Universitetet
i Stavanger

**DET TEKNISK-NATURVITENSKAPELIGE FAKULTET**

# MASTEROPPGAVE

| | |
|---|---|
| Studieprogram/spesialisering:<br>Informasjonsteknologi – kybernetikk/signalbehandling | Vårsemesteret, 2012<br>Åpen |
| Forfatter:<br>Espen Hatlestad | …………………………………………<br>(signatur forfatter) |
| Fagansvarlig: Kjersti Engan<br>Veileder: Kjersti Engan | |
| Tittel på masteroppgaven:<br>Objekt Klassifisering ved bruk av LEGO Mindstorms NXT og MATLAB<br>Engelsk tittel:<br>Object Classification using LEGO Mindstorms NXT and MATLAB | |
| Studiepoeng: 30 | |
| Emneord:<br>LEGO Mindstorms NXT, background subtraction, background modelling, mathematical morphology, object representation, feature extraction, Maximum Likelihood estimation, Parzen Window estimation, $k_n$-Nearest-Neighbor estimation, cross validation | Sidetall: 148<br>+ vedlegg/annet: 64 + CD<br>Stavanger, 13.06.2012<br>dato/år |

# Summary

This master thesis involves further development of the physical sorter system that was developed in the preliminary project (MIK110). The system is built up by LEGO Technic parts and is controlled by MATLAB and LEGO Mindstorms NXT. The system basically consist of a conveyor belt, a sorter arm, a web camera and control system (MATLAB). The conveyor belt transfers desired objects from one end to the other and are controlled by a servo. The sorter arm can push a object off the conveyor belt and into a tray. The sorter arm is also controlled by a servo. Both servos are connected to the NXT Intelligent Brick (NXT). The web camera detect objects moving on the conveyor belt. This web camera is connected to a computer running MATLAB. MATLAB runs the system (connected to NXT) and manages the handling of detected objects. To get MATLAB and LEGO Mindstorms NXT to interact, a toolbox called RWTH Toolbox is implemented. All computational task are handled in MATLAB.

The web camera is controlled by a function called `MOD`. This function enables the web camera to capture one frame at a time. To detect movement from one frame to another a method called background subtraction (BS) is used. Since `MOD` is intended to only detect moving objects and not the movement of the conveyor belt, which creates a dynamic background, a method called background modeling (BM) is implemented. The BM method creates a model of the background and slowly updates the model when a new frame is captured. This method suppresses movement and illumination/reflection from the conveyor belt which can produce false positive detections.

The frame containing the object of interest (OOI) is then pre processed by a function called `ooi`. This function is solely based on mathematical morphology and uses this to register, extract and represent the OOI. Representation includes both internal (area) and external (boundary/signature) characteristics, and these are used as features to describe the object. From a object, four different features are computed and they are the area, the mean and standard deviation computed from the signature and the amount of vertices/peaks in the signature. All features are gathered in a feature vector X and all features are normalized.

There are two different objects that the system have to detect. These are two different types of LEGO Technic gears. Both have 24 teethes, but one gear is crowned and the other is normal. The features collected from these objects are very much a like. A classifier is implemented in the system to distinguish between these two objects and solve this 2-class problem. A method called *Cross validation* was used in the experiments to find the best classifier. The cross validation was performed on a three different classifiers which were based on Maximum Likelihood (ML) estimation, the Parzen Window technique and $k_n$-nearest-neighbor (kNN) technique. The data set (consisting of gathered feature vectors from both objects) that each classifier was trained on, was 2-, 3- and 4-dimensional for each experiment. Also, in each experiment, raw and normalized data was used.

The classifier that was implemented in the system, based on the results from the cross validation, was the kNN classifier for normalized 4-dimensional data. From the cross validation results, this classifier had a error rate of 6.43%, but this error rate is affected by illumination. Experiments showed that the error rate varies between 3 and 10% when the classifier was implemented in the system and new feature data was gathered.

On YouTube there is a short video demonstration of the system when it is running. The clip can be found via this URL: http://www.youtube.com/watch?v=AAGNuTu7Tfk&feature=plcp.

# Preface

This report shows the results of the work carried out in the master thesis for the spring of 2012 at the University of Stavanger.

I would like to thank my supervisor, Professor Kjersti Engan, for all the ideas and all the encouragement she has given me during the last six months. Her guidance has been of immense help.
I would also like to thank Professor Trygve Eftestøl for all his help regarding the subject pattern recognition.

And finally, I would like to express a big thanks to my girlfriend, Vibeke, who has had many evenings alone at home, but who nevertheless have shown understanding for my absence.

Stavanger, June 2012

Espen Hatlestad

# Notations

| | |
|---|---|
| $\alpha$ | Learning coefficient (adapt coefficient) |
| $\sigma_B^2$ | Between-class variance |
| AT | Average Threshold |
| $BC$ | Border Coordinates |
| $B_i$ | Current background model |
| $B_{i-1}$ | Previous background model |
| $BM$ | Background Model |
| $BS$ | Background Subtraction |
| $c$ | Column vector |
| $Comp$ | Image Component (**R, G or B**) |
| D | Data set |
| $DET$ | Detecting mode |
| ER | Error Rate |
| $F_{fd}$ | Frame-result after frame differencing |
| $F_i$ | Current frame |
| $F_{i-1}$ | Previous frame |
| $F_{RGB}$ | RGB Frame |
| FOV | Field Of View |
| GUI | Graphical User Interface |
| $i_{max}$ | Index to maximum value in a vector |
| $junk$ | Unwanted information (not used) |
| $L$ | Number of grayscale intensities (256) |
| $LM$ | Labelled Matrix |
| $m$ | Vertical pixel coordinate |
| $M0$ | 0-matrix (size: $350 \times 705$) |
| $M_{rm}$ | Matrix containing RM (size: $360 \times 705$) |
| MOD | Moving Object Detection |
| $n$ | Horizontal pixel coordinate |

| | |
|---|---|
| NOCC | Number Of Correct Classifications |
| NoP | Number of Peaks |
| NOWC | Number Of Wrong Classifications |
| $n_{rp}$ | Horizontal pixel coordinate reference point (image) |
| NXT | NXT Intelligent Brick |
| $OData$ | Object Data (Area, centroid and bounding box) |
| OOI | Object Of Interest |
| $r$ | Row vector |
| RM | Reference Mask |
| ROI | Region Of Interest |
| SE | Structural Element |
| Std | Standard Deviation |
| $T$ | Threshold |
| TS | Training Set |
| $UDT$ | User-Defined Threshold |

# List of Figures

# List of Tables

# List of Algorithms

# Contents

# Chapter 1

# Introduction

This report presents further work that has been done based on the preliminary project, *Objekt Sorterer - Egenskaps Uttrekning* (MIK110).
The background for the preliminary project and this report is based on an idea that Professor Kjersti Engan at the University of Stavanger presented in the autumn of 2011. The idea was to combine MATLAB and LEGO Mindstorms, and use it in a educational context to future students. To make this idea useful, the issues concerning both reports had to be based on subjects that are taught at Information Technology, cybernetics and signal processing.

The solution was to develop a conveyor belt system which detects two different objects and sort them into different trays. A web camera is used to detect the objects moving on the conveyor belt and capture an image containing it. MATLAB is used to pre process the image, find the object of interest, compute features and classify the object. MATLAB also controls the NXT which runs the conveyor belt and controls the sorter arm, by two servos.

Figure 1.1 shows the physical sorter system.

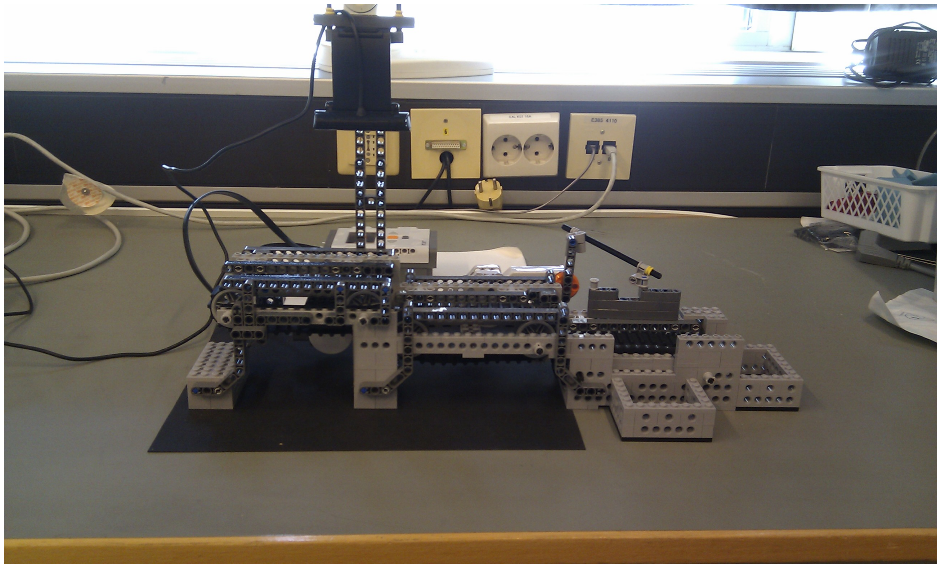


Figure 1.1: The developed sorter system.

In the preliminary project the main tasks was to

- Build a physical sorter system out of LEGO Technic parts.
- Get familiar with LEGO Mindstorms and the RWTH Toolbox.
- Implement connection between Mac OS X and NXT.
- Get familiar with the Image Acquisition and Image Processing Toolbox in MATLAB.
- Preliminary work on how to detect objects in a image and gather useful features from objects.

The main issues concerning this report deals with

1. Develop an algorithm that automatically detects moving objects on the conveyor belt, by using the web camera as a frame grabber and as a sensor.
2. Improving the preliminary work on how to extract an object from an image/frame.
3. Develop/train a optimal classifier which is used to distinguish between two objects.

The first two issues, in this report, is based on image acquisition and image processing theory. Image processing is a important course at Information Technology and teaches students the fundamental methods/ideas about image processing and image processing in MATLAB.
The last issue is based on pattern recognition theory which applies math and probability methods to design/train classifiers. A random classifiers purpose is to classify data to a class based on qualified guesses. Pattern recognition was an optional course at Information Technology when the writer went here, but it is highly recommended and a useful subject.

## 1.1 Overview

All chapters below are listed chronological and gives a short review of content.

**Chapter 2 Basic Theory**
A brief explanation of the fundamental theory that the work is based on. Sections 2.3 and 2.4 is based on the work from the preliminary project, *Objekt sorterer*.

**Chapter 3 A System Overview**
A brief explanation of the systems physical and program-based (MATLAB) modules.

**Chapter 4 Moving Object Detection**
This chapter presents the moving object detection module. It's development, the conducted experiments and results from the experiments.

**Chapter 5 Object Representation and Feature Extraction**
This chapter presents the object of interest module's development, conducted experiments and results. It also shows how objects are represented and how features are extracted.

**Chapter 6 Feature Classification and Object Recognition**
A presentation of the experiments done on three different classifiers. The final classifier is chosen from a set of criterias and from the result of cross validation.

**Chapter 7 Explanation of System Functionalities**
Flowcharts and explanation to `main.m` and functions included in `main.m`.

**Chapter 8 Experiments & Results**
A presentation of the results, experiments and adjustments to the whole system.

**Chapter 9 Conclusion & Further Development**

**Appendix A Content on CD**

**Appendix B Set Theory**
Some basic concepts behind set theory which is the foundation for methods like morphological erosion, dilation, opening and closing.

