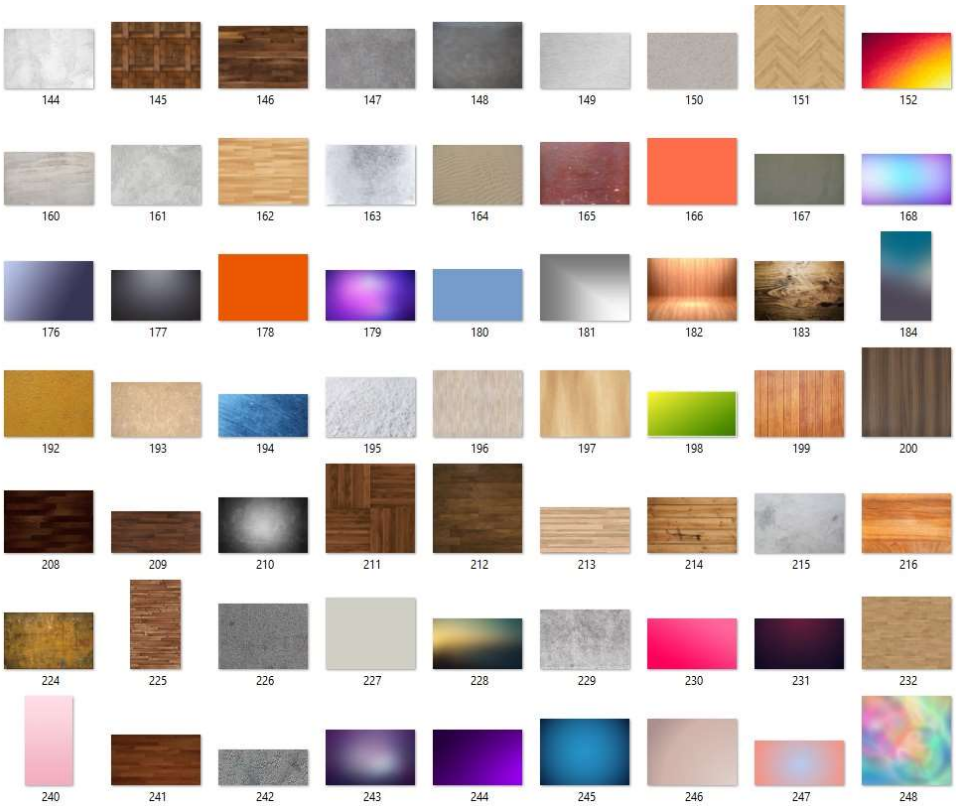


Figure 3.12: Examples of data samples from the 'background' category

The variation of the data samples was purposely attempted to be high during the collecting and download phase, for two reasons. Firstly, the initial intent was to make the prediction attempts on image partitions only containing parts of clothing items be able to handle noise to some degree. By aquiring background images of such a large variety, perhaps the CNN would predict any image that did not contain the specific features of one of the clothing categories to be a background image. Secondly, making the system robust enough to handle similar use cases as the one presented in this thesis would be desirable. The inclusion of background images showing wood patterns would as an example possibly make the system more able to handle situations in which clothing to be classified would be lying on wooden floors. The caveats and complications of the decision to make the background data set diverse to such a large degree is discussed in more detail in Chapter 5 Discussion.

Prediction errors

Two substantially large anomalies were discovered while testing the predictions of image partitions, as shown in Figure 3.13. One of the two t-shirt images used for testing was consistently classified as a hoodie, and the one hoodie present was classified as background. However, the image showing an entire table with random edges and features along the image borders was interestingly enough classified correctly with a high certainty. These results quickly prompted the re-evaluation of several aspects of the approach, and the differences between the training data and the actual testing use case became more apparent.

Figure 3.13: Examples of testing results produced after the inclusion of the 'background' category in the training data set

For one, every single one of the training data samples had strictly white backgrounds ecompassing the clothing items, whereas the use case had wooden floor patterned backgrounds. Though the background training samples were attempted to have a high degree of variation, it was suspected that the CNN over-emphasized the importance of the white border backgrounds in every clothing item data sample. Furthermore, the variation in the background data samples may simply be too diverse, and is consequently attributing clothing patterns and features to that of random background noise. It is also possible that there's a "Photoshop bias" present, in the sense that the data distribution of completely smooth backgrounds differ strongly from the rougher and more 'real' environments. These issues are also further discussed in Chapter 5 Discussion.

Reducing training data complexity

These results incited the assumption that the current training set for the background images category may be too general and complex for the present use case. This may heavily impact the performance, and may be the cause that makes the predictions as unreliable as they seem to be. The following was thus attempted in order to try to make the training data conform more to its actual use case:

The background training data set was reduced by removing some complex backgrounds, and images taken of a table were added along with 8 artificial augmentations. With this change, a new question arose regarding the validity of the training data samples when utilizing the same augmentation functions twice to produce training data. Only a few images were really taken of the actual environment and added to the data set as entirely 'new' images, the rest are essentially just enhanced copies of the one. Further copying 9 'copies' 9 more times by the same method will then result in as many as 81 copies of one single image sample, which in itself may lead to a significant increase in training bias.

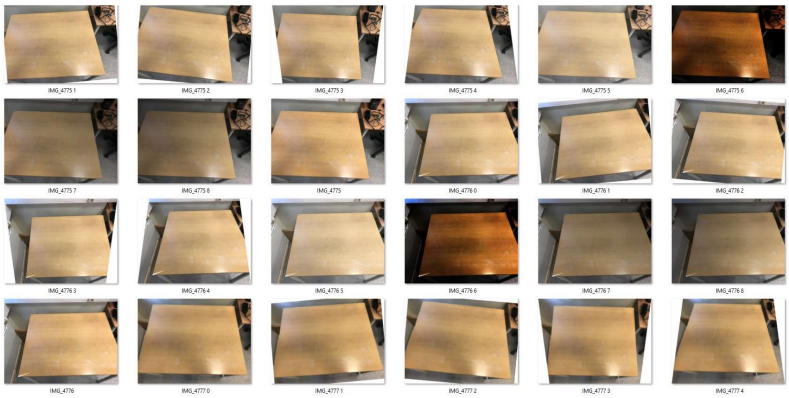


Figure 3.14: Examples of new additions to the background data set including their augmentations, replacing some of the too general and complex backgrounds present in the set

However, one may also argue that the different augmentations does not necessarily negate each other, and that a slight bias towards the actual use case may after all be beneficial to the application in this situation. As an example, performing the rotations on each background image introduce an artificial white border (which again will be rotated both back again and even further while generating the actual training vectors), producing non-identical images for every repeated augmentation. Nevertheless, as a consequence of other developments during experimentation, deeper investigation into this matter was not pursued any further.

Classification of larger environments

Considerable efforts were subsequently made to implement a prediction distribution map across a large image on which the sliding window algorithm was applied. All of the label predictions on each image partition are weighted together in order to produce a final classification probability for the entirety of the larger image. The resulting prediction values are shown in Figure 3.15, and some notable remarks may be made regarding these results:

The percentages indicating how much of the image areas that are 'just backgrounds', i. e. not containing any parts of clothing, seem to be correspond to the reality to some degree. If the prediction weighting had been further advanced to include less bias towards smaller image partitions in the lower parts of the image pyramid as an example, perhaps the results would reflect the reality to an even larger extent. After all, the partitions produced by the sliding window on layers near the top of the image pyramid contain much larger portions of the actual imaged area, and should be weighted accordingly.

Additionally, the remaining prediction values from the testing images for the pants and the pink t-shirt show some statistical preference to the correct clothing items present in the large images. However, there's also significant uncertainties apparent in the dark t-shirt sample. The reason for this behaviour may possibly be that it is a result of more sliding window area overlap containing only fractions of consequently wrongly classified clothing, unlike the pink t-shirt positioned closer to the top image border.

Yet the hoodie image interestingly shows a bias towards the hoodie and pants classes. It is not entirely unlikely that the sleeves and parts of the main body of the hoodie were consistently classified as pants, as the partition window slid across the clothing during runtime. This may also be a symptom of the unrefined system's inherent significant bias toward the lower image pyramid layers partitions.

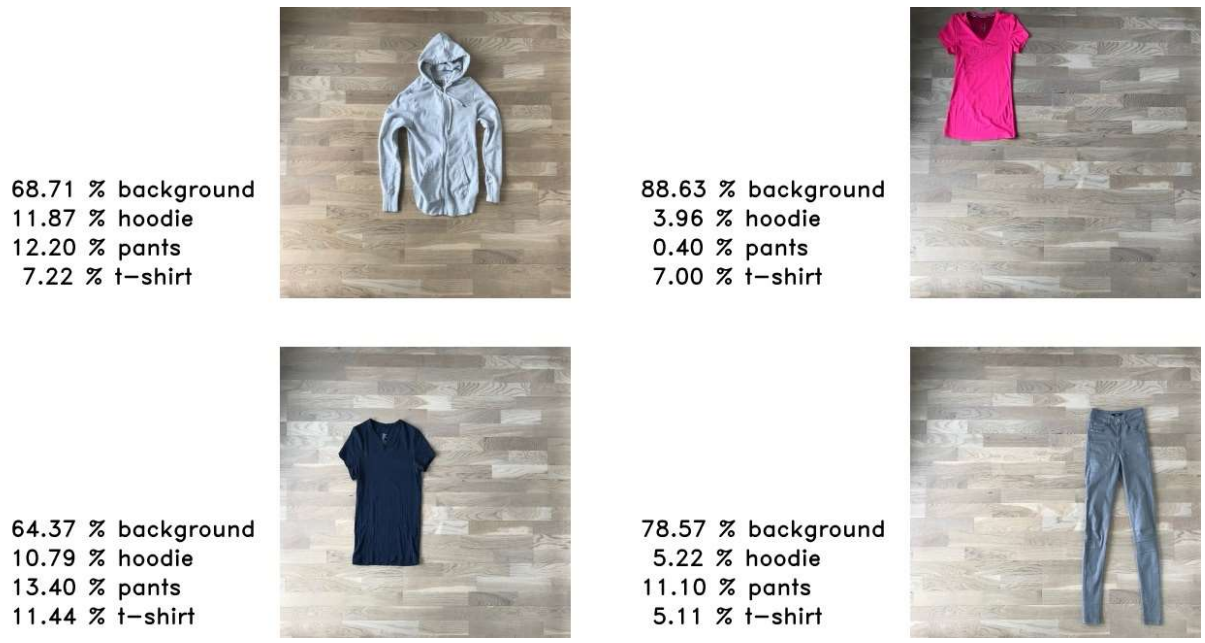


Figure 3.15: Examples of testing results produced by the total average label predictions on a large partitioned image using the sliding window algorithm.

Workspace limitations and final conclusion

While considering different alternatives to some of the challenges that had appeared during the development of this approach, it was eventually concluded that the manipulator to be used in any case was too small for the sliding window algorithm to be really applicable. Though the approach would be quite relevant in a more general use case and real environments, the considerably strict workspace limitiation prompted the consideration of a different direction. Further work on the CNN and the prediction weighting function was laid aside, and the sliding window method was ultimately concluded to be unappropriate for the new application in question.

3.3 The Iterative Single Item approach

The alternative proposed approach firstly focuses on constructing a rigid platform on which both the imaging sensor and the avaliable manipulator would have fixed - and thus assumed known - initial positions and orientations in relation to each other's coordinate systems. Next, an open-loop algorithm is developed with the goal to make the system capable of classifying one single clothing item laying on the platform, by iteratively unfolding or pulling on the garment between each classification attempt.

3.3.1 Algorithm overview and system architecture

Significant amounts of development time went into writing, reviewing and restructuring the software project architecture. In order to make the code simple to maintain, debug and experiment with, the entire foundation as well as relations between the different classes and modules were completely overhauled twice during the work period. The functions producing training vectors and retraining the CNN were especially made increasingly easier to run separately, which eventually saved time spent on waiting for the classifier to initialize itself for each invididual run. The resulting class hierarchy is shown in Figure 3.16.

Environment paths are imported to every custom module from the file **paths**, and constants or runtime settings are imported from the file **settings**. This setup was decided upon to avoid using globals in the code structure, and to gather all of the settings - which under development were continously altered and tweaked - into one file. The CNN module from the authour's earlier specialization project was also overhauled to fit this new hierarchy, but its core functionality was kept separated and largely untouched in order to reference it as an independent work.

The parser reads the downloaded training data as well as testing images produced by the OpenKinect (freenect) and image modules, and the classifier translates these images into vectors to be trained on and predicted by the CNN. Lastly, the classifier runs the main loop **classify clothing** which iteratively employs the planner module to communicate how the manipulator should move, based on how the sensors perceives the current environment.

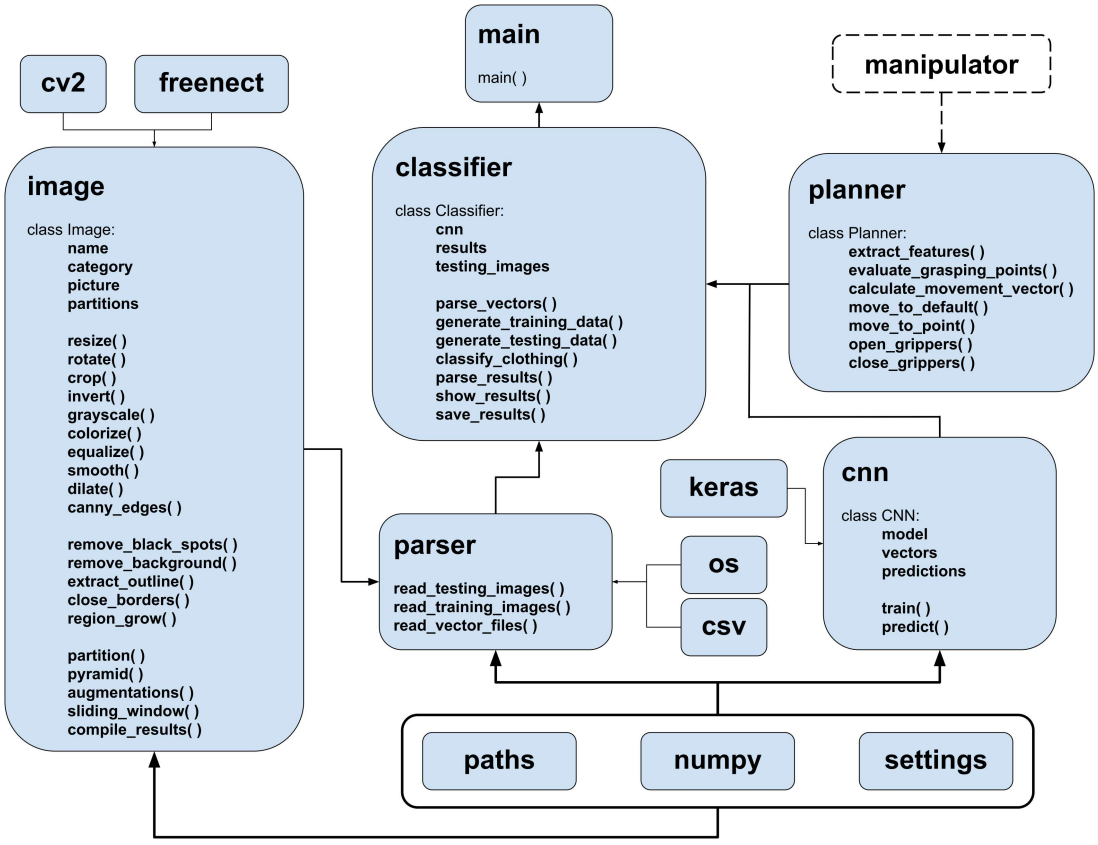


Figure 3.16: Classifier class diagram including variables and methods

3.3.2 Planning the setup

The camera and depth sensor

The ASUS Xtion Pro Live camera was initially considered for this thesis work, but reading any input from the device proved to be unsuccessful. Neither OpenCV nor any other regular video recording software was able to capture video from it, and no driver installations - like installing Artec Studio 12 [50], supposed to include PrimeSense/ASUS-drivers - was able to fix the problem. Windows detected the device when plugged in, but were not able to read anything from it.

The immediate available alternative was the Kinect v1 camera for the Xbox 360 console. In order to be able to use it on a computer, a power and USB adapter as well as a transformer had to be ordered from the UK. Installing the Kinect for Windows Software Development Kit v1.8 [17] and Developers Tool Kit [18] enables the user to open Kinect Studio and assert that everything is working correctly.

The manipulator workspace

As noted in the previous section, regular clothing items was realized to be too big for the manipulator to handle, and as such the proposed alternative was to instead classify baby clothes as a substitute considering the fact that no other manipulator was avaliable. Three different clothing items from each classification category was thus purchased from a used clothing store, and are shown in Figure 3.17.

Figure 3.17: These images taken by the Kinect camera show the three different clothing articles aquired for further testing in the new environment.

Additionally, the workspace of the manipulator was measured thoroughly to estimate the size and proportions of the platform to be built for the new setup. Figure 3.18 shows how the diagonal range of the end effector was measured to calculate the maximum size of the imaging area to be explored by the Kinect camera. The maximum extended range from base to end effector was measured to be approximately 42 cm. The manipulator base is positioned at the midpoint of the right border of the imaging area, which itself will be oriented as a 'portrait mode' 3:4 ratio image into which the clothing items will fit. The resulting area was estimated to be approximately 34.5 by 46 centimeters as the height and width, respectively. This area size will fit most of the baby clothing items to be pictured in the frame, and was deemed sufficiently large enough for the application. As one also may notice in Figure 3.18, a portion of the circular base is observed in the imaging area as a result of the attempt at maximizing the workspace area of the manipulator. This issue is addressed by designing the platform to hide the robot base below a wooden plate.

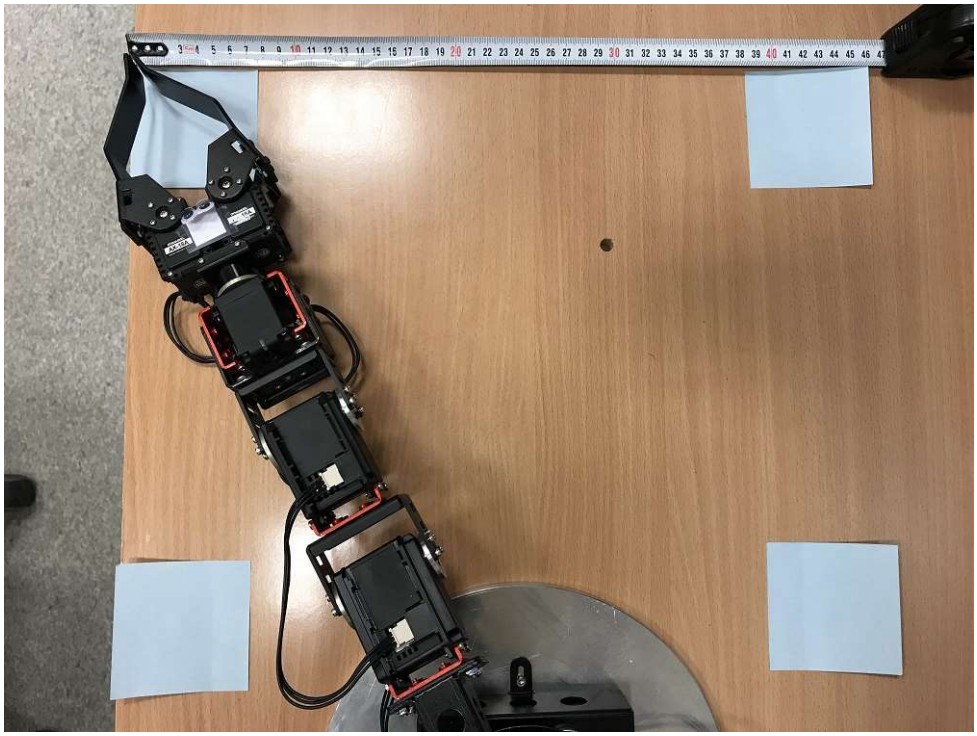


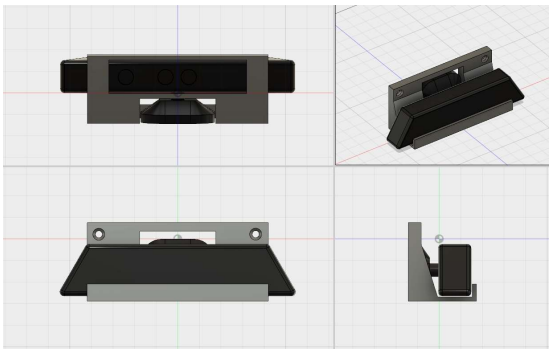
Figure 3.18: The diagonal of the robotic manipulator was measured to estimate its end effector range in the far corners of the workspace. Post-it notes show the approximate corners and borders of the imaging area.

By carefully constructing the imaging platform to achieve fixed and rigid positions for both the camera and the base of the manipulator, the position of the end effector of the manipulator will be considered known at all times through inverse kinematics and its known link lengths and joint angles. This decision was made collectively with the author of the manipulator module, in order to align the interfaces of the modules and enable straightforward communication between them.

The camera mount

In order to securely fasten the Kinect camera to a wooden frame in a rigid position, it was decided to use a 3D-printed mount. It was suggested that the 3D model for this mount should be modeled in Autodesk Solidworks, and could be printed at the workshop of the local community for Cybernetics students. However, as this software has been somewhat outdated, the official site redirects users to its successor, Autodesk Fusion 360 [51]. A model of the Kinect camera was produced and a support mount fitting this model was modeled around it, as can be seen in Figure 3.19. The resulting 3D model was measured to be 22 cm wide, and had a height of 9 cm. Two holes strategically positioned above the back of the Kinect camera serve as screw sockets. This design was refined to make the model easily fastened to its wooden base, while simultaneously ensuring that the camera is positioned in the same location and orientation every time. The base of the camera will be 'standing' 90 degrees on a wooden plate, pointing downwards onto the imaging area while the front edge of the mount holds the top of the camera in place. The horisontal position of the camera is in a similar manner held fixed by the tight fit around its base foot.

(a) The rough model based on a Kinect v1 camera

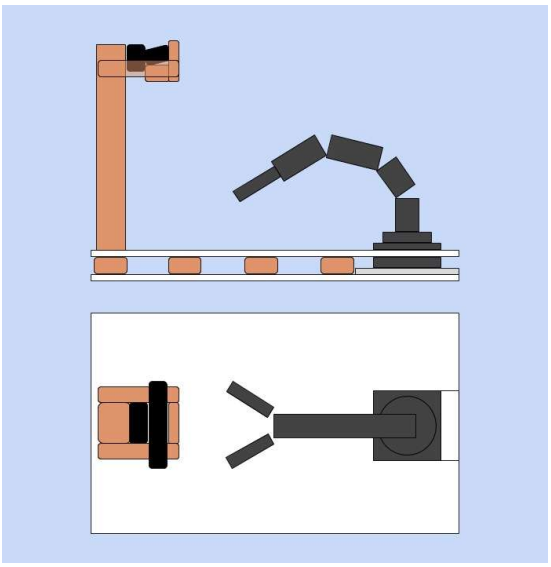


(b) The 3D mount to be fitted around the Kinect model

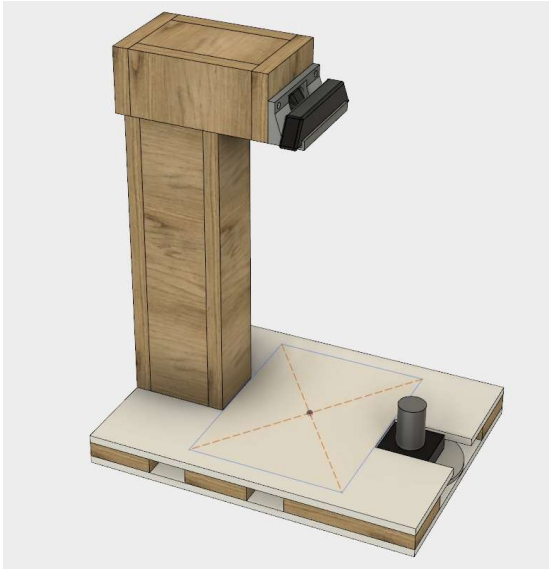
Figure 3.19: The camera mount 3D model

The platform design

The process of building the physical platform connecting it all together began with another 3D model in Autodesk Fusion 360 [51] based on a simple digitally painted rough sketch, both shown in Figure 3.20.



(a) A rough sketch showcasing the initial idea



(b) The first 3D model of the original sketch

Figure 3.20: The first designs of the platform build

The camera was initially intended to be positioned slightly off the midpoint of the image area, but the decision was made to move it as close to the middle as possible in order to avoid having the camera producing tilted and hence distorted views of the workspace area. The imaging area can be seen drafted on top of the white base plate in the 3D model, and the actual camera and depth sensor lenses are positioned as close as possible to the center. A simple slot in the top plate snugly cut around the surface of the base of the robot attempts to make the robot easibly removable while at the same time remain positioned in the same location during runtime.

The choices of woodwork and plates were made in order to reduce the cost and the complexity of the build. Planks were bought at a nearby hardware store, and the plates are actually repurposed IKEA kitchen cabinet shelves, 80 cm long and 60 cm wide. These proportions fit the intended design perfectly, and the white color matches the background of the available training data. Using the same size of wood planks for the entire platform would decrease the need for buying more of the same type if something went wrong. However, the quite large

top part of the structure resulting in the choice of plank sizes was concluded to be somewhat over the top, and a new iteration of the design was developed and rendered as seen in Figure 3.21. The new choice of mount support would hold the reduced weight with less woodwork used and a slightly better look.

(a) A rendered version of the platform build

(b) The last design iteration of the platform

Figure 3.21: Another iteration of the platform build design

3.3.3 Building the platform

The base plate

The bottom of the platform needed to be solid enough to handle both the manipulator and a sizable camera mount support column, and it was thus decided to fill the entire center of the base with solid planks except where the manipulator slot would be cut out. The IKEA plates were screwed to the planks from both directions to secure a rigid construction.

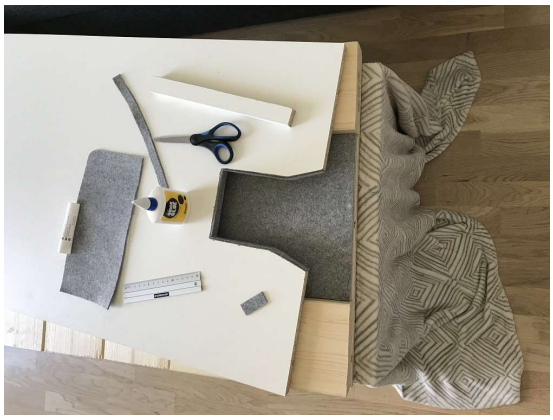
The fitted manipulator slot

The manipulator was fitted in a slot in the base plate, ensuring that the robot arm is positioned in the same location at all times during runtime. Additionally, soft fabrics were glued to every adjacent surface, in order to avoid damaging the surface of the manipulator base.

(a)

(b)

Figure 3.22: Images showing the base plate during the construction phase.