**Appendix C Calculations**
Some calculation examples.

**Appendix D Implementation of Software and NXT**
Step-by-step guides on how to get started with MATLAB and NXT on Mac. This chapter is based on the work from the preliminary project, *Objekt sorterer*.

**Appendix E Implementations in MATLAB**
A overview of all functions implemented in MATLAB.

**Appendix F Data Collection from Cross Validation**
Collected data from each classifier with normalized and unnormalized data.

# Chapter 2

# Basic Theory

This chapter presents some of the theory behind the work that is performed in this report. Every section refers to the specific chapter that is based on the current theory.

## 2.1 Region Of Interest

Region of interest (ROI) is a area in an image that contains the information that is relevant for further analyze and processing. By only focusing on this area, the rest of the image isn't used and is discarded. To enable ROI in an image there are four constants that the user needs to declare and that is the x- and y-offset, also called $n_0$ and $m_0$, and the width ($n_1$) and height ($m_1$) of the ROI. Figure 2.1 shows an example.

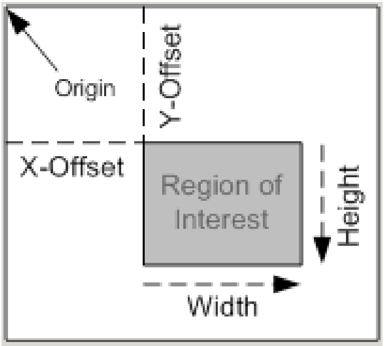


Figure 2.1: Region of Interest. © 2012 The MathWorks, Inc. All rights reserved.

The coordinates $n_0$ and $m_0$ decides the new image origin, or upper left corner, and the width and height decides the resolution of the ROI.
This section only covers squared representation of an ROI. There are several other techniques that can be implemented to shape the ROI in a arbitrary user defined shape.

## 2.2 Thresholding

Thresholding are mainly used on grayscale images to distinguish and extract f.ex. an object from a background in the image scene. Each pixel in a grayscale image contains information about the intensity and this intensity can vary from black (0) to white (255). The area between black and white contains different shades of grey. Every pixel are assigned a value based on

their intensity level and these values can be between 0 and 255.
If a grayscale image contains a bright object in a dark environment, a histogram could group together intensity levels that have the same value and give an impression of which values that are lighter and darker.
Figure 2.2 shows a histogram that clearly distinguishes two different sets of intensity levels.

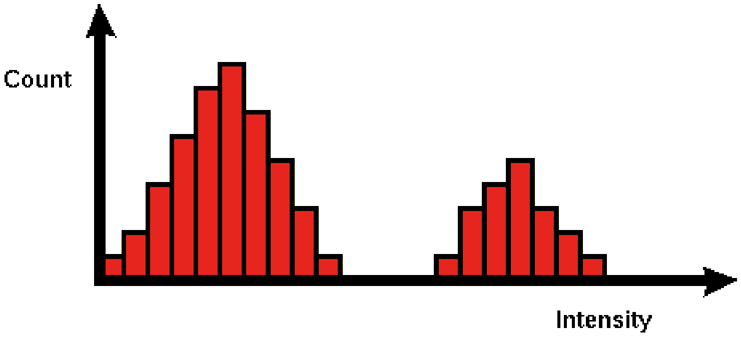


Figure 2.2: Intensity histogram. The intensity levels are lower for the left group then for the right.

The x-axis contains the intensity levels from 0 to 255 and the y-axis sums up each pixel in the image that have a given intensity value.
The group to the left in the figure above contains all the pixels in the image that are dark and/or dark shades of grey. The group to the right contains the pixels that are bright. Since the object is brighter than the background, the group of intensity levels to the right in the image represents the object.
To extract the object from the background, the threshold (T) should be a intensity value between the two grouped sets and the value is always between 0 and 1, because the value is divided by 255.
There are different ways to compute thresholds in grayscale images and one of those is called Otsu's method. This method is used by the MATLAB function `graythresh` and are explained in the next section.

### 2.2.1 Otsu's method

According to [5] (page 561), a histogram of an image groups together pixels that have the same intensity level from 0 to 255 (grayscale values). To classify one set ($S_1$) of pixels with values {0, 1, 2, ...., k} from another set ($S_2$) with values {k+1, k+2, ...., L-1}, where L equals 256, can be done by the threshold k. Otsu's method chooses the value of k that maximizes the *between-class variance* $\sigma_B^2(k)$. The mathematical expression for the between-class variance is defined as

$$\sigma_B^2(k) = \frac{\left[m_G P_1(k) - m(k)\right]^2}{P_1(k)\left[1 - P_1(k)\right]} \tag{2.1}$$

, where $m_G$ is the mean of all intensity levels in the image, $P_1(k)$ is the probability of $S_1$ occurring, $m(k)$ is the mean intensity up to level k and $1 - P_1(k)$ is the probability of $S_2$ occurring.
The larger $\sigma_B^2(k)$ is, the more likely it is that the threshold k will segment the image properly.

Thresholding is used in Chapter 4 to convert grayscale images to binary.

## 2.3 Morphological Operations

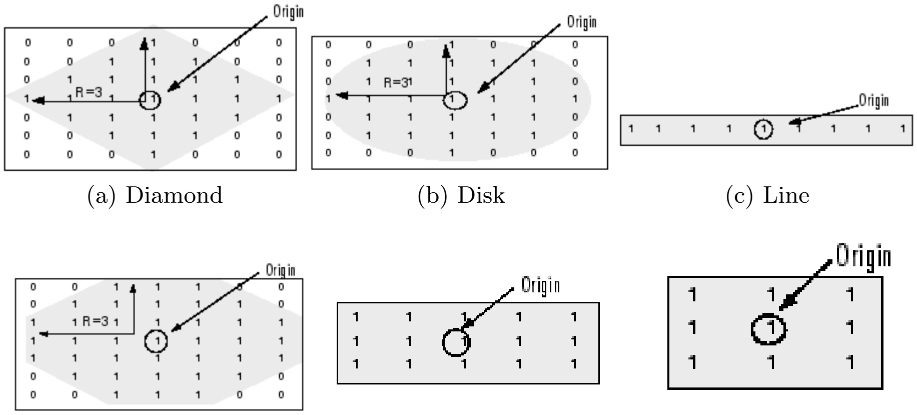
Mathematical morphology (MM) is a technique that is used to analyze and process geometrical structures in images. It is used to extract information about objects in an image, but it is also used to pre- and post process images. The theory underlying this section mainly focuses on two fundamental operations in MM which are *dilation* and *erosion*, and it's solely based on information from [5].

Chapter 5 are based on this theory.

Equations 2.2, 2.3, 2.4 and 2.5 are based on set theory. Some basic concepts from this theory is presented in appendix B.

### 2.3.1 Dilation

Dilation increases or thickens objects in a binary image. The extent of the object growth after dilation is determined by a so called *structural element* (SE) [5]. A SE may have many different shapes and some shapes are shown in figure 2.3.



(a) Diamond (b) Disk (c) Line

(d) Octagon (e) Rectangle (f) Square

Figure 2.3: Structural elements. © 2012 The MathWorks, Inc. All rights reserved.

The MATLAB function `strel` can be used to create these elements. The user can determine the shape, radius, length, height and/or angle of the element.
When used with dilation, the SE is shifted across every pixel element in the binary image and for every 1-valued pixel (foreground pixel) the SE's origin overlaps, it expands this element with the size/shape of the SE.

The mathematical expression [5] for dilation can be expressed as follows

$$A \oplus B = \left\{ z \middle| (\hat{B})_z \cap A \neq \emptyset \right\} \tag{2.2}$$

, where A is a collection of connected 1-valued elements, B is the SE and $\emptyset$ is the empty set.

### 2.3.2 Erosion

Erosion operates opposite of dilation by shrinking objects. To shrink an object (a set of connected 1-valued elements) or a part of it, the whole SE must overlap the object or this part. If a horizontal line shaped element with a length of 3 pixels hits a single 1-valued element that is not connected to any other 1-valued elements, the element is removed. But if it hits 3 connected 1-valued elements in the same direction as the SE, only the element that is overlapped by the origin remains after erosion.

The mathematical expression [5] for erosion can be expressed as follows

$$A \ominus B = \left\{ z \middle| (B)_z \cap A^c = \emptyset \right\} \tag{2.3}$$

, where the erosion of A by B is when all elements in B overlaps A.

### 2.3.3 Opening

Opening first performs erosion on a binary image and the result is then subjected to dilation. The operation first remove objects that are smaller then the SE and reduces objects that are larger, then it increases the remaining objects to their original size.

The mathematical expression [5] for opening can be expressed as follows

$$A \circ B = (A \ominus B) \oplus B \tag{2.4}$$

### 2.3.4 Closing

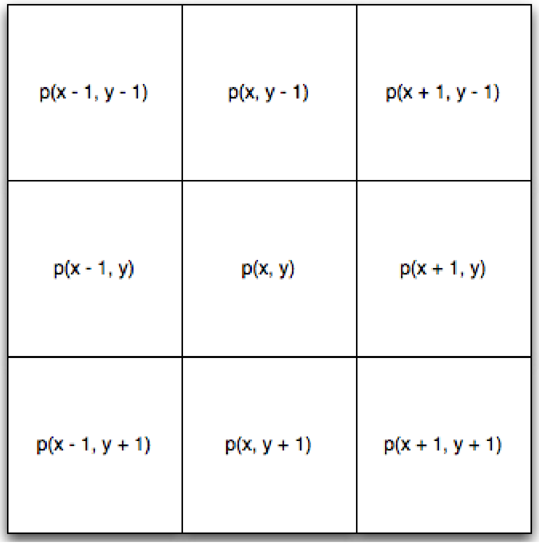
Closing first performs dilation on a binary image and the result is then subjected to erosion. The result of closing smooths object perimeters, but it doesn't remove objects that are smaller than the SE, because dilation is performed before erosion [5].

The mathematical expression [5] for opening can be expressed as follows

$$A \bullet B = (A \oplus B) \ominus B \tag{2.5}$$

### 2.3.5 Labeling

Every pixel element in a binary image have the value 1 or 0. Foreground pixels, or objects, have the value 1 and are white, background pixels have the value 0 and are black. A set of foreground pixels in a neighborhood are defined as *connected components* [5]. To determine which pixels that are connected together as a component, the connection between each pixel in the neighborhood have to be defined. The figure below shows 3-by-3 neighborhood of pixels and their coordinates.



| | | |
|---|---|---|
| p(x - 1, y - 1) | p(x, y - 1) | p(x + 1, y - 1) |
| p(x - 1, y) | p(x, y) | p(x + 1, y) |
| p(x - 1, y + 1) | p(x, y + 1) | p(x + 1, y + 1) |

Figure 2.4: A 3-by-3 pixel neighborhood. The text inside each square refers to the pixel coordinate.

Pixel p(x, y) have two vertical and two horizontal neighbors, and this neighborhood is defined as $N_4$(p). Another neighborhood to p(x, y) is it's diagonal neighbors; p(x+1, y-1), p(x-1, y+1), p(x-1, y-1) and p(x+1, y+1). This neighborhood is defined as $N_D$(p). There is also a defined a third neighborhood that is a combination of the two others and it's called $N_8$(p). $N_8$(p) is

the union of $N_4(p)$ and $N_D(p)$.
All foreground pixels that have the specified connection (set by user) are labelled with the same number. Figure 2.5 shows a matrix that represents a binary image. Figure 2.6 and 2.7 show the labeling of this image with both $N_4(p)$ and $N_8(p)$ connection.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |

Figure 2.5: Matrix representing a binary image. All 0-valued pixels is defined as the background.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 5 | 5 | 5 | 0 | 0 |
| 1 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 7 | 7 |
| 0 | 3 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 3 | 3 | 3 | 0 | 0 | 0 | 8 | 0 |
| 2 | 0 | 0 | 0 | 3 | 0 | 0 | 6 | 0 | 0 |

Figure 2.6: Labeling connected components with a 4-connected neighborhood ( $N_4(p)$ ).