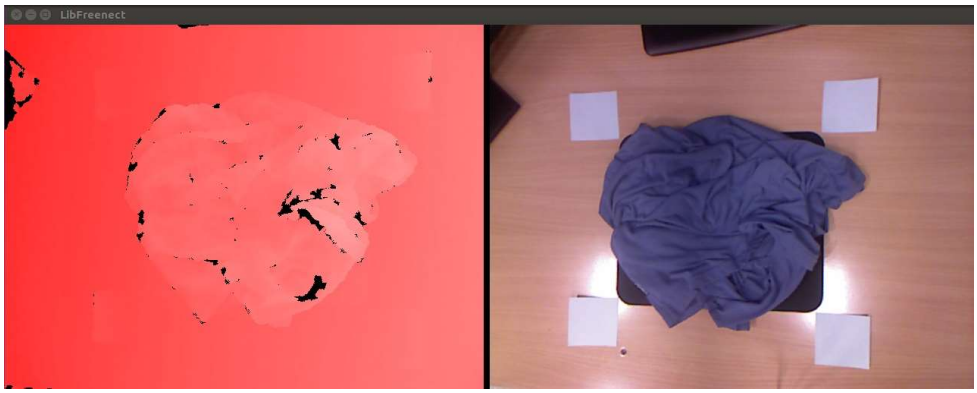
(a)

(b)

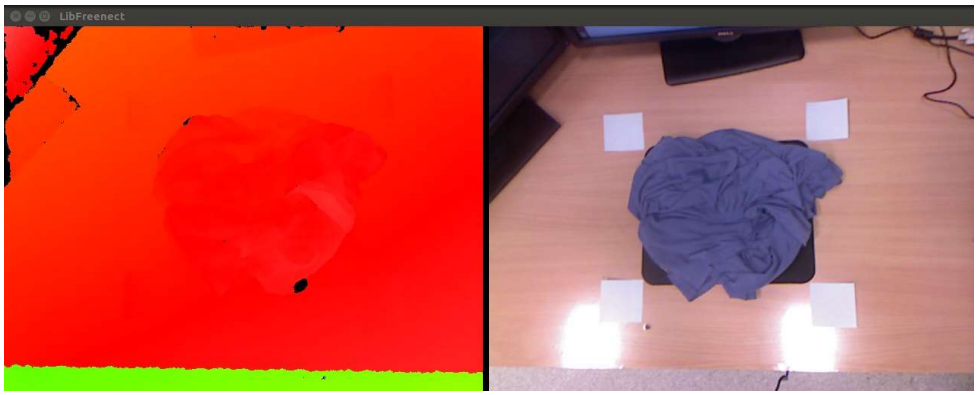
Figure 3.23: Images showcasing the manipulator slot during and after construction.

The camera height decision

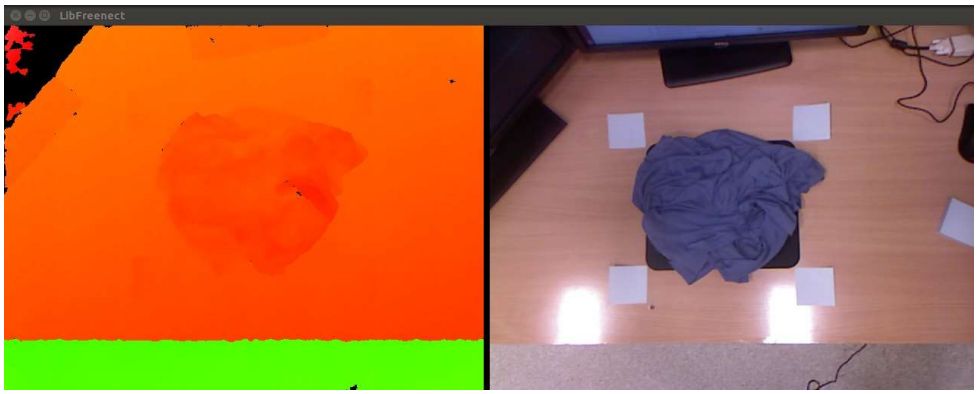
According to the documentation of the depth sensor of the Kinect camera [17], at least 50 centimeters of space is required between the sensor and the surfaces to be detected. By simply holding the camera by hand above the table with for three different approximately measured heights, it was determined that a height of 70 cm was a sufficient compromise between the quality of the resulting depth maps and the height and size of the physical platform.



(a) 60 centimeters above the base plate



(b) 70 centimeters above the base plate

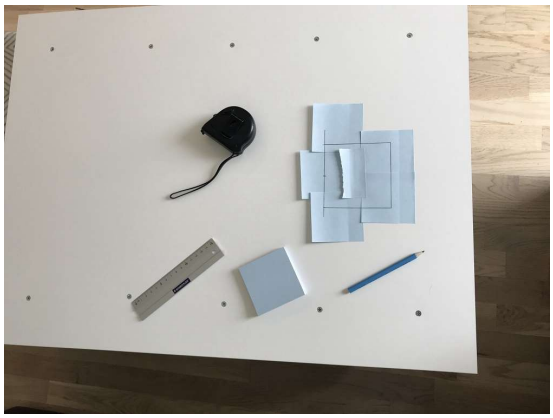


(c) 80 centimeters above the base plate

Figure 3.24: The Kinect camera placed different heights above the base plate.

The camera mount support structure

Measuring and cutting the camera mount support was more challenging than the rest of the build. The diagonal side supports had to be cut precisely in order to be perfectly aligned with the main column surface, and was fastened to the base plate from the bottom with extraordinary massive screws. The position of the support was carefully measured and outlined on the base plate, resulting in a correctly sized imaging area for the clothing within the manipulator workspace.



(a)

(b)

Figure 3.25: These images show the camera mount support structure outline on the base platform as well as the final result including a clothing item.

Fastening the camera to the support

Though originally intended to be attached by a 3D printed specifically for this purpose, the camera was ultimately fastened to the platform base frame support by using standard baggage straps. The resulting setup was suprisingly rigid, and was deemed sufficient for the application in question.

Figure 3.26: The Kinect camera was rigorously fastened to the camera mount support structure with baggage straps.

3.3.4 Reading the imaging sensors

A surprisingly big challenge was to be able to read the Kinect image data through Python in Windows. Through a significant period of time, several different tutorials were followed and the necessary dependencies depicted in these tutorials were attempted installed. However, ultimately building the OpenKinect library from source was not successful. The decision was made to instead use the Linux / Ubuntu platform moving forward.

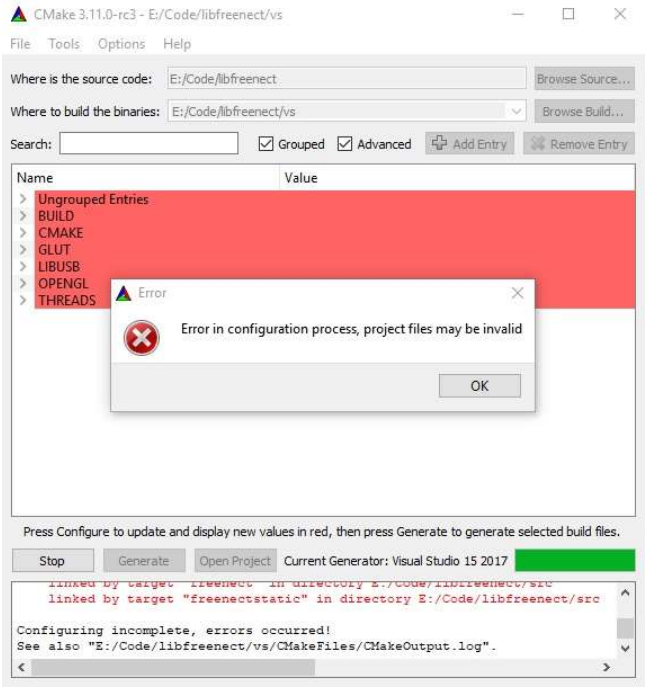


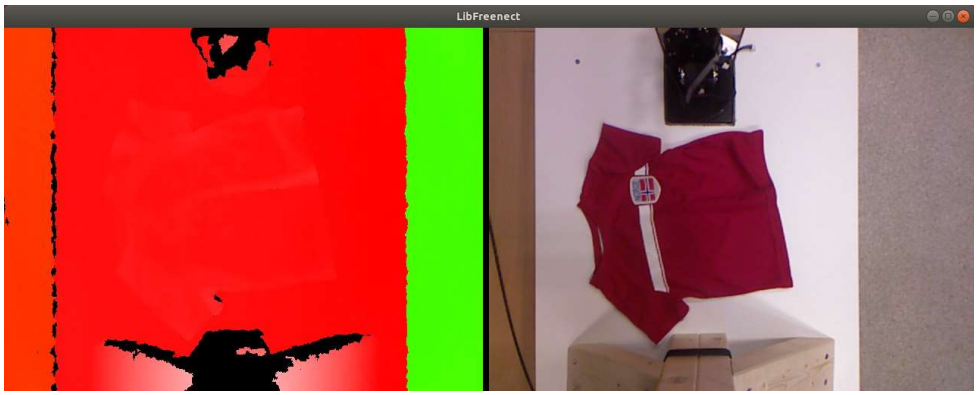
Figure 3.27: The last CMake error message seen before the attempt to read Kinect data using Python in Windows was abandoned.

In order to ensure that the installation steps given in the attached instructions Chapter C Installation instructions could be followed in order to allow other to utilize the work and results developed in this thesis, both the operating system and the Python environment was reinstalled several times. Ultimately, the OpenKinect (libfreenect) package was finally accessible and working using **Ubuntu 18.04**. A small module reading the image and depth sensor data from the Kinect camera was implemented, parsing images and depth data taken from the top of the platform into the feature extraction module.

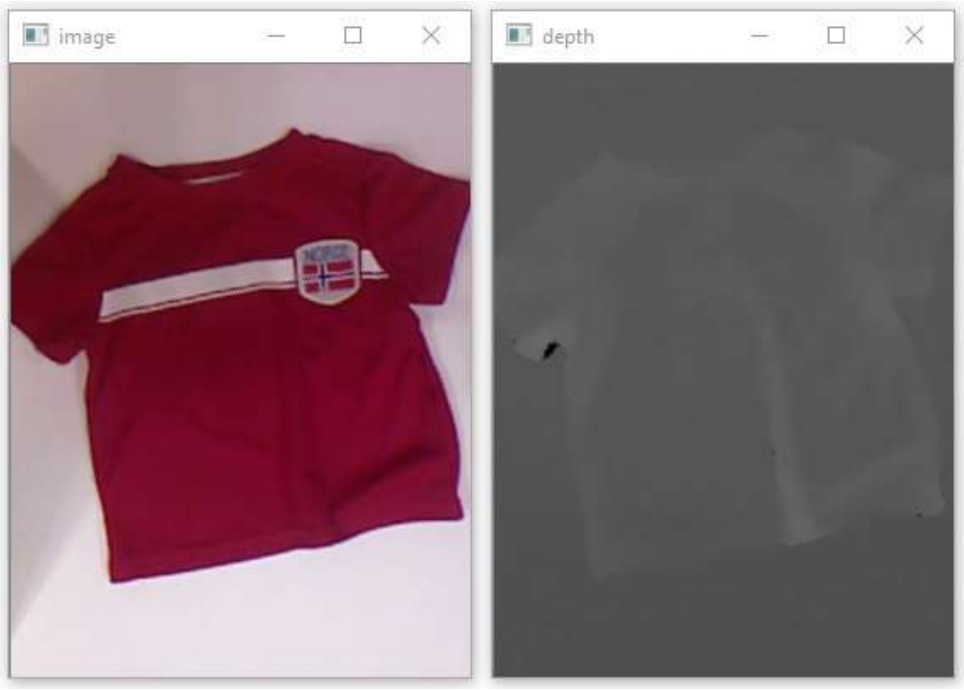
Targeting the workspace area

Both the color and the depth data images are firstly rotated 90 degrees since the camera sees down on the detection surface 'side-ways'. Next, the correctly sized classification area needed to be extracted from the complete pictures. However, as the camera sensors naturally are not positioned in the exact same location on the camera, there are is a slight discrepancy between the coordinates for the cropped windows in each image. These coordinates were empirically found by repeatedly trying to match the resulting cropped images by hand.

Additionally it was quickly noted that the "zoom" of the sensors were different, which meant having to resize the depth image to 91% of its original size in order to make the clothing match in both size and positions for both inputs. Lastly, the depth image was converted to grayscale to produce a cleaner and more homogeneous image ready for further image processing. The resulting output images are presented in Figure 3.28, and an illustration demonstrating how the cropping coordinates were estimated is shown in Figure 3.29.



(a)



(b)

Figure 3.28: Example input images of a red t-shirt in its natural state are perceived by the Kinect camera and processed to produce suitable testing data.

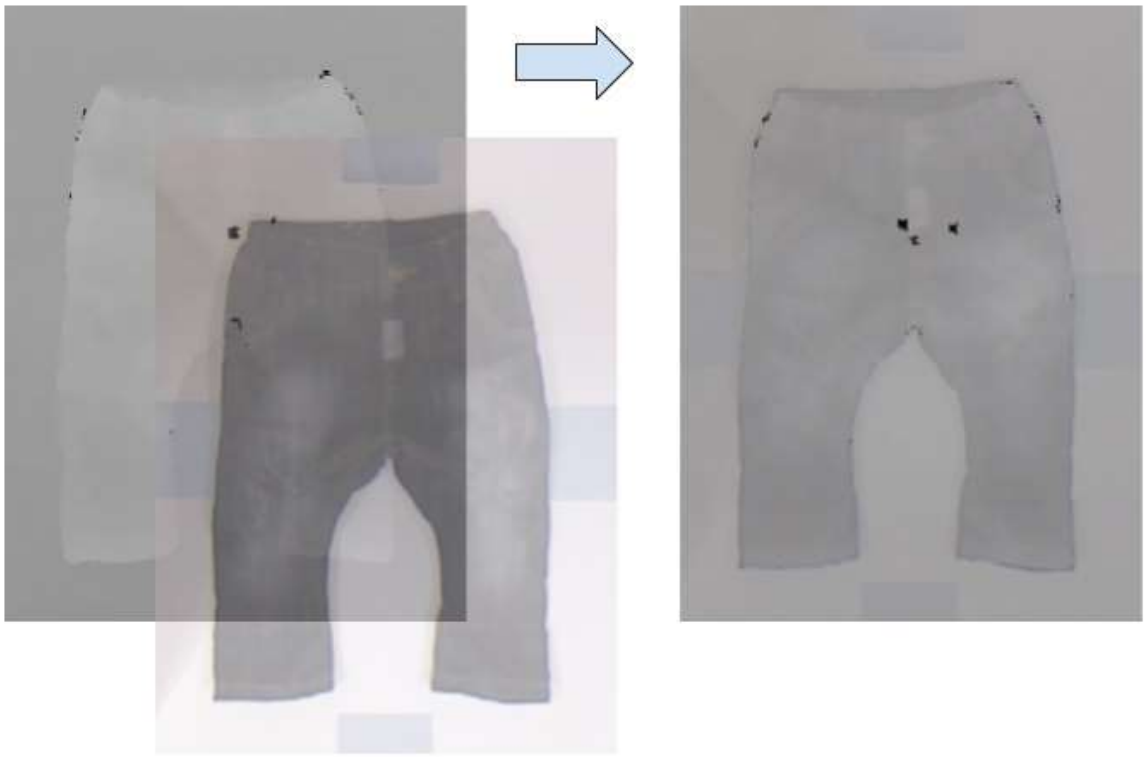
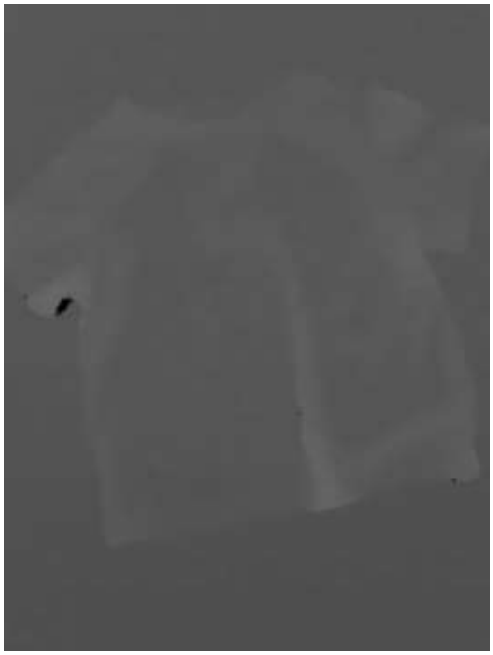


Figure 3.29: The process of matching the cropped positions of the color image and depth image was a tedious task.

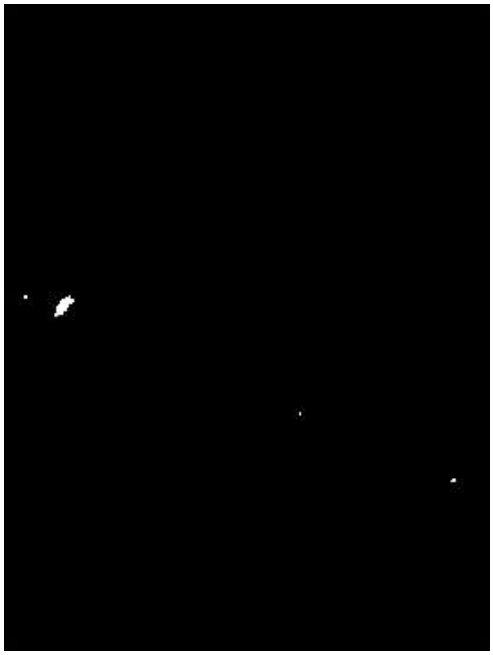
3.3.5 Feature extraction

Removing image noise

The cropped and grayscale depth images contained some black spots. These are assumed to be related to the issue that the emitted speckle pattern of the Kinect emitter does not always get reflected back to the infrared camera sensor, usually caused by large depth differences between objects in its view or highly reflective surfaces. Removal of these black spots was attempted by the use of a self-implemented image inpainting method [52]. The original image is scanned for black pixels with intensity values below a certain threshold. The identified spots are then set as foreground (white), and the rest of the image pixels are made black to create a binary mask targeting the spots to be removed. See Figure 3.30.



(a) The original depth image containing black spots of image noise.



(b) The binary image mask used for image inpainting.

Figure 3.30: Image noise example and its associated inpainting mask

Next, the produced mask is used along with the original depth image in an attempt to remove the spots in the image. This is achieved by simply calculating the average intensity value of all neighboring pixels around the white spots in the mask, and then setting the corresponding pixel values inside the mask as the calculated average value. The resulting noise-reduced image along with its original input is presented in Figure 3.31.

(a) (b)

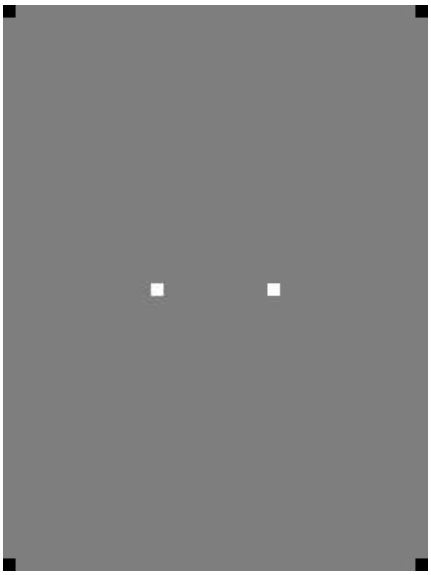
Figure 3.31: Removing image noise from a depth image by inpainting

The GrabCut and the Watershed algorithms

Two popular foreground extraction methods were studied and experimented with, but the attempts ultimately proved unsuccessful. The first approach was the GrabCut algorithm [53], an interactive foreground extraction method designed for fast and easy supervised extraction. The application is operated by manually drawing a rectangle on the image from which one wish to extract a specific foreground, which yields an example cut for the user. If the cutout is not precise enough, one may further draw lines on particular parts on the image to specifically mark segments as foreground or background. One may also apply a predefined marking mask, providing the algorithm a more accurate way to determine sure backgrounds or foregrounds. However, as the objective of this work is to produce a fully autonomous system and the algorithm did not produce satisfactory foreground extractions without any manual input, this method was deemed unappropriate for this application. See Figure 3.32 for an unsuccessful example attempt along with the applied extraction mask.

The next method for consideration was the Watershed algorithm [54], which treats the intensity values of grayscale images as 'peaks' and 'basins' of high and low valleys and mountains. By regarding the image as a topographical map rather than a flat surface, one may then 'fill' the basins between the differing intensity heights in an image to segment perceived bod-

(a) The resulting image after GrabCut has been applied.



(b) The attempted mask for marking sure foregrounds and backgrounds.

Figure 3.32: An unsuccesful GrabCut algorithm attempt

ies from each other. The algorithm works by first converting the image to a binary image, and eroding the white edges to mark the sure foreground. Similarly, the sure background is marked by dilating the edges. Next, a mask is created using a distance transform and thresholding, which is finally applied to the image in order to fill its 'valleys' inside the masked foreground. However, this algorithm was also considered to be somewhat excessive and inadequate for this type of application, as the segmentation part in this work will only extract the foreground for a single entity every time. Figure 3.33 shows an example attempt at extracting the foreground for an input sample.

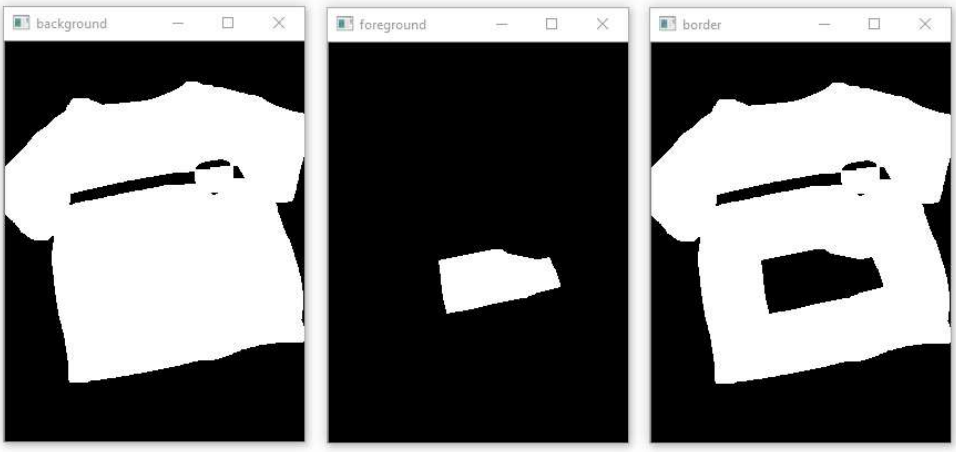


Figure 3.33: An attempt at Watershed foreground extraction

Outline detection

The focus was instead aimed at firstly extracting the contours of input images, by using the famous Canny edge algorithm [21] for edge detection in an image. Its parameters were set as they were experimentally determined during the work on the CNN module from the author's specialization project, and the algorithm thus immediately produced satisfactory edge maps for every clothing item available for examination. The resulting edge maps would further serve as the base for the following foreground and extraction methods presented next.

(a) (b)

Figure 3.34: The Canny edge map for outline detection

Foreground and outline extraction by region growing

Next, a custom border closing algorithm was implemented in an attempt to close the outline borders of an object which extends past the image frame borders. Every actual edge or corner pixel along the image borders are firstly set to background (black), and the function subsequently runs along the neighboring image frame rectangle from the image borders to check for white edges. If a white pixel is found at any point, every black pixel between it and the next detected white pixel is set to white - effectively closing the border of an assumed foreground edge.

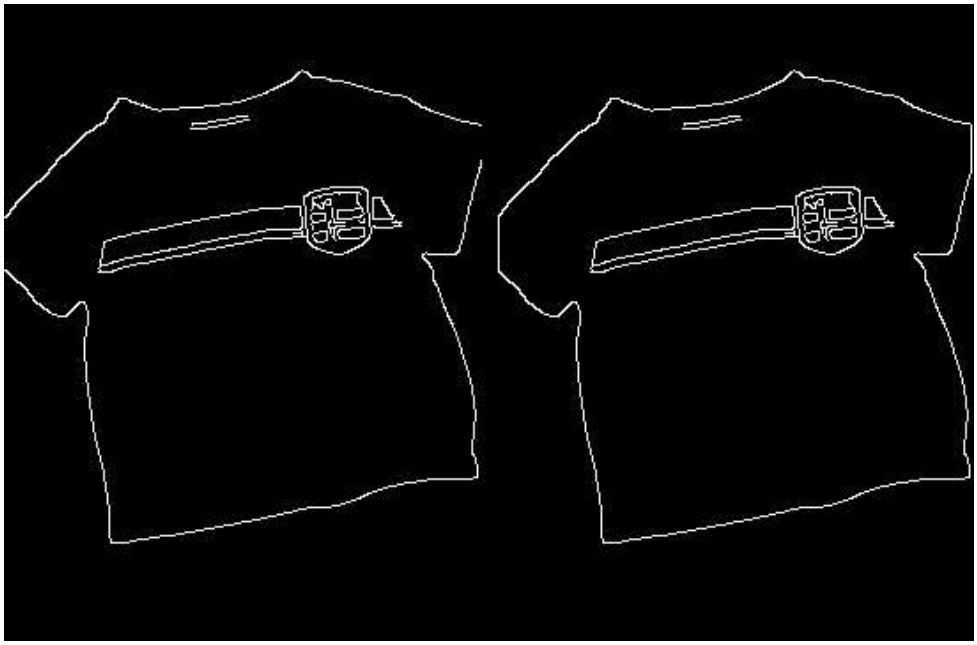


Figure 3.35: The borders of an outlined object in a Canny edge map are attempted closed by a self-implemented iterative pixel intensity function along the image borders.

Following the border closing procedure, the foreground is attempted filled by implementing a simplified inverted region [55] growing algorithm. The region grow segmentation technique works by choosing a seed foreground point in a binary image that "grows" outwards from the seed, until every background pixel within a closed edge outline is set to white. However, the challenge within this approach lies in the fact that the seed point is often unknown or difficult to determine, and is a familiar problem in the field of computer vision and segmentation.

This generally difficult problem is in this work solved by essentially setting every image border pixel as a background seed point. By instead working inwards from the image edges and corners, the algorithm would fill the entire background around a detected object outline, assuming no background holes are present inside the object contours. Hence, the inverted region growing method determines which pixels belong to the background segment, and setting the remaining undetermined pixels as foreground pixels is left a simple task. Figure 3.36 shows how the algorithm iterates inward from the image borders setting every neighboring non-white pixel gray. After every unexplored black pixel is set as background, the algorithm then simply sets every remaining black pixel to foreground.

Figure 3.36: The region grow algorithm during runtime. The gray color illustrates the determined background pixels, black pixels are yet to be determined, and white represents the resulting foreground pixels.

Combining the foreground and depth maps

The enhanced depth map discussed in the previous sections is subsequently superimposed onto the extracted foreground image, after another few slight alterations. The depth map is firstly equalized using the Histogram Equalization method *cv2.equalizeHist* [23], in order to spread the intensity differences across the entire grayscale spectrum. Next, a percentage of the average pixel intensity value of the depth map is subtracted from every pixel value, essentially removing the miniscule and uninteresting brightness differences between the intensity values. The resulting map is then added onto the foreground image, and the non-zero intensity values outside the clothing outline are discarded.

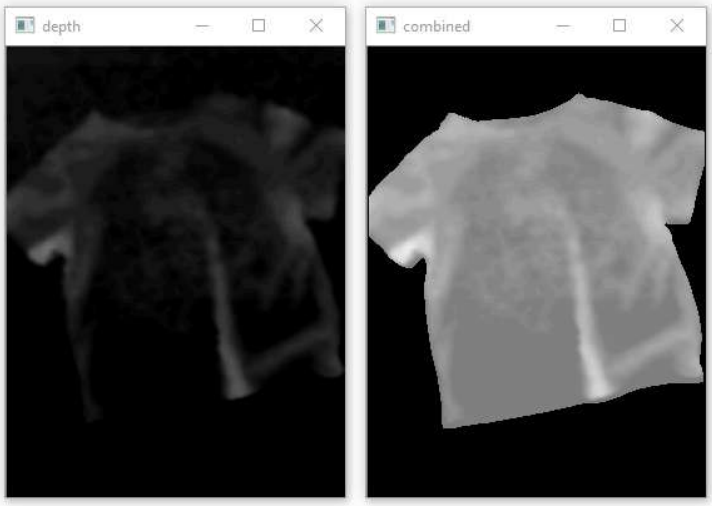


Figure 3.37: Superimposing depth data onto the foreground

In addition to the depth perspective percieved in the foreground of the clothing item, it is also useful to extract the outline of the foreground itself. Simply marking every white foreground coordinate with neighboring black background pixels as an edge produces the desired contour of the clothing, onto which one may apply dilation [22] to increase its size and visibility. In order to further increase the visibility of the extracted features, the foreground depth is colored yellow and the green outline trace is added on top of it. See Figures 3.38 and 3.39 for resulting example images.

Figure 3.38: Outline extraction. The dilated outline of a Canny edge map is shown to the right in green.