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 2 | 2 | 2 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 2 | 2 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 3 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 3 | 0 | 0 |

Figure 2.7: Labeling connected components with a 8-connected neighborhood ( $N_8(p)$ ).

The different gray colors and numbers represents connected components found in the image when using 4- and 8-connected neighborhood. When using a 8-connected neighborhood, the labeling finds less objects in the image.

## 2.4 Object Representation

There are two common options on how to represent objects (also known as image regions) mathematically. These are often used in MATLAB and are called internal and external characteristics [5]. Chapter 5 is based on this theory.

### 2.4.1 Internal Characteristics

Internal characteristics of an object focuses on regional properties [5] like

- Object color
- Surface texture
- Object area
- Object centroid (coordinate of the objects center of gravity)
- Length / height of object

The MATLAB function `regionprops` is used to extract some of these characteristics from an object and it will be represented in section 5.2.1.

### 2.4.2 External Characteristics

The external characteristics of an object focuses on object shape. [5]. These properties can be

- Object perimeter / boundary
- Signature
- Chain codes
- Skeletons

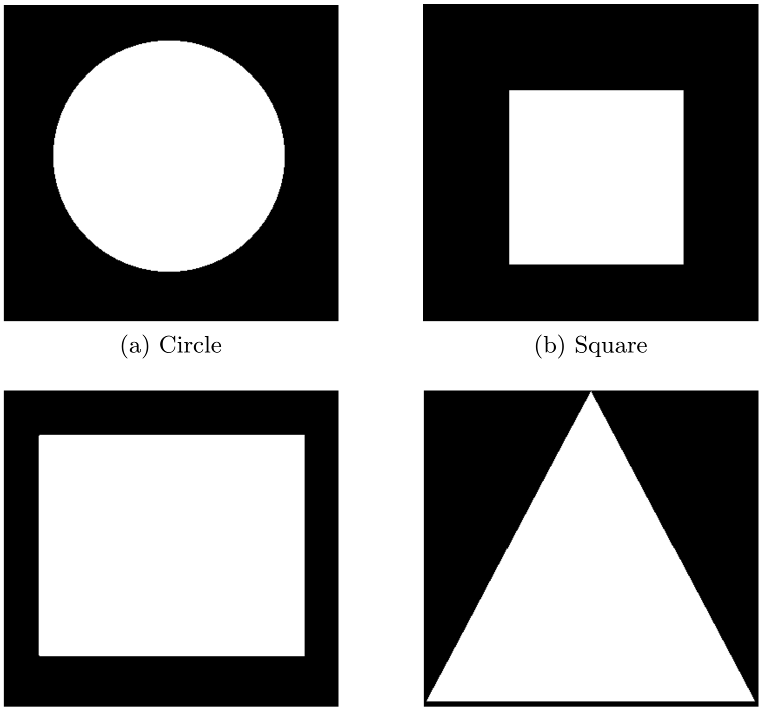
Finding the object perimeter / boundary is represented in section 5.2.2.

#### 2.4.2.1 Signature

A signature is a representation of an objects boundary in a one dimensional room. One option to represent a signature is to plot the distance, D (Q-by-2 vector), from an objects centroid to it's boundary as a function of the angle $\theta$ [5]. Equation 2.6 shows the mathematical definition

$$\vec{S} = \vec{D}(\theta) \tag{2.6}$$

The angle $\theta$ is rotated 360 degrees (one degree increment) and the signature $\vec{S}$ (360-by-2 vector) contains all distances from the centroid to the perimeter. This approach makes the analysis of an object form easier to understand. Figure 2.8 shows four common geometrical shapes and figure 2.9 shows their respective signatures.



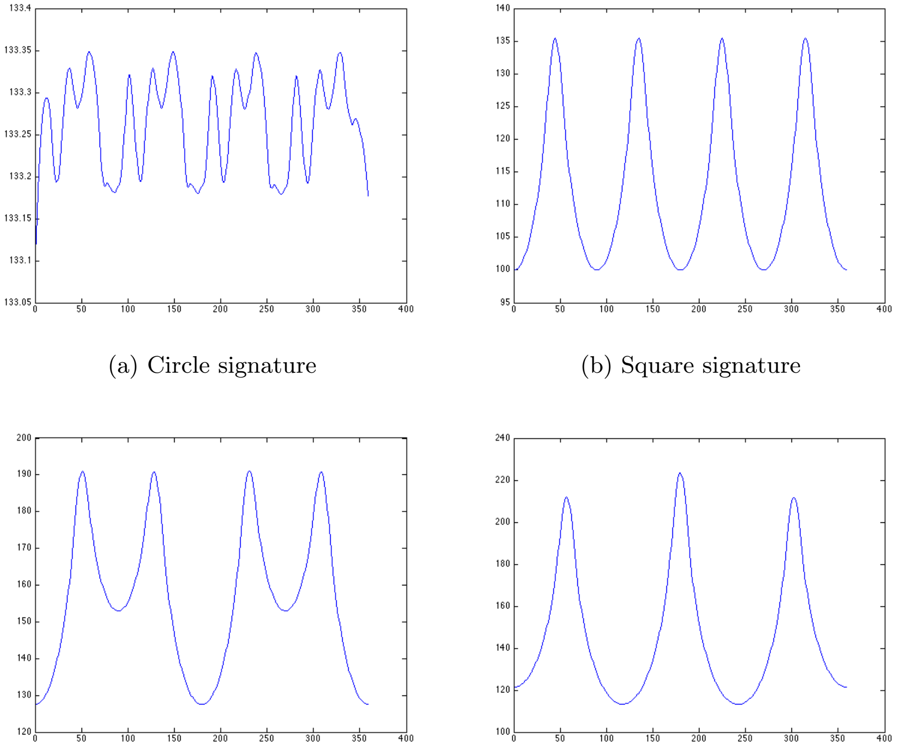
(a) Circle

(b) Square

(c) Rectangle

(d) Triangle

Figure 2.8: Four common geometrical shapes.



(a) Circle signature

(b) Square signature

(c) Rectangle signature

(d) Triangle signature

Figure 2.9: The signatures for each of the four geometrical shapes presented in figure 2.8.

The first and last point in all signatures is actually connected.
The signature for the circular object should in theory be a straight line because the distance from the object center to the boundary is the circles radius (constant for any $\theta$), but in practice it's very difficult to extract a perfect circle perimeter and hence the signature varies in figure 2.9a.
From figure 2.9b all four peaks, representing the square corners, in the signature are equal an the distance between each peak is approximately 90 degrees, which is two of the squares geometrical properties. The valleys between the peaks indicates that all sides in the square have the same length.
In figure 2.9c there are also four equal peaks (in height), the two small valleys are the two opposite sides (vertical) and the two large valleys are the

diagonal sides (horizontal) that represents a rectangle.
The last figure represents a triangle. There are three peaks in the signature that is the triangle corners and three valleys that represent the sides / edges. The value of the peak in the middle indicates that it is not a equilateral (all the angles are 60°) triangle, but a isosceles triangle (45-45-90°).

All valleys in all signatures, in figure 2.9, refers to sides / edges of the object, except for the circular object. The data gathered from the signature is used as features that describes an object. This is presented in section 5.2.3.

## 2.5 Pattern Recognition

Pattern recognition provides techniques to classify input data to a number of classes or categories. The authors in [3] defines pattern recognition as *the act of taking raw data and taking an action based on the "category" of the pattern.*

This section provides fundamental and basic theory to explain some areas of pattern recognition. All information in this section is mainly based on [3] and the theory is fundamental to Chapter 6.

### 2.5.1 Bayes Decision Theory

For several pattern classification problems the Bayes decision theory is a important statistical set of tools. To use this theory to solve a problem, all probability values must be known [3]. This section describes some of the Bayes decision theory's fundamental definitions.

**Class(es)**

$$\omega = \{\omega_1, \omega_2, \ldots, \omega_c\} \tag{2.7}$$

Example:
Two-class problem. Distinguish glass bottles ($\omega_1$) from plastic bottles ($\omega_2$).

**A priori probability**
Our prior knowledge about which bottle is more probable to come next.

$P(\omega_1)$ express the probability that the next bottle belongs to class one and $P(\omega_2)$ is the probability that the next bottle belongs to class two.
Independent of how many classes there are, all a priori probabilities sum to one.
Decision rule:

$$\text{Decide} = \begin{cases} \omega_1 & \text{if } P(\omega_1) > P(\omega_2) \\ \omega_2 & \text{otherwise} \end{cases} \tag{2.8}$$

**Class-conditional Probability Density Function (PDF)**
If an observation is used to assist the decision of choosing which bootle is next, an pdf can be used and it is defined as $p(x|\omega_j)$.
$p(x|\omega_j)$ is simply the probability density function for $x$ given that the state of nature is $\omega_j$. For the the bottle example, there are two pdf's, one for $\omega_1$ and one for $\omega_2$. The difference between these two pdf's describes the difference between a glass bottle and a plastic bottle.
Figure 2.10 shows an example of what the pdf's for $\omega_1$ and $\omega_2$ might look like.

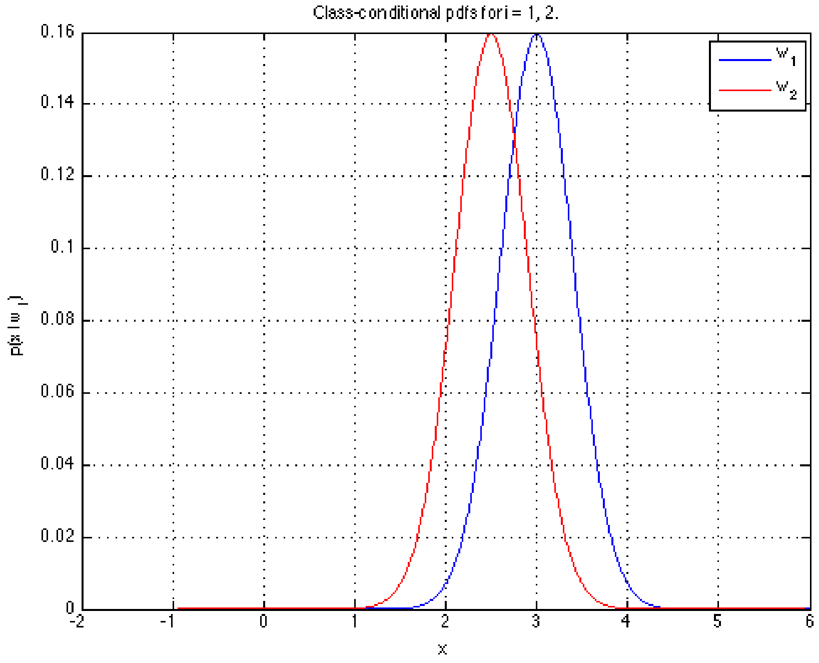


Figure 2.10: Example: Two class-conditional pdf's for $\omega_1$ and $\omega_2$.