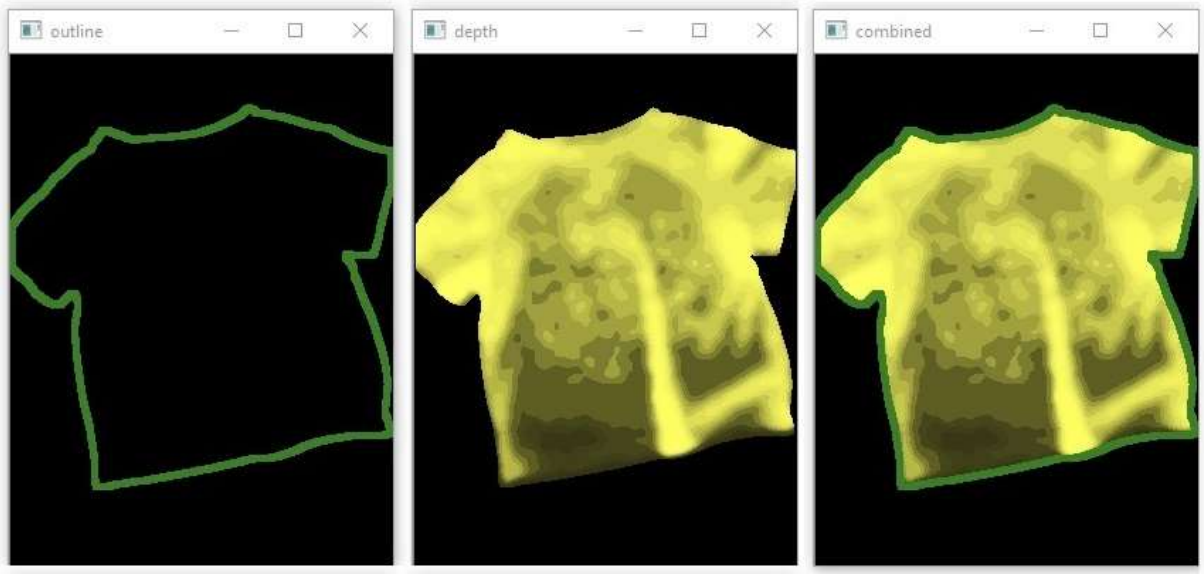


Figure 3.39: Combining the outline and depth images

Lastly, it was eventually clear that the depth map was not as informative in its current state, and another segmentation method was applied by simple thresholding, similar to the Watershed algorithm mentioned above. If the intensity values of the clothing were below a 60% of the total intensity difference of the foreground, it was colored yellow. If the brightness was above 80% the pixels was colored red, and the intermediate range in between was colored orange.

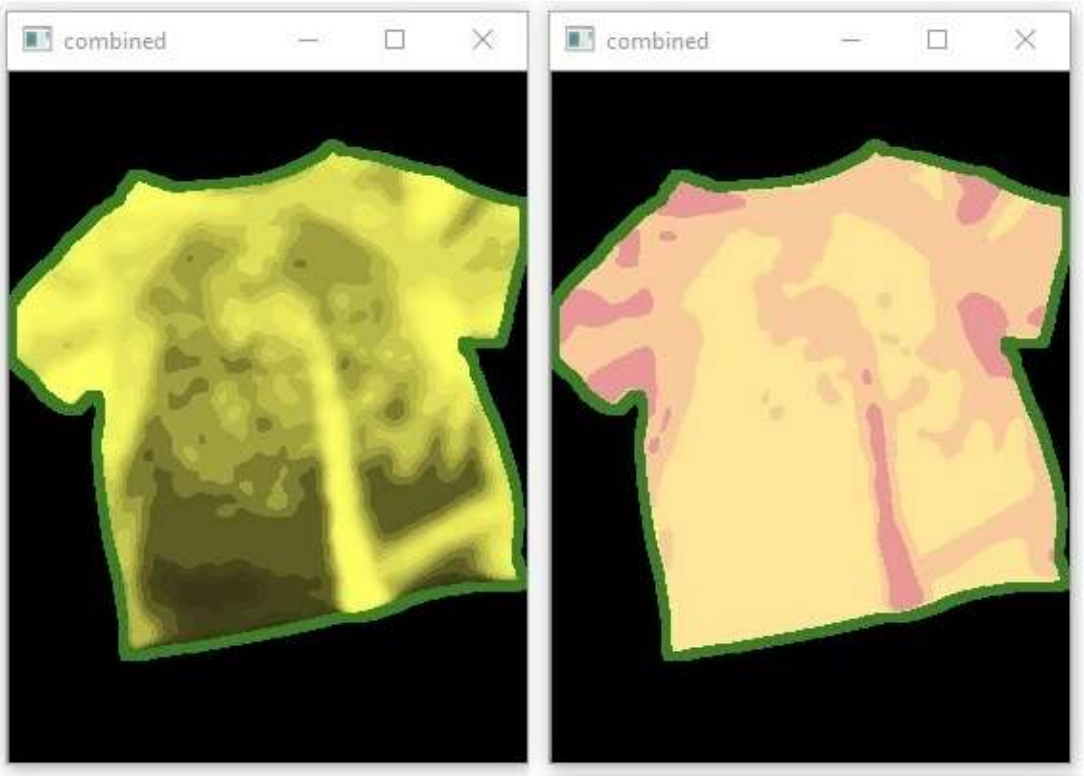


Figure 3.40: A depth image segmentation clustering similar pixel intensity values together by use of thresholding.

Corner detection

The final feature extraction method to be applied to the input images provided by the Kinect camera is corner detection. In addition to the outline around a foreground of the clothing item, one may also find the sharp directional changes in the outline interesting for grasping purposes. The corners are simply extracted by applying the well-known Harris corner detection algorithm [56], which efficiently produces a collection of corner coordinates after some minor parameter tweaking. These corners are finally marked as blue spheres onto the feature image, ultimately concluding the feature extraction scheme explored in this thesis. See Figure 3.41 for the resulting feature detection output.

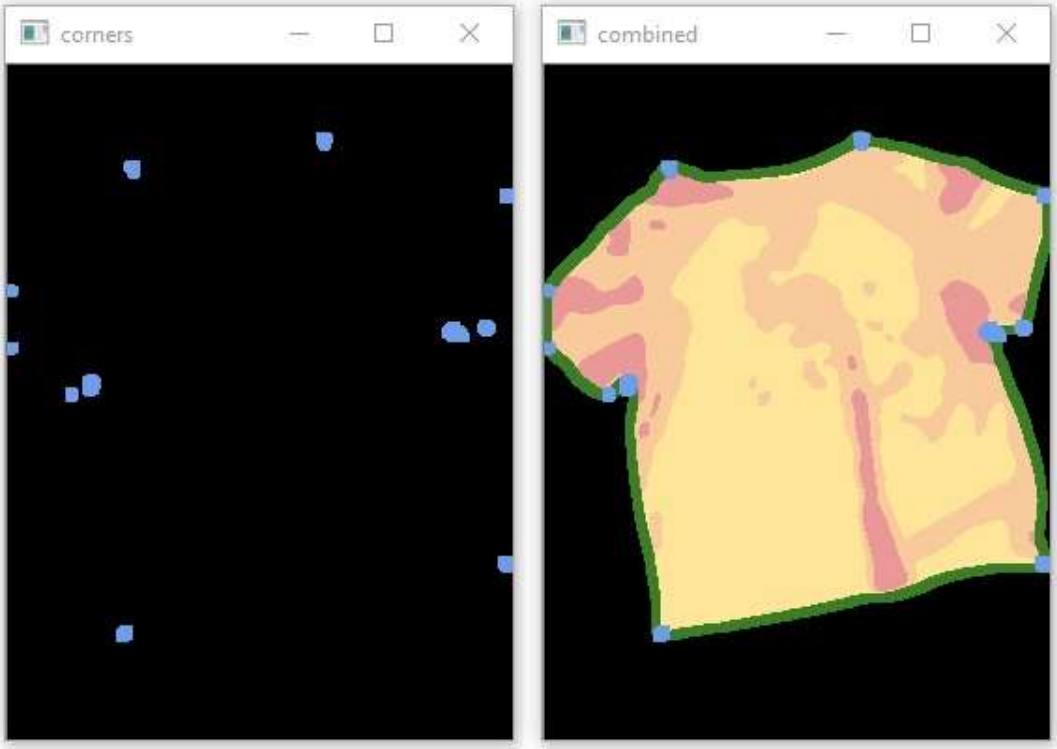


Figure 3.41: The detected corners of a combined depth and color image are marked as blue spheres at their respective coordinates in the image.

3.3.6 Planning and control

After sucessful feature extraction, the planner decides upon which of the available features is to be grasped, and controls the movement of the manipulator accordingly.

The grasp point ranking algorithm

Firstly, the **evaluate grasping points** algorithm is run to rank the feasible grasping points if available:

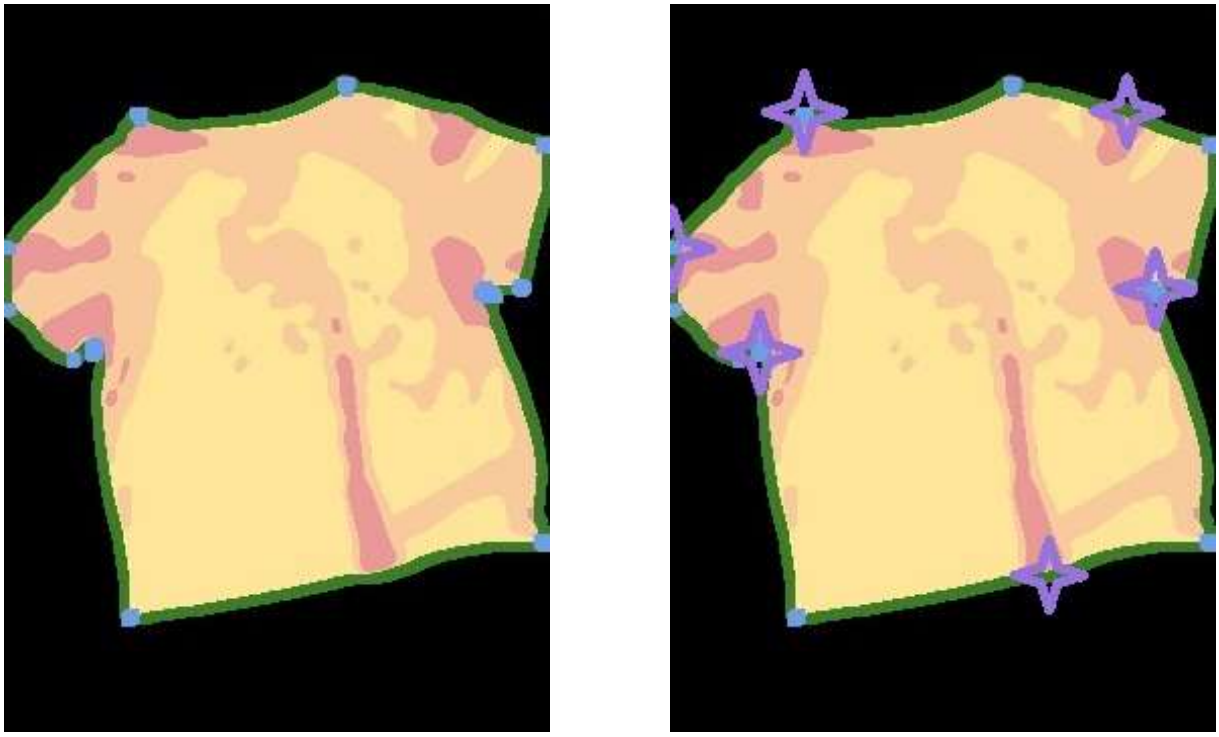
1. **If corners are detected, number the corners based on the neighboring red pixel score within a radius.**
   More red pixels is better as it can be somewhat assumed that the red height differences show folds or creases that needs to be straightened out.

2. **If no corners but outline exists, estimate the best grasping point on the outline based on number of neighboring red pixels like above.**

3. **If no outline exists, use depth map and look for lowest points with the *least* amount of red in neighborhood *within a radius of red or orange peaks*.**
   This scenario is by far the most difficult to handle. The thought is that this particular approach would hopefully attempt to grasp a part of the clothing which is not folded, and will produce the least amount of distortion to the rest of the clothing if pulled. Grabbing a garment in a protruding area may very well further distort the clothing to an even more unrecognizable shape if the grasp point is not near the edge of the fabric. Yet, it will also possibly be very difficult to determine which perceived flat part of the image is actual clothing and not the table, if no outline exists. This issue is regarded as a significant potential pitfall of the approach.

4. **If no corners, outlines, or depth map differences are discernible, the process terminates with indefinite results.**

Figure 3.42 shows an illustration depicting highly ranked grasping points. After ranking the points, the planner calculates an appropriate movement vector for the highest ranked grasping point and instructs the manipulator to perform the grasp.



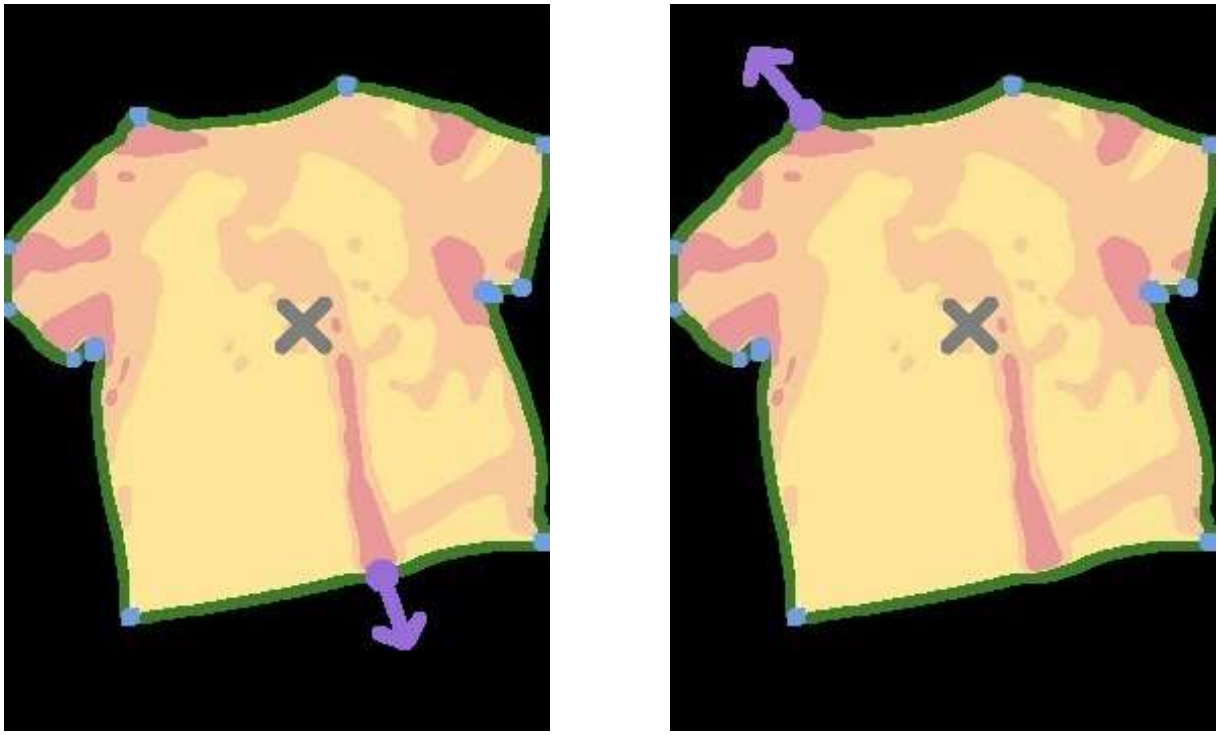
(a) (b)

Figure 3.42: An example illustration of the resulting ranked grasping points, shown in the right picture as violet stars. The corners (in blue) are ranked higher than grasping points located on the outline (green).