The observation $x$ is a continuous random variable, also called a feature, and in this case it can represent the light reflection from the bottles. There could also be more features from each bottle that assist the choosing decision and then the observation $x$ gets a higher dimension.

$$\underline{x} = [x_1 \; x_2 \; x_4 \; \ldots \; x_d]^T \tag{2.9}$$

The subscript $d$ represents the dimension of the feature vector $\underline{\mathrm{x}}$ in equation 2.9. The observation x is not longer a scalar, but a feature vector in a $d$-dimensional Euclidean space, $\mathbf{R}^d$.

The class-conditional pdf can f. ex. be normally distributed, i.e. $p(x|\omega_j) \sim N(\mu_j, \sigma_j)$.

**Bayes Formula**

The Bayes formula can be found by putting together the previous definitions.

$$P(\omega_j|\underline{x}) = \frac{p(\underline{x}|\omega_j)P(\omega_j)}{p(\underline{x})} \tag{2.10}$$

This formula is used when $p(\underline{x}|\omega_j)$ and $P(\omega_j)$ is known. It expresses the posteriori probability that the state of nature is $\omega_j$ when the feature vector $\underline{\mathrm{x}}$ is measured.

$p(\underline{x})$ is called the evidence factor, or scaling factor, and is defined as

$$p(\underline{x}) = \sum_{j=1}^{c} p(\underline{x}|\omega_j)P(\omega_j) \tag{2.11}$$

A decision rule that minimizes the probability of error can be formulated from equation 2.10.

$$P(error|\underline{x}) = \begin{cases} P(\omega_1|\underline{x}) & \text{if } \omega_2 \text{ is choosen} \\ P(\omega_2|\underline{x}) & \text{if } \omega_1 \text{ is choosen} \end{cases} \tag{2.12}$$

and this formula supports the Bayes decision rule

$$\text{Decide } = \begin{cases} \omega_1 & \text{if } P(\omega_1|\underline{x}) > P(\omega_2|\underline{x}) \\ \omega_2 & \text{if } P(\omega_2|\underline{x}) > P(\omega_1|\underline{x}) \end{cases} \tag{2.13}$$

Both equation 2.12 and 2.13 is formulated for a two-class problem. Its not necessary to include $p(\underline{x})$ in the decision rule because this is just a scaling factor and it doesn't change the posteriori probability.

### Decision Border and Error Probabilities

The decision border is a line that separates the two pfd's where $p(x|\omega_1) = p(x|\omega_2)$. Figure 2.11 shows an example.

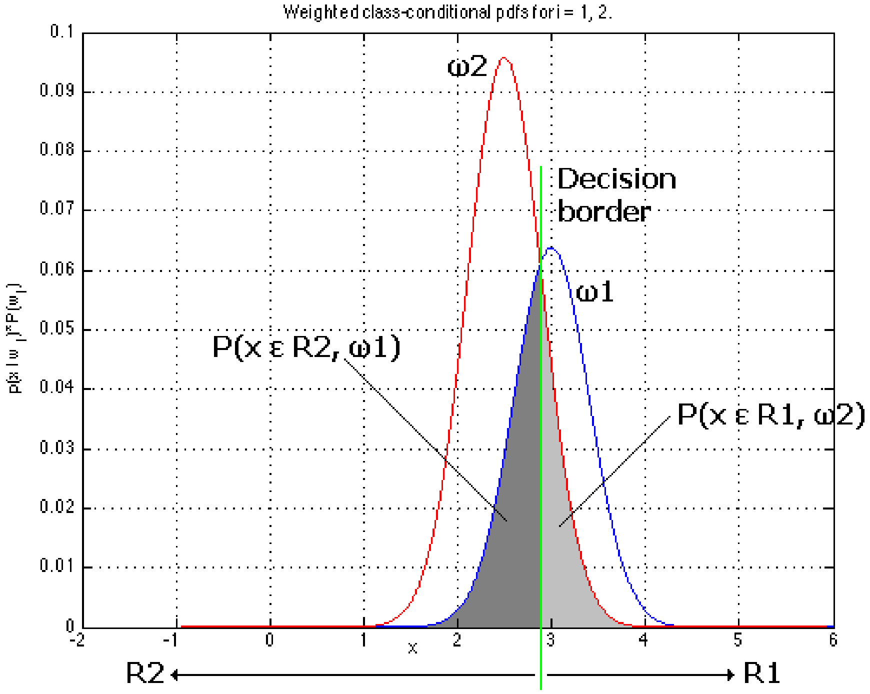


Figure 2.11: The figure shows an example of a computed decision border between two pdf's.

The area under $\omega_1$ and $\omega_2$ (R1 and R2) is the decision region of each class. The dark grayed area is the error probability, $P(error)$, that $x$ falls into R1 when $\omega_2$ is the true state of nature. The light grayed area is the error probability that $x$ falls into R2 when $\omega_1$ is the true state of nature.
The error probability is defined as

$$\begin{aligned} P(error) &= P(x \in R2,\, \omega_1) + P(x \in R1,\, \omega_2) \\ &= \int_{R2} p(x|\omega_1)P(\omega_1)\; dx + \int_{R1} p(x|\omega_2)P(\omega_2)\; dx \end{aligned} \tag{2.14}$$

### 2.5.2 Normal Density

If the observation $x$ is one-dimensional and normal distributed the class-conditional pdf is defined as,

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{2.15}$$

where $\mu$ is the expectation and $\sigma$ is the standard deviation.
If $x$ has more dimensions ($d$) the pdf is defines as,

$$p(\underline{x}) = \frac{1}{(2\pi)^{\frac{d}{2}} |\sum|^{\frac{1}{2}}} \cdot e^{-\frac{1}{2}(\underline{x}-\underline{\mu})^T \sum^{-1}(\underline{x}-\underline{\mu})} \tag{2.16}$$

where $\mu$ is the expectation and $\sum$ is the covariance matrix which determines the shape and orientation of the pdf.

### 2.5.3 Discriminant Functions for the Normal Density

Normally classifiers are represented by discriminant functions. The discriminant for the Bayes formula is defined as

$$\begin{aligned} g_j(\underline{x}) &= p(\underline{x}|\omega_j)P(\omega_j) \\ &= \ln\big(p(\underline{x}|\omega_j)\big) + \ln\big(P(\omega_j)\big) \end{aligned} \tag{2.17}$$

Equation 2.17 ensures minimum error classification.
For a two-category case the discriminant functions $g_1$ and $g_2$ are formulated like

$$g(\underline{x}) = g_1(\underline{x}) - g_2(\underline{x}) \tag{2.18}$$

If $g(\underline{x}) > 0$ $\omega_1$ is chosen, otherwise $\omega_2$.

Equation 2.17 can be evaluated for three different cases:

**Case 1:** $\Sigma_j = \sigma^2 I$

$$\begin{aligned} g_j(\underline{x}) &= \underbrace{\frac{1}{\sigma^2} - \underline{\mu}_j}_{\underline{w}_j^T} \underline{x} + \underbrace{\frac{-1}{2\sigma^2}\underline{\mu}_j^T \underline{\mu}_j + \ln\big(P(\omega_j)\big)}_{w_{j0}} \\ &= \underline{w}_j \underline{x} + w_{j0} \end{aligned} \tag{2.19}$$

**Case 2:** $\Sigma_j = \Sigma$

$$\begin{aligned} g_j(\underline{x}) &= \underbrace{\left(\Sigma^{-1}\underline{\mu}_j\right)^T}_{\underline{w}_j^T} \underline{x} + \underbrace{-\frac{1}{2}\underline{\mu}_j^T \Sigma^{-1} \underline{\mu}_j + \ln\left(P(\omega_j)\right)}_{w_{j0}} \\ &= \underline{w}_j^T \underline{x} + w_{j0} \end{aligned} \tag{2.20}$$

**Case 3:** $\Sigma_i$ arbitrary

$$\begin{aligned} g_j(\underline{x}) &= \underline{x}^T \underbrace{\left(-\frac{1}{2}\Sigma_j^{-1}\right)^T}_{W_j} \underline{x} + \underbrace{\left(\Sigma_j^{-1}\underline{\mu}_j\right)^T}_{\underline{w}_j} \underline{x} \\ &\quad + \underbrace{\frac{-1}{2}\underline{\mu}_j^T \Sigma_j^{-1} \underline{\mu}_j - \frac{1}{2}\ln\left(|\Sigma_j|\right) + \ln\left(P(\omega_j)\right)}_{w_{j0}} \\ &= \underline{x}^T W_j \underline{x} + \underline{w}_j \underline{x} + w_{j0} \end{aligned} \tag{2.21}$$

### 2.5.4 Maximum Likelihood Estimation (Multivariate Case)

Often when working with pattern recognition problems, the data used for training and testing has unknown prior probabilities and class-conditional densities ($p(\underline{x}|\omega_j)$). One way to solve this problem is to use the training data to estimate the prior probabilities and pdf, and use them like they were true values [3].
Maximum likelihood (ML) estimation assumes that the pdf is normal distributed with mean $\underline{\mu}$ and covariance matrix $\underline{\Sigma}$, and these are the parameters that are estimated. The parameters are looked at as quantities with fixed values. The value ($\hat{\underline{\theta}}$) that maximizes the probability of obtaining the samples, in the training set, that are actually observed is defined as the best estimate.

To perform a ML estimation on a set of class defined samples (training set $D$), the vector $\underline{\theta}$ has to be estimated. This parameter includes $\underline{\mu}$ and $\underline{\Sigma}$.

$$\underline{\theta} = \begin{pmatrix} \hat{\underline{\mu}} \\ \hat{\underline{\Sigma}} \end{pmatrix} = \begin{pmatrix} \underline{\theta}_1 \\ \underline{\theta}_2 \end{pmatrix} \tag{2.22}$$

The likelihood of $D$ being explained by the parameter vector $\underline{\theta}$ is then defined as,

$$p(D|\underline{\theta}) = \prod_{k=1}^{n} p(\underline{x}_k|\underline{\theta}) \tag{2.23}$$

Instead of working with the expression above, the log-likelihood is analyzed because it is easier.

$$l(\underline{\theta}) = \ln\big(p(D|\underline{\theta})\big) \tag{2.24}$$

The result is differentiated and set equal to zero by using the gradient.

$$\underline{\nabla}_{\theta} l = \begin{pmatrix} \frac{\partial}{\partial\theta_1} \\ \frac{\partial}{\partial\theta_2} \end{pmatrix} \tag{2.25}$$

The maximum likelihood estimation is then $\hat{\underline{\theta}} = \arg\max l(\underline{\theta})$ or $\underline{\nabla}_{\theta} = 0$.

**Example: Unknown $\mu$ and $\Sigma$:**
The log-likelihood

$$l(\underline{\theta}) = \sum_{k=1}^{n} \Big\{ -\frac{1}{2}\ln\big(2\pi\theta_2\big) - \frac{1}{2\theta_2}(\underline{x}_k - \theta_1)^2 \Big\} \tag{2.26}$$

The gradient

$$\begin{aligned} \nabla_{\theta} l &= \sum_{k=1}^{n} \nabla_{\theta} \ln\big(p(\underline{x}_k|\underline{\theta}\big) \\ &= \sum_{k=1}^{n} \nabla_{\theta}\Big\{ -\frac{1}{2}\ln\big(2\pi\theta_2\big) - \frac{1}{2\theta_2}(\underline{x}_k - \theta_1)^2 \Big\} \\ &= \sum_{k=1}^{n} \begin{pmatrix} \frac{\partial}{\partial\theta_1}(-\frac{1}{2}\ln\big(2\pi\theta_2\big) - \frac{1}{2\theta_2}(\underline{x}_k - \theta_1)^2) \\ \frac{\partial}{\partial\theta_2}(-\frac{1}{2}\ln\big(2\pi\theta_2\big) - \frac{1}{2\theta_2}(\underline{x}_k - \theta_1)^2) \end{pmatrix} \\ &= \begin{pmatrix} \sum_{k=1}^{n} \frac{1}{\theta_2}(\underline{x}_k - \theta_1) \\ \sum_{k=1}^{n} -\frac{1}{2\theta_2} + \frac{(\underline{x}_k - \theta_1)^2}{2\theta_2^2} \end{pmatrix} \end{aligned} \tag{2.27}$$

By letting equation 2.27 equal zero, the parameter $\hat{\underline{\theta}}$ can be calculated

$$\hat{\underline{\theta}} = \begin{pmatrix} \hat{\underline{\mu}} \\ \hat{\underline{\Sigma}} \end{pmatrix} = \begin{pmatrix} \frac{1}{n}\sum_{k=1}^{n} \underline{x}_k \\ \frac{1}{n}\sum_{k=1}^{n} (\underline{x}_k - \hat{\underline{\mu}})(\underline{x}_k - \hat{\underline{\mu}})^T \end{pmatrix} \tag{2.28}$$

### 2.5.5 Nonparametric Techniques

In nonparametric procedures the pdf is estimated from sample patterns (training sets). The pdf can have a arbitrary distribution. Practical problems often don't fit a parametric model, like ML, because the problem is multimodal and ML is unimodal [3].

In nonparametric estimation the goal is to estimate the density in a local region.
Imagine a region $R$. The probability of $\underline{x}$ being in the region is,

$$P = \int_R p(\underline{x}')\ d\underline{x}' \approx p(\underline{x})V \tag{2.29}$$

where $V$ is the volume of $R$. If there is n samples $(\underline{x}_1,\ \ldots,\ \underline{x}_n)$ drawn from $p(\underline{x})$ the probability of $k$ samples falling in $R$ is,

$$P_k = \binom{n}{k} P^k (1-P)^{n-k} \tag{2.30}$$

$P_k$ is binomial distributed and the expectation of $k$ is $nP$. In [3] the authors shows that $k/n$ is a good estimate for the probability $P$ and this gives the following estimate for $p(\underline{x})$,

$$p(\underline{x}) = \frac{k/n}{V} \tag{2.31}$$

If $n$ is a fixed number and $V$ approaches zero, the region will become small and enclose no samples. Then the estimate would approximately become zero. Instead consider estimating the density of $\underline{x}$ for several regions, $R_1,\ R_2,\ \ldots R_n$ and let the number of samples $n$ be unlimited ($R_1$ contains one sample, $R_2$ contains two samples and so on). $p_n(\underline{x})$ is then the nth estimate of $p_n(\underline{x})$ and is defined as,

$$p_n(\underline{x}) = \frac{k_n/n}{V_n} \tag{2.32}$$

where $k_n$ is the number of samples in $R_n$ and $V_n$ is the volume of $R_n$.
To converge $p_n(\underline{x})$ to $p(\underline{x})$, three criterias is required

$$\lim_{n\to\infty} V_n = 0 \tag{2.33}$$

$$\lim_{n\to\infty} k_n = 0 \tag{2.34}$$

$$\lim_{n\to\infty} k/n = 0 \tag{2.35}$$

The Parzen Window technique and $k_n$-Nearest-Neighbor Technique (kNN) satisfies these criterias.

#### 2.5.5.1 Parzen Window

Parzen Window assumes that $R_n$ is a hypercube and that an edge on this hypercube has the length $h_n$. The volume $V$ is then $h_n^d$ (d - dimension). The number of samples falling in $R_n$ is defined as the window function, $\varphi(\underline{u})$,

$$\varphi(\underline{u}) = \begin{cases} 1 & \text{if } |u_j| \leq 0.5 \; j = 1, ..., d \\ 0 & \text{otherwise} \end{cases} \tag{2.36}$$

where $\varphi(\underline{u})$ is a unit hypercubes centered at the origin because $\underline{u}$ is normalized.

$$\underline{u} = \frac{\underline{x} - \underline{x}_i}{h_n} \tag{2.37}$$

The number of samples in the hypercube ($R_n$) is defined as,

$$k_n = \sum_{i=1}^{n} \varphi\Big(\frac{\underline{x} - \underline{x}_i}{h_n}\Big) \tag{2.38}$$

With the help of equation 2.38 the expression for the estimate $p_n(\underline{x})$ can be written as,

$$p_n(\underline{x}) = \frac{1}{n} \sum_{i=1}^{n} \varphi\Big(\frac{\underline{x} - \underline{x}_i}{h_n}\Big) \frac{1}{V_n} \tag{2.39}$$

These estimates is an average of functions of $\underline{x}$ and the sample $\underline{x}_i$. Each $\underline{x}_i$ contributes to the estimate according to it's distance from $\underline{x}$.

A hypercube window doesn't produce a legitimate density function. To avoid this, window functions are defined as density functions,

$$\delta_n(\underline{x}) = \frac{1}{V_n} \varphi\Big(\frac{\underline{x}}{h_n}\Big) \tag{2.40}$$

where $V_n = h_n^d$. $h_n$ is called the window width. Now the estimate is espressed,

$$p_n(\underline{x}) = \frac{1}{n} \sum_{i=1}^{n} \delta_n(\underline{x} - \underline{x}_i) \tag{2.41}$$

The window width is defined as,

$$h_n = \frac{h1}{\sqrt{n}} \tag{2.42}$$

where $h1$ is the variable the varies the window width and n is the number of samples in $\underline{x}$.

The window width affects the amplitude and and width of $\delta_n(\underline{x})$. Large $h_n$ produces low amplitude and high width, and the distance between $\underline{x}$ and $\underline{x}_i$ must be large to make $\delta_n(\underline{x} - \underline{x}_i)$ change a lot. If $h_n$ is small, the effect will be a large peak in $\delta_n(\underline{x} - \underline{x}_i)$ near $\underline{x} = \underline{x}_i$ and $\delta_n(\underline{x} - \underline{x}_i)$ is narrow. The figure below shows the effect of large and small window width.

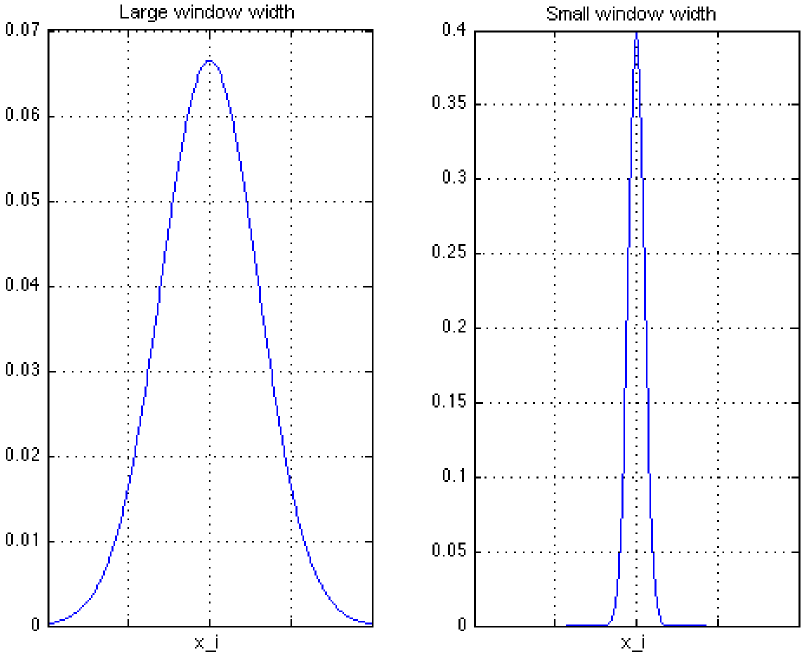


Figure 2.12: The left plot shows the effect of large $h_n$ and the right plot shows the effect of small $h_n$. Both examples are for 1 dimensional data. The purpose is only to give an image of the effect of using large and small window width values.

The window function could also be Gaussian and this function is defined as,

$$\varphi(\underline{u}) = \frac{1}{(2\pi)^{\frac{d}{2}}} e^{-\frac{(\underline{x}-\underline{x}_i)^2}{2h_n^2}} \tag{2.43}$$

The $p_n(\underline{x})$ estimate then becomes,

$$\begin{aligned} p_n(\underline{x}) &= \frac{1}{n}\sum_{i=1}^{n}\frac{1}{V_n}\varphi(\underline{u}) \\ &= \frac{1}{n}\sum_{i=1}^{n}\frac{1}{(2\pi)^{\frac{d}{2}}h_n} e^{-\frac{(\underline{x}-\underline{x}_i)^2}{2h_n^2}} \end{aligned} \tag{2.44}$$

#### 2.5.5.2 $\mathbf{k_n}$-Nearest-Neighbor (kNN)

Imagine a set of $n$ samples (training data) spread out in a plane. To estimate $p(\underline{x})$ a region ($R$) is centered around $\underline{x}$ (feature vector) and $R$ grows until $k_n$ samples are captured [3]. All captured samples are the $\mathrm{k_n}$-nearest-neighbors to $\underline{x}$. The estimate is defines as,

$$p_n(\underline{x}) = \frac{k_n/n}{V_n} \tag{2.45}$$

where $k_n$ is fixed and determines how many captured neighbors $R$ shall contain until it stops growing. $n$ is the number of samples and $V_n$ is the distance from $\underline{x}$ to the $k_n$-th nearest neighbor.

The region $R$ is defined as a hypersphere and $V_n$ decides it's area. $V_n$ is defined as,

$$V_n = \pi||\underline{x} - \underline{x}_i||^2 \tag{2.46}$$

where $\underline{x}_i$ is the $k_n$-nearest neighbor.

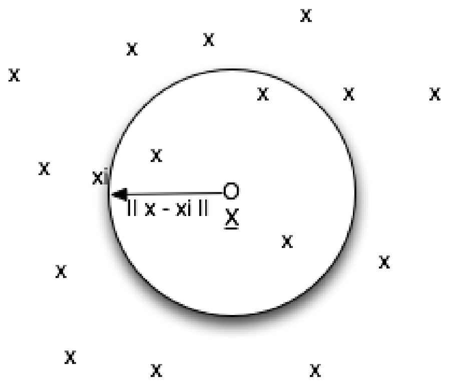


Figure 2.13: The region radius is marked with an arrow. The figure shows the area containing the $k_n$-nearest-neighbors to $\underline{x}$.

The a posteriori probability to the kNN can be expressed as,

$$P_n(\omega_i|\underline{x}) = \frac{p_n(\underline{x}|\omega_i)}{\sum_{j=1}^{c} p_n(\underline{x}|\omega_j)} = \frac{k_i}{k} \tag{2.47}$$

where $p_n(\underline{x}|\omega_i)$ is the probability estimate for $\underline{x}$ given that the state of nature is $\omega_i$, $k_i$ is the number of samples belonging to $\omega_i$ and $k$ is the total number of captured samples in the kNN. $p_n(\underline{x}|\omega_i)$ is defined as,

$$p_n(\underline{x}|\omega_i) = \frac{k_i/n}{V} \tag{2.48}$$

**Example: 2 class problem**
A feature vector $\underline{x}$ is classified with kNN to a set of labeled samples that belongs to $\omega_1$ and $\omega_2$. $k_n$ equals 3. The total number of samples captured by the kNN is 7[1] (two or more samples can have the same distance to $\underline{x}$). Of these 7 samples, 3 belongs to $\omega_1$ and 4 belongs to $\omega_2$. The feature vector $\underline{x}$ is therefore classified to $\omega_2$ since $\frac{4}{7} > \frac{3}{7}$ $(P_n(\omega_2|\underline{x}) > P_n(\omega_1|\underline{x}))$.