Calculating movement vectors

The movement path coordinate algorithm will choose the most suitable grasping point if available, or at random if the estimated ranks are the same. Then the center point of the clothing will be calculated, based on the collective positions of every pixel inside the outline - if the outline exists. If the outline is not available, the point where the depth map is the most dense, i. e. large height difference values, is chosen.

Next, a linear 2D vector from the middle point to the grasping point is calculated, which points in the direction away from the center point. The distance to move away along the movement vector is estimated based on a measure of the total count of red and/or orange pixels in its neighborhood, as well as its relative point to the center points of both the clothing and the image frame. Large height differences may indicate more unfolding potential, and the center points are included in the calculations as an attempt to keep the clothing as close to the center of the image frame as possible. The resulting end point of the vector is the release point for the manipulator, as illustrated in Figure 3.43.



(a) (b)

Figure 3.43: Two different grasping and movement vectors. The vectors are calculated by extending a line which runs from the center point of the clothing and through the grasp point.

Manipulator control in 3D world coordinates

The challenge of this aspect of the manipulator controller is to translate the previously exclusively two-dimensional coordinate points into three dimensions. Namely, the height above the base plate along the z-axis must be estimated. The proposed scheme uses the enhanced depth maps produced by the Kinect sensor in order to estimate an approximate height. By simply converting the perceived height differences in the clothing to rough distances in real world centimeters, one is able to estimate an approximate grasping point above the base plate. Precise and exact grasp points are not necessary as clothing typically is both soft and flexible in nature, making this approach adequate in most cases.

The manipulator would thus move to a grasping point with an estimated certain height above the base plate, close its grippers, move to the release point at the end of the calculated movement vector with the same height, open the grippers, and move back to default position outside of the imaging area. If the new perceived image is not changed from the last, the height above the base plate of the grasping point may be slightly reduced, and the grasp could be repeated for a second attempt. However, the performance and success rate of this procedure is unfortunately not possible to put to trial for evaluation and adjustments. An illustration of the resulting movement vector for the manipulator is shown in Figure 3.44.

Figure 3.44: Illustration of a manipulator grasping movement vector.

Chapter 4

Results

Short summaries recapping the main insights gained throughout this thesis work as well as references to the final planning and prediction results are presented in this chapter.

4.1 The complete system algorithm

The final novel approach proposed in this thesis may be summarized by the following steps:

1. Color and depth images are read into the system by a Kinect camera, oriented down towards a workspace surface.
2. The foreground of a clothing article located in the imaging area is segmented and extracted using region growing inside its enclosed contour borders.
3. Next, the outline of the clothing is extracted by the use of the Canny edge algorithm.
4. The depth map from the input sensor of the same imaging area is subsequently masked and superimposed onto the identified foreground area.
5. Corners in the previously extracted outline are then detected using Harris corner detection.
6. The detected corners are together with the original outline also superimposed onto the foreground, to produce a visual map of the useful features extracted by the image pair.
7. Next, the planner module firstly attempts to classify the clothing article in the original color image using the already implemented convolutional neural network module.

8. The resulting predictions are then used to guide the algorithm forward. If the predictions are confident enough based on certain criteria, the classification is assumed successful and consequently stopped. If however there are still significant uncertainties present in the prediction percentage distributions, the algorithm continues.

9. The manipulator then attempts to grasp and move the clothing by use of the proposed grasping points evaluation and movement vector calculation schemes, as an effort to increase the visibility of the natural features of the clothing article.

4.2 The sliding window method

4.2.1 Using classification as a detection scheme

Training the CNN to determine wether an image partition is just a background or if it is containing a clothing item proved itself to be an exceedingly challenging task. The inclusion of the *background* training data category attempted to differentiate "anything else" from the three other possible prediction classes. However, because the intent at the time was to further use the same algorithm to detect the position of the manipulator end effector 'online' - while visible in the image during runtime - the added training data was eventually suspected to be too variant for this use case. Though the approach was consequently abandoned as a result of the manipulator size issue, the method may still ultimately have produced satisfying results if pursued further, such as using white plain images as the 'background' data set more similar to the rest of the training data.

4.2.2 The weighted predictions method for large environments

It is speculated that more carefully balancing the differently sized sliding windows applied to the image pyramid could have significantly improved the classification results for this approach. All four of the different prediction attempts applied onto the large environment scenario were either showing tendencies favoring the correct classification labels, or could to a reasonable degree be defended to be erroneous for the most part by the fault of the superficially and quickly implemented prediction area weighting function. If not for the strict limitations imposed on the environment size during this work, these results would be deeper investigated and the method further advanced.

4.3 Physical setup and construction

The design and construction of the physical platform is considered to be one of the most successful products of this work. Though the manipulator eventually became unusable throughout the work period, the platform seems to accomodate its intended use case to a satisfactory degree. Additionally, the baggage straps substituting the 3D printed camera mount are regarded as sufficiently rigid for the application in question, as the imaging sensor is indeed securely fastened in place enabling suitable overhead pictures taken of the classification environment.

4.4 Performance

4.4.1 Feature extraction and detection

The successive combination and application of several various and well-known computer vision techniques for image processing, feature extraction and detection ultimately produced both visually and spatially informative representations in relation to the perceived environment considered in this thesis. The resulting generated feature characteristics images enables further analysis and calculations for subsequent physical manipulatons in space, and ultimately, classification.

4.4.2 Manipulator planning and control

Although considered one of the more time-consuming and challenging parts of the complete system presented in this work, the efforts and performance related to the planning scheme of the manipulator control module are difficult to thoroughly assess and evaluate due to the internal damage inflicted to the physical manipulator. As it was not possible to test the finished application as intended, the implemented planning module is thus left as a hypothetical element of this work. However, it is believed that an open-loop manipulator module controlling the movements of a robotic arm using only 3D world coordinates as its input may be able to perform the desired functions discussed in this thesis to a satisfactory degree. No mathematical calculations were performed with regards to robotics dynamics and inverse kinematics.

4.4.3 CNN parameters and settings

The following training and prediction parameters were set as experimentally attempted optimized in the author's specialization project.

1. 3 separate clothing categories (classes), namely 'hoodies', pants and t-shirts.
2. 500 data samples per category, along with 8 alterations of each producing a total of 13500 samples.
3. The validation ratio was set to 0.2, splitting the data into 80% training and 20% validation data.
4. The number of epochs to train was set to 10.
5. The data is shuffled between each iteration by default.
6. The training batch size was set to 64.
7. The prediction batch size was set to 32.

4.4.4 Classification output

The final label predictions generated by the convolutional neural network module were as expected, given the results produced by the specialization project from which it was implemented. However, there were also considerable issues attributed to the new and altered application use case considered in this thesis. The resulting classification outputs of the clothing images taken by the Kinect camera were generally overconfident and often erronous. Additionally, the orientation of the pictured garments effected the prediction results to a large degree. These issues are further discussed in Chapter 5.1.4 Classification challenges. A number of example outputs of prediction results as well as detections may be reviewed in Chapter A Feature extraction and detection results and Chapter B Classifier prediction results.

Chapter 5

Discussion

This section discusses a few of the challenges and problems encountered during development throughout this thesis.

5.1 Development challenges

5.1.1 The sliding window approach

Training the CNN with a 'background' class

At the time of development, this approach was intended to work with the manipulator still in the picture, which meant having significant amounts of noise present in the imaged environment during runtime. The intent was to be able to detect both the clothing items and later the actual manipulator end effector in the same image, and the classification module consequently needed to be sufficiently robust in order to handle this use case. Thus, a very diverse collection of backgrounds was used to train the CNN along with the other clothing classes. This decision likely had a considerable negative impact on the prediction results, but as it at around the same time also became apparent that the manipulator to be used for this work was not big enough for the sliding window anyway, the issue was not investigated further. If however the goal of detecting the manipulator end effector was to be set aside and the available manipulator would indeed have the size to manage larger environments, simpler background data samples could replace the current complex ones and possibly significantly improve the prediction results produced in this particular use case.

Furthermore, the briefly mentioned so-called "Photoshop bias" present in the internet samples may also have a severe effect on prediction results. Deep learning techniques are generally prone to data distribution bias, and this application is no exception. As every single training data samples have "perfect" white backgrounds as borders, one may assume that there would be a large disrepancy between the these samples and the actual backgrounds containing various patterns and textures. For the purpose of testing during the development process, only the use case with clothing lying on a wooden floor was attempted to be classified. Though the background training samples were attempted to have a high degree of variation, it is suspected that the CNN over-emphasized the importance of the white border backgrounds in every clothing item data sample. When these white and perfectly smooth backgrounds were not present in the testing data at all, it would not be unlikely that the classifier interpreted any texture other than flat values of 255 as "backgrounds". Furthermore, the variation in the background data samples may simply be too diverse, and is consequently attributing clothing patterns and features to that of random background noise. These concerns give rise to several ideas for future work, mentioned later in this chapter.

Detecting only parts of the clothing

The most prominent issue with the sliding window approach combined with a CNN was that the windows containing only a small part of the clothing could produce unreliable predictions. If the CNN is trained exclusively on pictures showing the entire clothing garment from top to bottom, attempting to predict the class label of only parts of these clothing items may produce completely random results. Though it could be tempting to assume the neural network would just produce evenly distributed probabilities for every image it was not certain of, this is not necessarily the case. Unstable predictions such as these may often lead to the bias of trying to infer relations in data that is utterly uncorrelated, and it is important to keep this in mind. Likewise, this issue would also cause problems for the attempted object detection algorithm. If for example a situation arises where merged probability distributions across an image indicate there are several separate clothing items present because of randomly distributed prediction, it would be hard for the algorithm to decide what the circumstances actually are in such a case.

The suggested solution to the above problem is to emphasize the differences in abstraction and environment area covered in the various layers of the image pyramid. Higher leveled layers cover larger areas of the total environment, and are more likely to actually contain the full bodies clothing articles in any given sliding window partition. Hence, these partitions should be weighted more when the total prediction distribution is calculated in order to locate garments. Furthermore, in addition to the fact that the "smaller" partitions - i. e. the subimages extracted from the lower levels of the image pyramid - cover less area of an image than the top levels, they are also significantly outnumbering the higher leveled partitions. Thus, these partitions should be weighted almost insignificantly compared to the ones containing more high level information. It is however still useful to utilize the knowledge gained from the smaller partitions, namely the case that if there is indeed nothing in the frame, the CNN would recognize the area as a background and subsequently produce meaningful evalutations related to exactly where in the environment clothing items may be detected or not.

5.1.2 Imaging platform issues

The uncertainty of the camera straps

Considerable efforts went into the planning and modeling of the to be 3D-printed camera mount, but it was regrettably laid aside after it was discovered that the available 3D printer actually was too small for the needed size of the mount. Another 3D printer and workshop was also considered, but required the user to have specific permits in order to be allowed to use it. As the thesis work period at that point was nearing its end, a quick substitute solution was applied. Using simple baggage straps, the camera was able to be fitted to 'stand' sideways in exactly the same position and orientation as the 3D printed mount would have provided support for. This however makes the exact true position of the camera and depth sensor lenses uncertain to a larger degree, and may no longer be assumed to be rigid and stay the same every time. Having the exact relative distance between the imaging sensors and the manipulator known at all times is key in order to be able to operate precisely in the real world. Hence, uncertainty introduced here could have proven to be a significant problem if the intended manipulator was operable, and as such finalizing the 3D printed camera mount would in that case have been a top priority as originally intended.

Lighting on the platform

Providing the platform with lightning became an unexpectedly challenging task. A lamp was set up close to the platform, which would illuminate the imaged environment to a satisfying degree. However, the depth sensor of the Kinect camera seemed to struggle with such a strong light source close by. The black spots of the depth image indicating that the sensor reads no bounce-back signal increased in numbers, and decreased the quality of the feature extraction and detection algorithm results. It was concluded that the algorithm worked best without the light source, and one may also argue that it is an advantage to have the method perform in variable lighting conditions in terms of operation robustness.

Baby clothing size and surface friction

It was quickly discovered that the small sizes of the baby clothing used in this work caused problems when manipulating the garments with the robot arm. As the clothing articles were so light, the fabric had a tendency to slip on the smooth surface of the imaging platform. This issue is however presumed to become insignificant if the proposed approach would be applied to a larger environment befitting of adult-sized and heavier clothing, as well as on other more coursely textured surfaces.