[1] $k = 7$.

# Chapter 3

# A System Overview

This Chapter presents a brief overview of the system modules. For the physical system, the main parts are presented in section 3.1. Section 3.2 gives a brief presentation of MATLAB.

Figure 3.1 shows a block schematic structure of the system. The system is divided into a physical module and a program module (MATLAB). The physical module consists of a conveyor belt, a sorter arm, the NXT and a camera. The MATLAB module is made up of three main functions; *Moving Object Detection (MOD)*, *Object Recognition & Feature Extraction (OOI)* and *Feature Classification.*

The MOD function is used to detect moving objects on the conveyor belt that passes under the camera. To detect an object, MOD uses two main techniques; *Background Subtraction* and *Background Modeling.* MOD is introduced in Chapter 4.

The OOI function extracts the object of interest from the image background and finds features that describe the object. OOI is introduced in Chapter 5.

The feature vector is classified by a classifier in the Feature Classification function. The result from this classification decides if the object belongs to class 1 or class 2. This function is presented in Chapter 6.

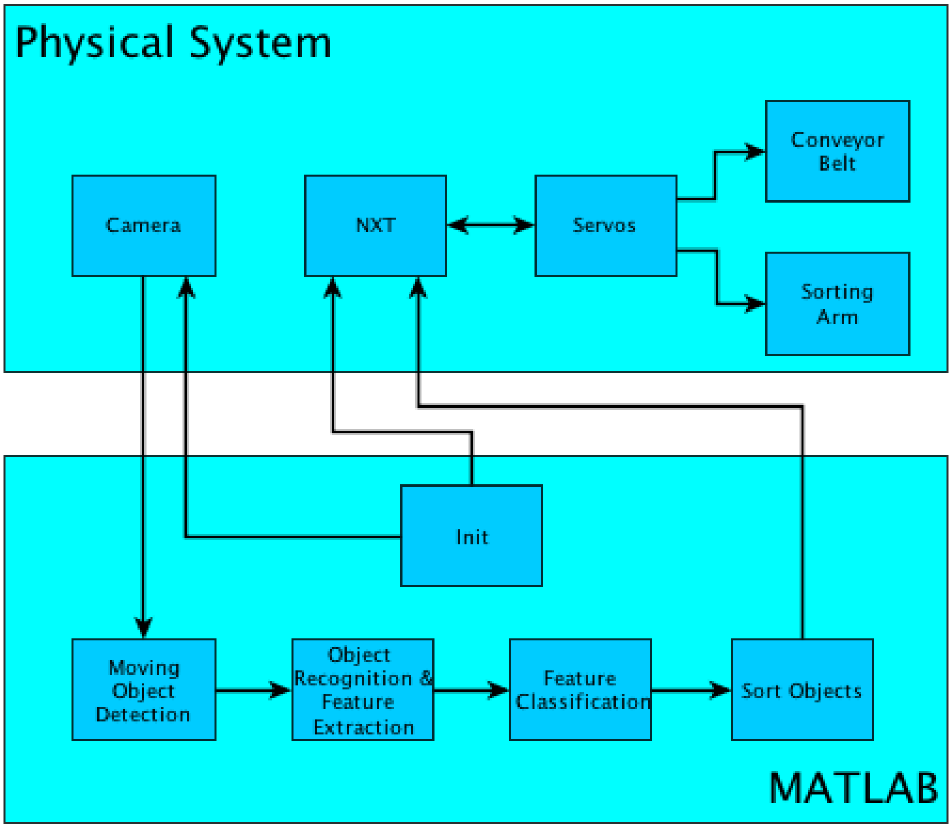


Figure 3.1: The figure shows the main components of the Conveyor belt system. The system is divided into to two main modules; physical and MATLAB. The physical module is the mechanical parts of the system and the MATLAB module is program that initiates and runs the system.

## 3.1 Physical System

This section presents some of the main mechanical modules on the object sorter system.

### 3.1.1 NXT Intelligent Brick 2.0 (NXT)

NXT is the brain of the Lego Mindstorms robots. The device has four inputs were sensors can be connected and three outputs which controls the servos. It's possible to connect a computer to the NXT by either USB or bluetooth. The figure below shows the NXT device.

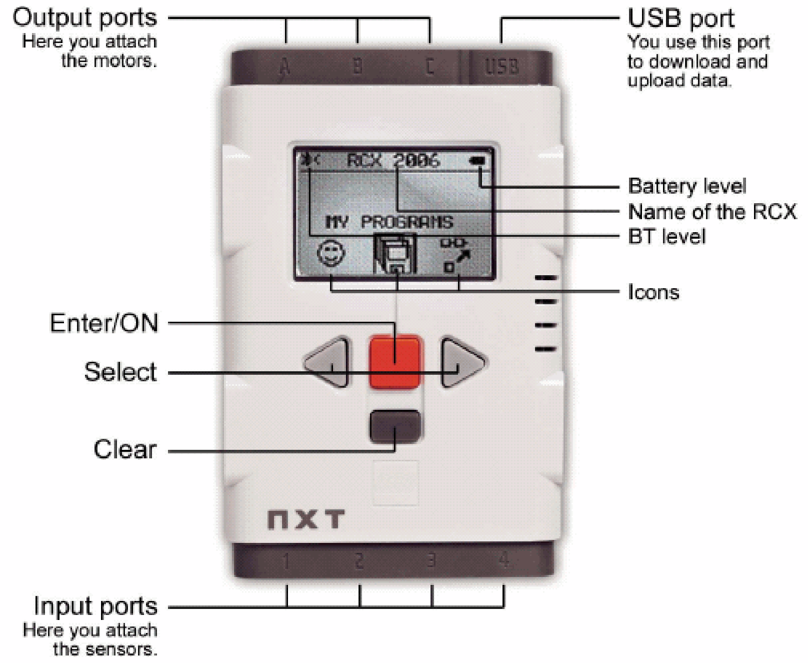


Figure 3.2: NXT Intelligent Brick. © 2012 LEGO. All rights reserved.

The NXT controls two servos on the sorter system. One servo controls the conveyor belt and the other servo controls the sorter arm. Each servo has an internal rotation-sensor which makes it possible to turn the servos precisely. This sensor reads the rotation of the servo with an accuracy of ± 1 degree. A full rotations of a servo is 360°.

### 3.1.2 The Conveyor Belt

The conveyor belt consists of three separate modules and is connected as a set of steps. This shown in figure 3.3. The conveyor belts task is to transport objects along the length of the belt.

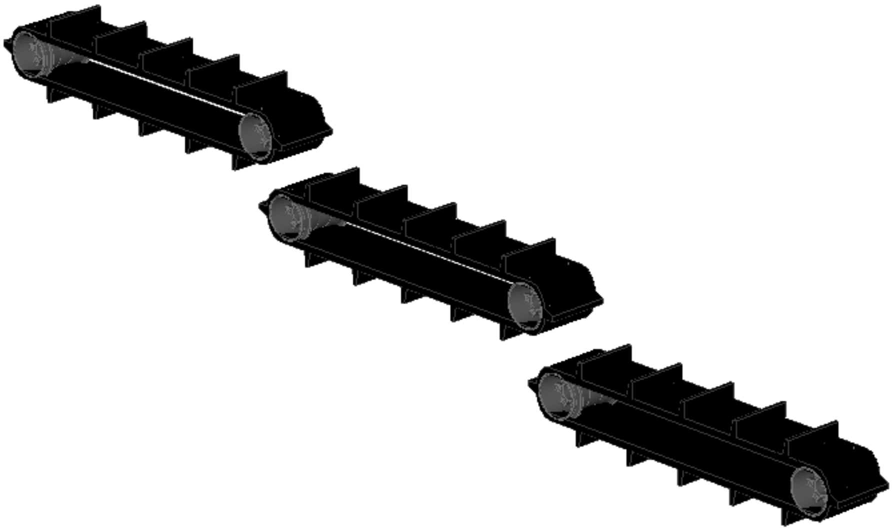


Figure 3.3: Design of the conveyor belt. The conveyor belt modules are placed in a tread pattern to prevent contact between each module.

There are two reasons for this implementation. First, if all three modules are placed at the same height, the tracks on each conveyor module will be in contact with each other when the conveyor belt is running and this creates unstable movement. Secondly if the gap between them increases so that the tracks isn't in contact with one another, objects can fall through the gap and possibly get stuck, making the conveyor belt crash. By creating a height difference between the modules, no tracks is in contact and the gap between each module is reduced, giving no room for an object to get through.
A NXT motor/servo which operates all modules and run the conveyor belt is connected to the first module. The connection between each module is a set of mechanical gears that is placed between the first and second module, and between the second and third module. The mechanism is shown in figure 3.4.

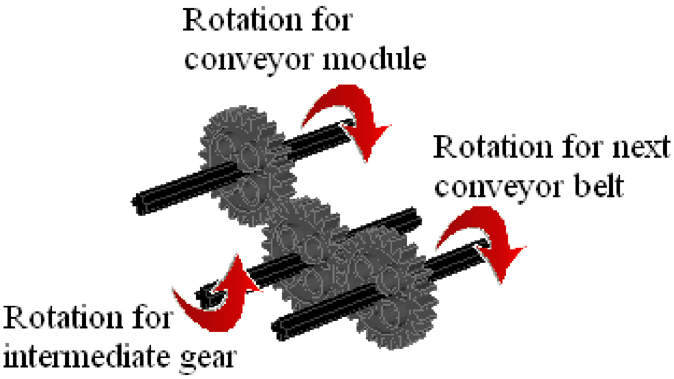
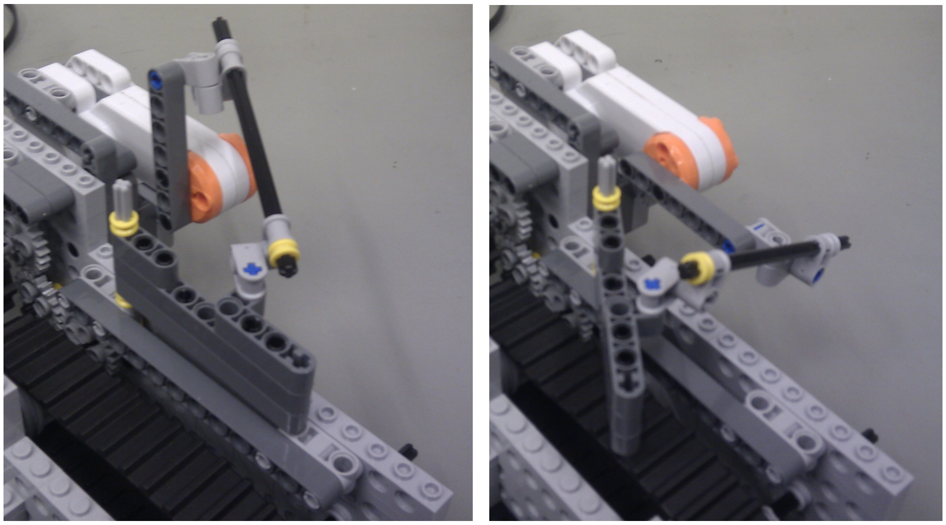


Figure 3.4: Mechanical gear module. The intermediate gear rotates in the opposite direction of the first gear. This rotation affects the next gear that is connected to another conveyor belt module and turns it in the same direction as the previous module.

### 3.1.3 Sorter Arm

The sorter arms task is to push objects of the conveyor belt and into a sorting tray. When an object is represented and recognized the sorter arm may be asked to push the object of the conveyor belt if the developed program's directives for this object is declared to execute the sorter arm's function.
Figure 3.5 shows the arm in stationary and working position.



(a) Stationary position (b) Working position

Figure 3.5: Sorting arm in stationary and working position.

Figure 3.6 shows only the sorter arm.

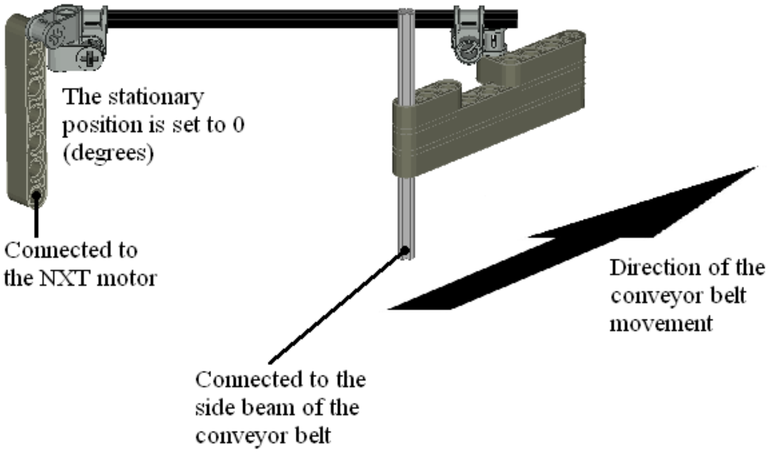

The stationary position is set to 0 (degrees)
Connected to the NXT motor
Connected to the side beam of the conveyor belt
Direction of the conveyor belt movement

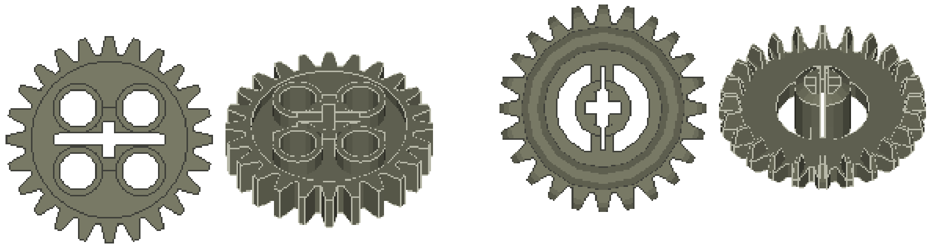
Figure 3.6: An illustration of the mechanical sorter arm in stationary position.

The NXT servo, which controls the sorter arm, is connected to the vertical beam shown in the figure 3.6. When the arm is in the stationary position the motor is in position 0° and when the arm is in the working position the motor rotates 90° (and stops) as shown in figure 3.7b.

### 3.1.4 Physical Objects

One of the problems was to find objects that was similar, but at the same time different. The idea was to have minimum three different objects that could be sorted. But because of limited space on the conveyor belt, the selection of objects became limited as well. In the end, two objects was chosen to be sorted. These two objects are very similar in size and shape. This makes the classification problem a lot more difficult, but that was the intention.

The objects are presented in the figure below.



(a) Technic Gear 24T

(b) Technic Gear 24TC

Figure 3.7: The figure shows the two objects that the sorter system shall distinguish between. Figure (a) shows a technic gear that consist of 24 teeth. Figure (b) shows a technic gear that also consist of 24 teeth, but the gear is crowned.

For the classification problem, the objects have to be defined into two classes. The Technic Gear 24T belongs to class 1 and the Technic Gear 24TC belongs to class 2.

### 3.1.5 Webcam

The webcam implemented on the object sorter system is a *Creative Live! Cam Chat HD*. The video resolution is 720p and it can capture images up to 5.7 megapixels.

Figure 3.8: Creative Live! Cam Chat HD. © 2012 Creative Technology Ltd. All rights reserved.

**Specifications:**

| | |
|---|---|
| Image Sensor | HD 720p (1280 × 720) |
| Video resolution | 720p HD |
| Image resolution | 5.7 megapixels |
| Frame rate (max) | 30fps (at HD 720p) |
| Connectivity | USB 2.0 Hi-Speed |

The main task of the webcam is to capture snapshots (frames) of the moving conveyor belt that MATLAB analyzes for further processing. MATLAB controls the webcam and decides when it should grab a frame. When a object is in the webcams field of view (FOV), MATLAB detects the object. This method is called *Moving Object Detection* and is presented in chapter 4.

The webcam is initialized in MATLAB by the *Image Acquisition* toolbox and enables camera devices to communicate directly with MATLAB and return frames automatically or manually to the MATLAB environment. The toolbox also enables users to configure camera settings. This toolbox is presented in section 3.2.3.

#### 3.1.5.1 Configuration of the Webcam

The webcam is configured to return RGB images that only contains the ROI. The normal size of an frame that is captured by the webcam is 1280 × 720, but since the configuration is set to only return the ROI, the image size is 370 × 705.
These configurations are enabled after the webcam is acquired by the Image Acquisition toolbox and implemented by the MATLAB function `set`. The webcam is initialized in `global_var.m` and at line 4 and 5 in the MATLAB implementation in appendix E.9, these configurations are enabled.

## 3.2 Development Tools

This section gives a short presentation of the development tools used in this report. Mainly MATLAB is used, but a important toolbox is also represented. This toolbox is called RWTH and is used to make MATLAB communicate with the LEGO Mindstorms NXT.

### 3.2.1 MATLAB

MATLAB is an abbreviation for MAtrix LABoratory and is developed by MathWorks. MATLAB is a numerical computing environment that integrates computation, visualization and programming. It's main uses are algorithm development, modeling, simulation, analysis and data acquisition [5]. An addition to MATLAB is a so called application-specific solution called *toolboxes*. These toolboxes is a collection of MATLAB functions that is used to solving different problems. A problem could perhaps be Image Processing or Signal Processing related. Each toolbox give the users specific functions that can be used to solve different problems concerning the subject. It also includes documentation and examples of use.
In this report the Image Acquisition and Image Processing toolboxes are mainly used. There is also a third toolbox, called RWTH, that is used to make MATLAB communicate with the LEGO Mindstorms NXT. A short summary of the different toolboxes will be given in the next three sections.

### 3.2.2 Image Acquisition Toolbox

This toolbox helps the user to acquire video and images from cameras directly into the MATLAB environment. It supports a wide range of camera vendors and it can both be used by low cost web cameras and high-end devices [11].
With the toolbox comes documentation that can be accessed on the MathWorks homepage or in the **Help** function in MATLAB. This documentation includes sections like getting started with the toolbox, a users guide, examples and more. There are also a list of MATLAB functions that are used for initializing and running the acquisition devices. More information can be found in [11]. The table below shows some common MATLAB Image Acquisition functions.

| | |
|---|---|
| `closepreview` | Closes the the live video window |
| `getsnapshot` | Returns a single captured frame immediately |
| `imaqfind` | Finds video objects initialized in MATLAB |
| `imaqhwinfo` | Returns information on available hardware devices |
| `preview` | Previews live video data in a window |
| `videoinput` | Creates a video input object |

Table 3.1: Common functions in the Image Acquisition toolbox.

### 3.2.3 Image Processing Toolbox

The Image Processing toolbox offers the user a set of functions, algorithms and graphical tools for image processing, analyzing and visualization. Some of the functions / algorithms can convert images into different formats, like RGB to grayscale, adjust contrast, do image registration and segmentation, do morphological operations and more [11].
Like the Image Acquisition toolbox, this toolbox also offers the same kind of documentation and can be found like mentioned in the previous section. Table 3.2 shows some common MATLAB Image Processing functions.

| | |
|---|---|
| `bwboundaries` | Finds the coordinates of the object perimeter |
| `bwlabel` | Labels connected components (objects) |
| `bwperim` | Finds the perimeter of an object |
| `bwselect` | Finds and selects labelled objects |
| `greythresh` | Computes global image threshold using Otsu's method |
| `imadjust` | Adjusts contrast in image |
| `imclearborder` | Clears objects that is placed on the image border |
| `imclose` | Morphological closing |
| `imdilate` | Morphological dilation |
| `imerode` | Morphological erotion |
| `imfill` | Fills holes and image regions |
| `imhist` | Shows histogram og image data |
| `imopen` | Morphological opening |
| `imshow` | Displays image in MATLAB |
| `im2bw` | Converts grayscale image to binary image |
| `regionprops` | Returns data on detected objects in image |
| `rgb2gray` | Converts RGB image to grayscale image |
| `strel` | Create a morphological structural element |

Table 3.2: Common functions in the Image Process toolbox.

### 3.2.4 RWTH Mindstorms NXT Toolbox

The RWTH Toolbox is developed at the RWTH Aachen University in Germany as a student project called *MATLAB meets LEGO Mindstorms*. The background of the project was only for educational purposes for the students at the Institute of Electrical and Computer Engineering at the mentioned university [14].

The RWTH toolbox is a free open source product that makes MATLAB available to interact directly with the NXT over a bluetooth or USB connection. MATLAB can compute time- and machine demanding tasks that the NXT can't handle because of it's limited CPU- and memory capacity. [14] argues that the biggest advantage of this toolbox is to combine MATLABs complex mathematical computation with robot applications.
All functions in the toolbox is divided up into four different categories which are:

1. **Low level functions:** Convert parameters into bytes that is determined by the LEGO direct command documentation.
2. **Direct NXT Commands**
3. **High Level Functions:** Controls the NXT motors, sensors and bluetooth.
4. **High Level Regulation / Utilities:**
   Provides accurate motor regulation.

Table 3.4 show some commonly used functions and commands from the RWTH toolbox.

| | |
|---|---|
| **Low level functions** | |
| `MOTOR_X` | Constant for motor X.<br>X can be A, B or C (outputs on NXT). |
| `SENSOR_X` | Constant for sensor X.<br>X can be 1, 2, 3 or 4 (inputs on NXT). |
| **Direct NXT Commands** | |
| `NXT_GetBatteryLevel` | Return current battery level (mV). |
| `COM_SetDefaultNXT(HNXT)` | Sets the handle `HNXT` to a global NXT handle. This handle are used by all NXT functions (default). |
| **High Level Functions** | |
| `COM_OpenNXT` | Opens a USB or bluetooth connection to the NXT and returns a handle (`HNXT`) to MATLAB. |
| `COM_CloseNXT` | Closes and deletes a handle (`HNXT`) in MATLAB. |
| **High Level Regulation** | |
| `NXTMotor(P)` | Makes an NXT motor object with motor port `P` (`P` can be `MOTOR_X`). Ex: `mA = NXTMotor(MOTOR_A)`. |
| `ReadFromNXT` | Reads the current state from a motor via NXT. Ex: `mA.ReadFromNXT()`. The function returns among other things the power and position. |
| `Stop` | Stops a specified motor. Ex: `mA.Stop()`. |
| `WaitFor` | Waits for a motor to stop. Ex: `mA.WaitFor()`. |
| `ResetPosition` | Resets the position of a motor. Ex: `mA.ResetPosition()`. |
| `COM_MakeBTConfigFile` | Creates a bluetooth configuration file. |

Table 3.4: Common functions / commands in the RWTH toolbox.