5.1.3 Manipulator concerns

Grasping and sensory feedback

One of the main and greatest concerns throughout this work was the open-loop nature of the grasping algorithm. Implementing control schemes without the inclusion of any form of sensory feedback may in many cases be inaccurate, unreliable and/or even dangerous. It is inherently difficult for the manipulator to know it it has performed a successful grasp, and it would be useful to employ both visual and force feedback schemes to handle this issue. However, this issue was not highly prioritized for the following reasons:

Firstly, the control scheme for the manipulator module was not in the hands of the author, and the resulting performance and accuracy of the manipulator was as such uncertain throughout the most part of the work period. It was decided to await more resolute indications regarding the abilities of the manipulator closer to the end of the work period before addressing this matter. Secondly, the manipulator to be used had motors with built-in force feedback sensors, enabling the developer of the manipulator module to implement appropriate safety measures with regards to grasping control and end effector movements. The inherent small size of the manipulator was also concluded to make the robot sufficiently safe to operate in its intended environment, without the need of visual sensory feedback in addition to the force sensors. Lastly, the manipulator was regrettably internally damaged during the development of the manipulator module, and it was for a long period of time uncertain if it would be operable toward the end or not. As the open-loop grasping problem would not be really applicable without a manipulator present, it was considered counterproductive to address this issue at all given the resulting end circumstances.

The choice of objects to be grasped

Grasping different objects with a robotic manipulator has varying complications associated with them. Choosing simpler items to detect and grasp was initially considered for this thesis, but was eventually decided against. It was argued that to continue the investigation and experimentation with types of items for which there is already aquired significant training data was advantageous with regard to classification perfomance. Utilizing an already trained classifier would cut down on a lot of potential time cost, as time is thus not spent gathering and tweaking the classifier to handle other use cases and objects. Closely related to the points considered in the previous section, it was furthermore conluded that it was not of high priority to address issues related to grasping performance, again given the uncertainty regarding the availability and the abilities of the manipulator.

5.1.4 Classification challenges

The results produced by the CNN classifier showed tendencies to generate overconfident and often incorrect predictions for a large number of the clothing articles in different orientations and configurations. One interesting result is the difference in classification output for the baby t-shirt in its natural shape vs. the tucked in sides on its main body. This high degree of certainty for almost every classification attempt would cause significant problems during runtime of the planning algorithm. As the classifier practically exclusively produces highly confident predictions even for exceedingly distorted clothing contours, the system pipeline would as such never utilize the feature detection and manipulator control modules to iteratively identify the correct label of a clothing item.

Furthermore, it was also noted that the orientation of the garments as one would imagine had the greatest impact on classification performance. Pants would for instance be classified correctly in its characteristic orientation, but would inevitably be wrongly labeled as a hoodie or a t-shirt if perceived upside down. This is assumed to be a direct result of the almost complete lack of orientation and mirroring variations in the training data provided by the module, and needs to be thoroughly addressed in further works if one intends to adapt a similar approach.

5.2 Not attempted methods

5.2.1 Camera calibration and point cloud generation

This thesis work utilizes a single camera along with a depth sensor in order to infer spatial relations in an environment. However, it is also possible to use a stereo camera for the same purpose. A stereo camera is just a regular camera with two separate lenses. By carefully investigating and perform calculations on the subtle differences between the two images produced by the pair of imaging sensors, one may determine spatial three dimensional coordinates of the imaged environment. This is called epipolar geometry computing, and is a powerful stereo vision tool. However, imaging sensors usually have slight disturbances present in the form of lense distortions or inaccuracies in the focal length and sensor orientation, and as such needs to be accounted for by the use of camera calibration techniques.

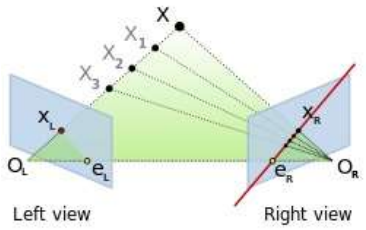


Figure 5.1: This illustration shows a few of the characteristics of epipolar geometry [57].

If such calculations on the intrinsic and extrinsic properties of the imaging sensors are performed, one can ultimately accurately estimate world coordinates in 3D space. Projecting these points onto the original images can thus give considerable spatial insight, and would be a suitable approach for the environment considered in this thesis. However, this approach was not studied as a result of the choice of imaging sensor and its satisfactory performance.

5.2.2 Tilted camera views and complex environments

Testing the classification algorithm on input images with a tilted camera sensor placement was considered in order to evaluate its robustness, but was ultimately dropped due to time constraints. Similarly, the desire to examine the performance of the classifier and planning algorithm in more complex environments using more difficult none-white backgrounds was also not attempted as a result of the already broad scope of the thesis work.

Figure 5.2: Example data sample of an image taken with a tilted view and a complex environment background.

5.2.3 Manipulator detection and closed-loop feedback by classification

Training the convolutional network with data samples of the manipulator end effector in order to detect its position in the image was considered as the preface for a closed-loop feedback mechanism for the manipulator grasping, but was eventually laid aside as a consequence of the situation regarding the damaged manipulator and the prediction difficulties encountered during the development of the sliding window algorithm.

5.2.4 Tweaking the CNN classification parameters

Further experimentation with the CNN classifier and its parameters could possibly improve the classification results for the particular use case considered in this thesis, such as tweaking the training or testing batch sizes as well as the number of epochs trained.

5.3 Future work

5.3.1 Investigating alternative manipulators

As mentioned in Chapter 3.2.4 Workspace limitations and final conclusion, the small size of the manipulator used in this thesis imposed the requirement that the clothing to be classified was children-sized. Larger manipulators with joint force sensors could however be used to manage adult sized clothing, and would be a natural step forward from this work.

5.3.2 Probability distributions for positioning

As was one of the original attempts at solving one of the main objective considered in this thesis, one may further consider the issue of merging probability distributins produced by a classifier to locate the positions of both clothing articles and/or the manipulator end effector. Assuming the workspace is enlarged accordingly, such a system may be capable of sorting even multiple clothing items present in the image.

5.3.3 Grasping points algorithm advancements

Though the detection and grasping planner algorithms produce satisfying results in this work, there's still plenty of potential for more complex coordinate calculations and decision making schemes for targeting grasping features and the calculation of manipulator movements.

5.3.4 System evaluation using a functional manipulator

The proposed feature extraction methods and planning algorithms presented in this thesis could not be properly evaluated using the intended manipulator. This may be interesting to further investigate following the restoration of its damaged joint motors.

Chapter 6

Conclusion

In this work, a complete pipeline of computer vision methods and a novel algorithm for detection and classification of clothing articles applied to a simplified environment is considered. Though there was not enough time available to fully implement the adequately advanced algorithms needed in order to produce a complete system capable of independently and correctly identifying clothing by the help of a physical manipulator and imaging sensors, the proposed approach shows promising potential for further consideration.

The inherent time constraint as a result of the considerable scope of combining three different field areas during the time period of a MSc thesis work was consistently prominent throughout the development of the proposed algorithms, prompting the need for several hard choices to be made regarding the focus and priority of the different aspects of constructing a complete and autonomous system. As the physical manipulator was not operable toward the end of this work, the focus was consequently and ultimately shifted towards the implementation of the detection element of the approach. Tweaking the classification performance of the convolutional neural network module adapted from the author's specialization project was not prioritized, as this module was intended to be used as a separate work along with this thesis. The resulting prediction results are thus not improved beyond the hypothetical assessments discussed throughout this work.

Ultimately, the proposed framework and classification scheme presented in this thesis may form the basis for further work on this topic.

Appendix A

Feature extraction and detection results

A.1 Hoodies

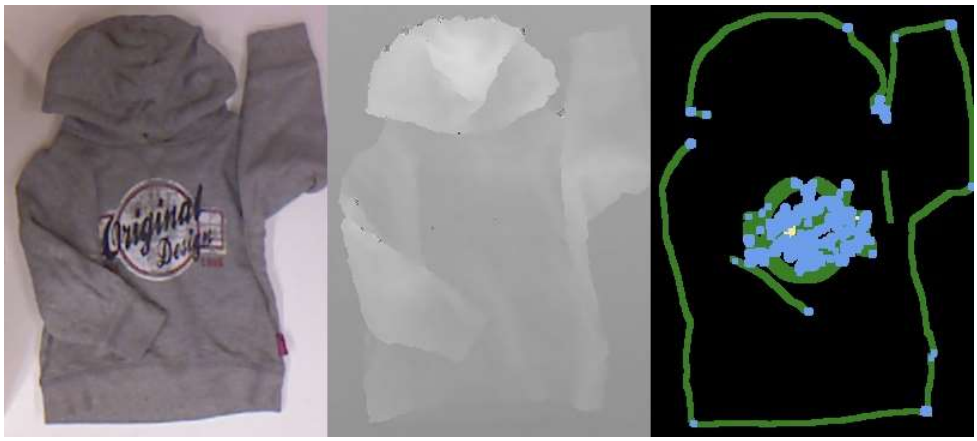


Figure A.1: Hoodie detection result 1

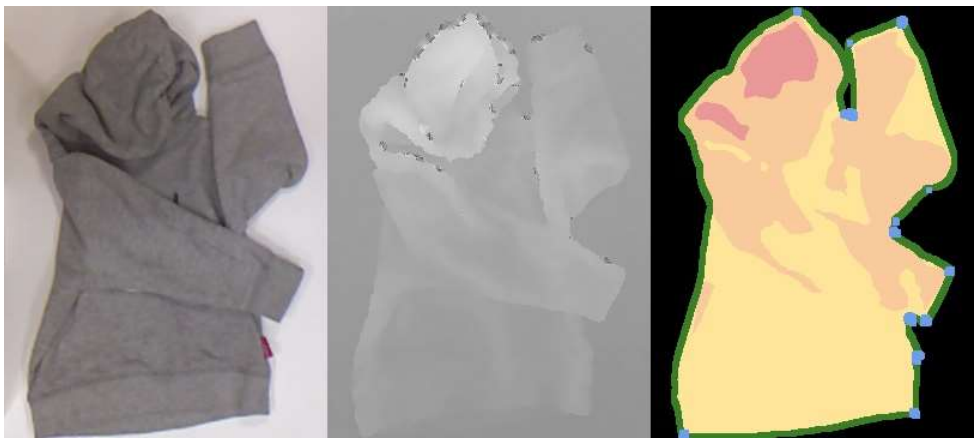


Figure A.2: Hoodie detection result 2

A.2 Pants

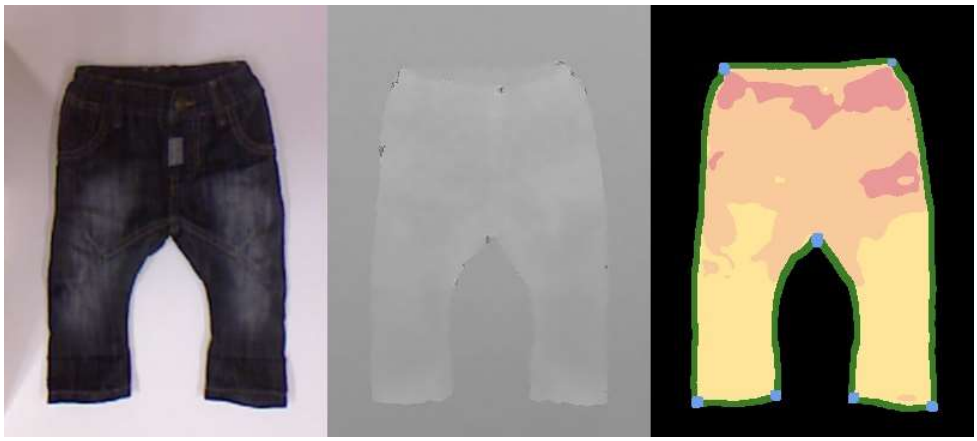


Figure A.3: Pants detection result 1

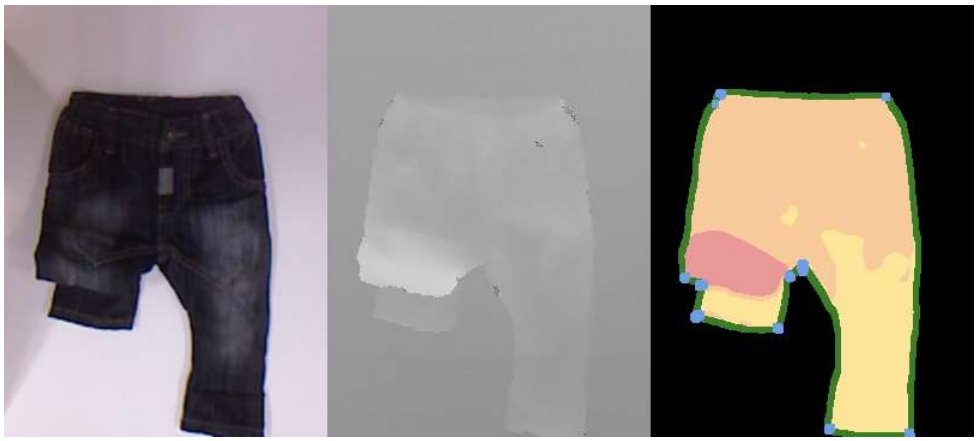


Figure A.4: Pants detection result 2

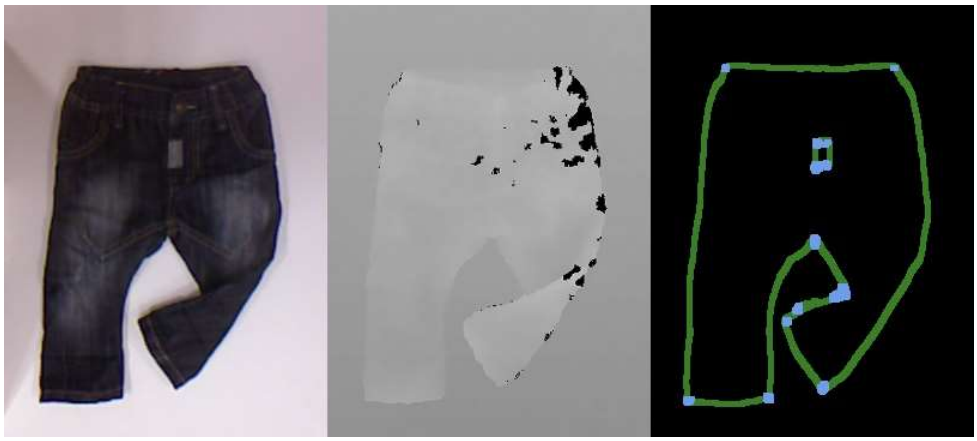


Figure A.5: Pants detection result 3

A.3 T-shirts

Figure A.6: T-shirt detection result 1

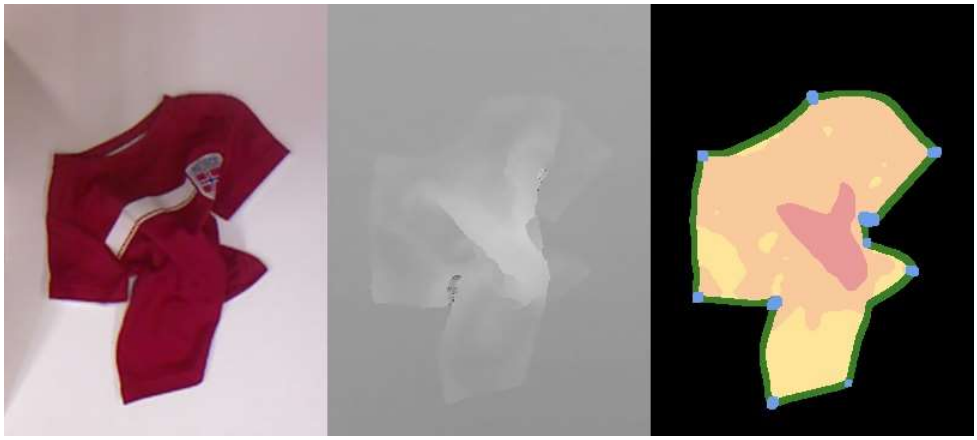


Figure A.7: T-shirt detection result 2

Appendix B

Classifier prediction results

B.1 Hoodies

Figure B.1: Hoodie prediction result 1

Figure B.2: Hoodie prediction result 2

Figure B.3: Hoodie prediction result 3

B.2 Pants

Figure B.4: Pants prediction result 1

Figure B.5: Pants prediction result 2

Figure B.6: Pants prediction result 3

B.3 T-shirts

Figure B.7: T-shirt prediction result 1

Figure B.8: T-shirt prediction result 2

Figure B.9: T-shirt prediction result 3

Appendix C

Installation instructions

C.1 Ubuntu 18.04

Ubuntu 18.04 (Linux) was installed as a dual boot configuration alongside Windows 10 by following the tutorial written by Tecmint user Matei Cezar [58].

C.2 Python 3.6

Python 3.6 was installed using the Anaconda3 5.0.1 package distribution as detailed in Lisa Tagliaferri's setup guide [59].

C.3 OpenKinect

In order to read data from the Microsoft Kinect (V1) camera, the open source drivers package OpenKinect for Python 3 was installed by use of the below Linux Terminal commands, adapted from the instructions described in a blog post [60] and its official documentation [61]:

The following commands install an up to date package manager and dependencies:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install git-core cmake freeglut3-dev pkg-config
    build-essential libxmu-dev libxi-dev libusb-1.0-0-dev
```

Next, the repository is fetched and built using:

```
$ git clone git://github.com/OpenKinect/libfreenect.git
$ cd libfreenect
$ mkdir build
$ cd build
$ cmake -L ..
$ cmake .. -DBUILD_PYTHON3=ON
$ make
$ sudo make install
```

These commands resolve necessary permissions and a rules file:

```
$ sudo adduser $USER video
$ sudo adduser $USER plugdev
$ sudo nano /etc/udev/rules.d/51-kinect.rules
```

Paste the following into the window prompted by the previous command, and save it:

```
# ATTR{product}=="Xbox NUI Motor"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02b0", MODE="0666"
# ATTR{product}=="Xbox NUI Audio"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02ad", MODE="0666"
# ATTR{product}=="Xbox NUI Camera"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02ae", MODE="0666"
# ATTR{product}=="Xbox NUI Motor"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02c2", MODE="0666"
# ATTR{product}=="Xbox NUI Motor"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02be", MODE="0666"
# ATTR{product}=="Xbox NUI Motor"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02bf", MODE="0666"
```

Reboot the system and test the drivers by running:

```
$ freenect-glview
```

Finally, install additional dependencies and python wrappers using:

```
$ sudo apt-get install cython
$ sudo apt-get install python-dev
$ sudo apt-get install python-numpy
$ cd libfreenect
$ cd wrappers
$ cd python
$ sudo /path/to/python/interpreter/.../python setup.py install
```

References

[1] L. Medina, “Mascot Illustration of a Washing Machine Handling a White Clean Shirt,” 2017. [Online]. Available: https://www.123rf.com/profile_lenm

[2] I. Watson, *The Universal Machine*, 1st ed. Copernicus Books, 2012.

[3] J. van den Berg, S. Miller, K. Goldberg, and P. Abbeel, “Gravity-Based Robotic Cloth Folding,” 2010. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-17452-0_24

[4] C. Bersch, B. Pitzer, and S. Kammel, “Bimanual robotic cloth manipulation for laundry folding,” in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2011, pp. 1413–1419.

[5] D. Estevez, J. G. Victores, S. Morante, and C. Balaguer, “Towards robotic garment folding: A vision approach for fold detection,” in *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, May 2016, pp. 188–192.

[6] J. Clark, “Latest Honda Asimo robot makes its European debut,” 2014. [Online]. Available: http://www.carmagazine.co.uk/car-news/first-official-pictures/honda/latest-honda-asimo-robot-makes-its-european-debut/

[7] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka, “The Development of Honda Humanoid Robot,” 1998. [Online]. Available: http://ieeexplore.ieee.org/document/677288/

[8] R. Bogue, “Domestic robots: Has their time finally come?” 2017. [Online]. Available: https://doi.org/10.1108/IR-01-2017-0018

[9] “Laundroid,” Seven Dreamers Laboratories Inc., 2017. [Online]. Available: https://laundroid.sevendreamers.com/en/

[10] “FoldiMate,” FoldiMate Inc., 2017. [Online]. Available: https://foldimate.com/

[11] A. Doumanoglou, J. Stria, G. Peleka, I. Mariolis, V. Petrík, A. Kargakos, L. Wagner, V. Hlaváč, T.-K. Kim, and S. Malassiotis, “Folding Clothes Autonomously: A Complete Pipeline,” 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7589002/

[12] L. Sun, G. Aragon-Camarasa, S. Rogers, and J. P. Siebert, “Robot Vision Architecture for Autonomous Clothes Manipulation,” *CoRR*, vol. abs/1610.05824, 2016. [Online]. Available: https://arxiv.org/abs/1610.05824

[13] W. Ravven, “Deep Learning: A Giant Step for Robots,” 2016. [Online]. Available: https://vcresearch.berkeley.edu/bakarfellows/profile/pieter_abbeel

[14] J. Maitin-Shepard, M. Cusumano-Towner, J. Lei, and P. Abbeel, “Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding,” in *2010 IEEE International Conference on Robotics and Automation*, May 2010, pp. 2308–2315. [Online]. Available: http://ieeexplore.ieee.org/document/5509439/

[15] S. Miller, J. van den Berg, M. Fritz, T. Darrell, and P. Abbeel, “A Geometric Approach to Robotic Laundry Folding,” 2011. [Online]. Available: https://www.researchgate.net/publication/254098952_A_Geometric_Approach_to_Robotic_Laundry_Folding

[16] “Willow Garage,” Willow Garage, 2017. [Online]. Available: http://www.willowgarage.com/

[17] Microsoft, “The Kinect for Windows SDK v1.8,” 2018. [Online]. Available: https://www.microsoft.com/en-us/download/details.aspx?id=40278

[18] ——, “The Kinect for Windows Developer Toolkit v1.8,” 2018. [Online]. Available: https://www.microsoft.com/en-us/download/details.aspx?id=40276

[19] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006.

[20] OpenCV dev team, “Geometric Transformations of Images,” 2014. [Online]. Available: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_geometric_transformations/py_geometric_transformations.html

[21] J. F. Canny, "A Computational Approach to Edge Detection," 1986. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.420.3300&rep=rep1&type=pdf

[22] OpenCV dev team, "Eroding and Dilating," 2017. [Online]. Available: https://docs.opencv.org/2.4.13.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html

[23] ——, "Histograms Equalization in OpenCV," 2015. [Online]. Available: https://docs.opencv.org/3.1.0/d5/daf/tutorial_py_histogram_equalization.html

[24] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 5, pp. 898–916, May 2011.

[25] T. N. J. Z. H. Li, J. Cai, "A benchmark for semantic image segmentation," in *ICME*, 2013. [Online]. Available: http://www.ntu.edu.sg/home/asjfcai/Benchmark_Website/benchmark_index.html

[26] S. S. Ahmad, "Edge detecting in OpenCV and Python," 2017. [Online]. Available: https://shahsparx.me/edge-detection-opencv-python-video-image/

[27] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, July 1959.

[28] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, 1st ed. The MIT Press, 2016.

[29] K. Murphy, *Machine Learning: A Probabilistic Perspective*, 1st ed. The MIT Press, 2012. [Online]. Available: https://mitpress.mit.edu/books/machine-learning-0

[30] H. Tan, Y. Zhou, Y. Zhu, D. Yao, and J. Wang, "Improved river flow and random sample consensus for curve lane detection," 2015. [Online]. Available: https://www.researchgate.net/publication/281733692_Improved_river_flow_and_random_sample_consensus_for_curve_lane_detection?_sg=79o3bN736ZlcRbtXT4ADOG5QW9m1g14_8ErSmftiaasCHVdJIqbo5a8m1GC6dR21nm0HRtFmZA

[31] karpathy@cs.stanford.edu, "CS231n Convolutional Neural Networks for Visual Recognition," 2015. [Online]. Available: http://cs231n.github.io/neural-networks-1/

[32] M. Sqalli, “Traffic signs classification with Deep Learning.” 2016. [Online]. Available: https://hackernoon.com/traffic-signs-classification-with-deep-learning-b0cb03e23efb

[33] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *CoRR*, 2012. [Online]. Available: https://arxiv.org/abs/1207.0580

[34] M. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. Wiley, 2005. [Online]. Available: https://books.google.no/books?id=wGapQAAACAAJ

[35] P. I. Corke, “A simple and systematic approach to assigning denavit ndash;hartenberg parameters,” *IEEE Transactions on Robotics*, vol. 23, no. 3, pp. 590–594, June 2007.

[36] Wikipedia, “Denavit-Hartenberg parameters,” 2018. [Online]. Available: https://en.wikipedia.org/wiki/Denavit%E2%80%93Hartenberg_parameters

[37] ROBOTIS, “DynamixelSDK,” 2018. [Online]. Available: https://github.com/ROBOTIS-GIT/DynamixelSDK

[38] icyimage, “Housewife Putting Clothes on Available Space,” 2018. [Online]. Available: https://www.shutterstock.com/image-photo/housewife-putting-clothes-on-available-space-12337672

[39] A. Rosebrock, “Sliding Windows for Object Detection with Python and OpenCV,” 2015. [Online]. Available: https://www.pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv/

[40] OpenCV dev team, “Image Pyramids,” 2018. [Online]. Available: https://docs.opencv.org/2.4/doc/tutorials/imgproc/pyramids/pyramids.html

[41] P. team, “3D Surface Plots in R,” 2018. [Online]. Available: https://plot.ly/r/3d-surface-plots/

[42] A. Mordvintsev, “Introduction to OpenCV-Python Tutorials,” 2013. [Online]. Available: http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_setup/py_intro/py_intro.html

[43] Wikipedia, "Comma-separated values," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Comma-separated_values

[44] F.-R. Stöter, "Keras Utils Documentation," 2017. [Online]. Available: https://faroit.github.io/keras-docs/1.1.1/utils/np_utils/

[45] "Convolutional Neural Networks (LeNet)," DeepLearning, 2017. [Online]. Available: http://deeplearning.net/tutorial/lenet.html

[46] "Getting started with the Keras Sequential model," Keras, 2017. [Online]. Available: https://keras.io/getting-started/sequential-model-guide/

[47] F. Tencé, "Keras CNN (inspiret by LeNet-5)," 2016. [Online]. Available: https://www.kaggle.com/ftence/keras-cnn-inspired-by-lenet-5

[48] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *CoRR*, vol. abs/1609.04836, 2016. [Online]. Available: http://arxiv.org/abs/1609.04836

[49] karpathy@cs.stanford.edu, "CS231n Convolutional Neural Networks for Visual Recognition," 2017. [Online]. Available: http://cs231n.github.io/neural-networks-3/

[50] A. 3D, "The Artec Studio 12 software documentation," 2017. [Online]. Available: http://docs.artec-group.com/as/12/en/

[51] A. Inc., "Autodesk Fusion 360," 2018. [Online]. Available: https://www.autodesk.com/products/fusion-360/overview

[52] OpenCV dev team, "Image Inpainting," 2013. [Online]. Available: http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_photo/py_inpainting/py_inpainting.html

[53] ——, "Interactive Foreground Extraction using GrabCut Algorithm," 2016. [Online]. Available: https://docs.opencv.org/3.2.0/d8/d83/tutorial_py_grabcut.html

[54] ——, "Image Segmentation with Watershed Algorithm," 2015. [Online]. Available: https://docs.opencv.org/3.1.0/d3/db4/tutorial_py_watershed.html

[55] M. M. S. J. Preetha, L. P. Suresh, and M. J. Bosco, "Image segmentation using seeded region growing," in *2012 International Conference on Computing, Electronics and Electrical Technologies (ICCEET)*, March 2012, pp. 576–583.

[56] OpenCV dev team, "Harris corner detection," 2014. [Online]. Available: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html

[57] R. Anand, "Obstacle Detection using Stereo Vision," 2010. [Online]. Available: http://ewh.ieee.org/r4/se_michigan/Fall2010/speakers.html

[58] M. Cezar, "How to Install Ubuntu 16.10/16.04 Alongside With Windows 10 or 8 in Dual-Boot," 2016. [Online]. Available: https://www.tecmint.com/install-ubuntu-16-04-alongside-with-windows-10-or-8-in-dual-boot/

[59] L. Tagliaferri, "How To Install the Anaconda Python Distribution on Ubuntu 16.04," 2017. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-install-the-anaconda-python-distribution-on-ubuntu-16-04

[60] "Experimenting with Kinect using opencv, python and open kinect (libfreenect)," 2014. [Online]. Available: https://naman5.wordpress.com/2014/06/24/experimenting-with-kinect-using-opencv-python-and-open-kinect-libfreenect/

[61] OpenKinect, "OpenKinect," 2018. [Online]. Available: https://github.com/OpenKinect/libfreenect