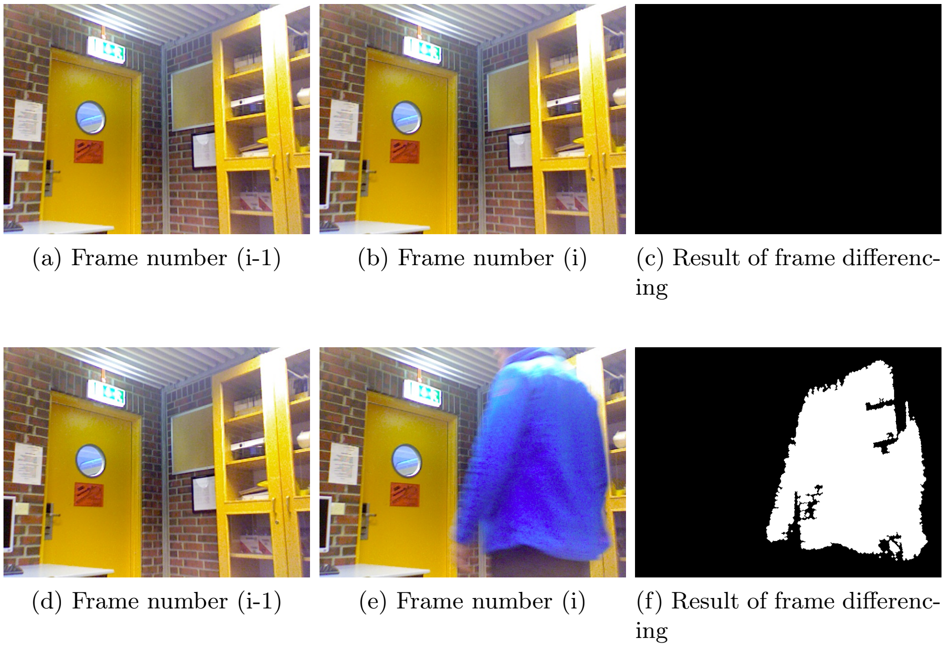
When creating a motor object with the `NXTMotor` function, the user can initialize a set of properties that controls the motor. The different properties besides motor port (`P`) are power, speed regulation (sync. two or more motors), tacho limit (angle of degrees the motor will turn), action at tacho limit and smooth start. More information about these functions and others can be found in [13].

# Chapter 4

# Moving Object Detection

To detect objects on a conveyor belt that is moving is a difficult task. In a static environment the background doesn't move particularly, except for some movements of certain objects, like trees, because of the wind or that it can rain or snow. These examples are based on a surveillance camera that is positioned outside. A camera that is positioned inside wouldn't have these problems if the cameras field of view (FOV) didn't contain a view to the outside of the building. Then the background would be completely static if no one was moving around in the cameras FOV or that the illumination suddenly changes.

The video recording from this camera could use a basic technique called background subtraction (BS) [8], also known as frame differencing or temporal difference [4], to compare the i-th frame (the frame captured now) with the (i-1)-th frame (the frame captured before) or with a reference frame to check for movements in the cameras FOV. If there are no movement, the two frames are identical and no movement is detected. But if the i-th frame contains movement (a person walking by for an example) and the frame before contained the background without the person, the frame differencing would detect movement. An example is shown in the figure 4.1.



(a) Frame number (i-1)

(b) Frame number (i)

(c) Result of frame differencing

(d) Frame number (i-1)

(e) Frame number (i)

(f) Result of frame differencing

Figure 4.1: Result of BS in a given environment for two different situations.

The result of frame differencing between the images shown in figure 4.1a and 4.1b detects no movement as shown in 4.1c, since the two frames are identical. While the frames in figure 4.1d and 4.1e shows a change in scene, the frame differencing detects movement and the result is shown in figure 4.1f.

Frame differencing subtracts two grayscale images pixel-by-pixel from another and the result is a image (of the same size) that contains the difference. For each pixel subtraction the result is compared with a threshold to determine which pixels that is a foreground pixel (detected object) or is a background pixel (static environment) [8]. A mathematical expression for frame differencing is as follows,

$$F_{fd}(m,n) = \begin{cases} 1 & \text{if } |F_i(m,n) - F_{i-1}(m,n)| > T \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

where $F_{fd}(m,n)$ is the result of frame differencing, $F_i(m,n)$ is the current frame, $F_{i-1}(m,n)$ is the previous frame (or background reference) and $T$ is the threshold. This expression is based on formula (1) in [8].

One way of implementing the BS technique in MATLAB can be done by the following simplified pseudocode:

**Algorithm 1** Moving Object Detection using BS

```
$i$ & $DET = 1$
**while** $DET == 1$ **do**
  get RGB frame from camera
  convert frame to grayscale
  save frame i in F{i}(m, n)
  **if** $i == 1$ **then**
    $i \leftarrow 1$
  **else**
    $F_{fd}(m,n) = F\{i\}(m,n) - F\{i-1\}(m,n)$
    convert $F_{fd}(m,n)$ to binary image
    **if** $F_{fd}(m,n) > T$ **then**
      **return** $F\{i\}(m,n)$
      $DET = 0$
    **else**
      $i \leftarrow 1$
    **end if**
  **end if**
**end while**
```

The variable $DET$ tells the **while**-loop that an object is either found from frame differencing or not. When it is 1 no object is found. The variable $i$ is used to increment *F{i}* and save each frame separately.

The BS technique operates well for moving object detection (MOD) when the background is static, but what happens when the background is dynamic, like for example with the conveyor belt that always is in motion?

The basic structure of the conveyor belt / camera setup is that the conveyor belt moves horizontally while the camera's position is perpendicular and above the conveyor belt with the camera lens pointing down on the belt, as shown in figure 4.2.

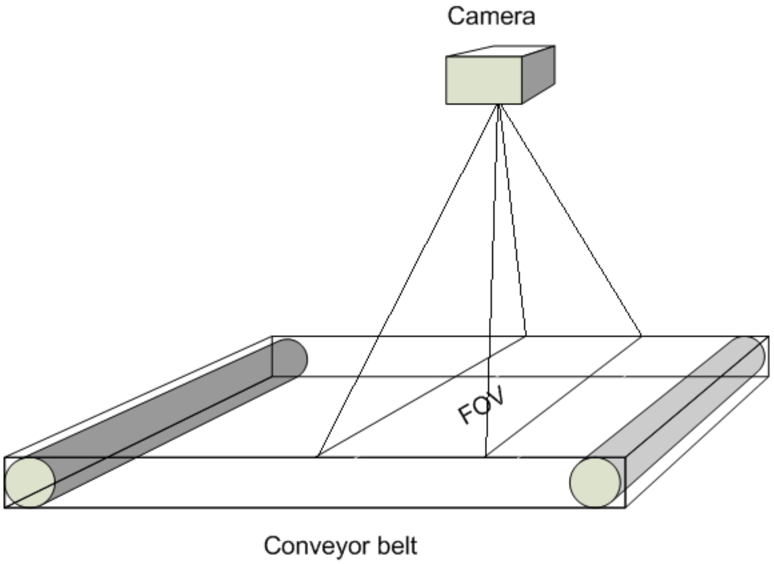
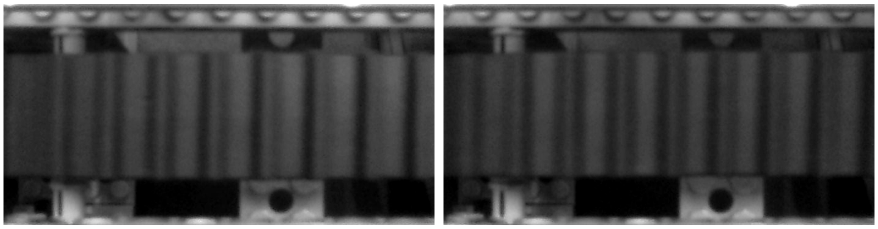


Figure 4.2: A simple sketch of the camera and conveyor belt layout. Note that this sketch isn't similar to the model in reality, in terms of FOV and dimensions. The conveyor belt is also not smooth. The surface of the conveyor belt consists of tags that go across the belt. Between each tag there's a gap.

Since the material that the surface of the conveyor belt is made from isn't non reflective, it may reflect light that hits the conveyor belt depending on the current illumination, while moving. When one frame (*i*) is captured by the camera without any object in the FOV, the reflection from the belt caused by illumination from the lights in the room may be present in this frame. In the next frame (*i+1*) the same thing happens, but the conveyor belt isn't positioned identical as it was in frame *i* and the reflection will be different because of the non reflective surface and the tags on the belt. When calculating the frame difference from the two mentioned frames, the system will detect movement and produces a so called *false positive* object detection. This makes the BS technique unsuitable for this purpose. An example of this is shown in figure 4.3 where the (i-1)-th and i-th frame is shown and figure 4.4 shows the result of the false positive object detection on a binary image.



(a) Frame number i-1

(b) Frame number i

Figure 4.3: The dynamic background of the conveyor belt for two frames captured at different times.

The difference in the two frames in figure 4.3 is the positioning of the tags on the conveyor belt.

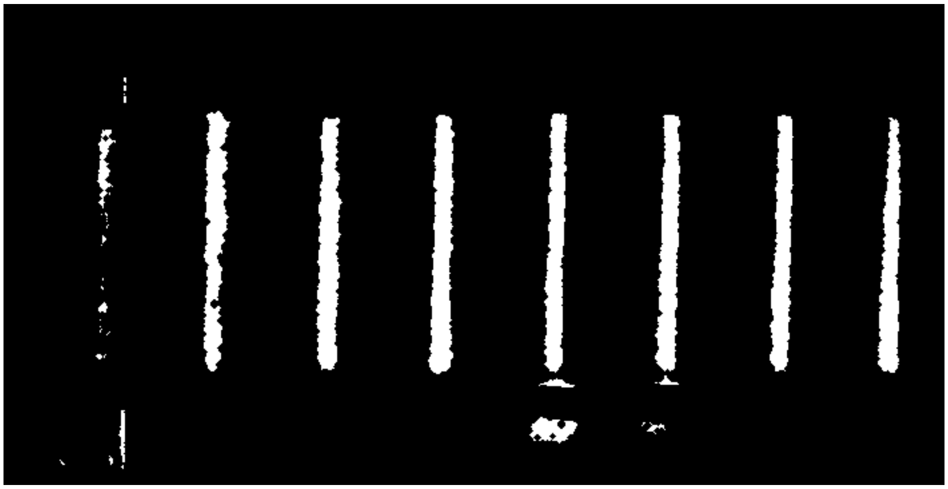


Figure 4.4: False positive object detection on the conveyor belt due to illumination. The figure shows the difference between the (i-1)-th and i-th frame as an binary image.

In this situation the conveyer belt / camera setup was position directly below a light fixture in the ceiling. When the unit was moved and placed away from the fixture, the illumination from the fixture wasn't directly above. The reflection from the belt was less than in the situation before. In terms of using the same pseudocode in algorithm 1 at the current placement produced the same result as before.

## 4.1 Enhancing the Current Pseudocode

The current pseudocode for the basic BS technique is always detecting false positives because the conveyor belt is reflecting illumination, from light sources that are present in the environment, differently in every frame the camera captures. The conveyor belt movement and light reflection creates a dynamic background. One idea for suppressing the dynamic background and ignore false positives came when studying the different thresholds that the MATLAB function `graythresh` computed for some captured frames with no object and frames with an object present, in the dynamic background. This function uses Otsu's method to determine the threshold [5].

To ensure for sudden light reflections that could affect the threshold in one frame and compute a high threshold value, the average threshold from the last ten captured frames are computed. Table 4.1 shows twenty computed average thresholds without an object in the cameras FOV.

| **Frame** | **1** | **2** | **3** | **4** | **5** |
|---|---|---|---|---|---|
| Threshold | 0.0533 | 0.0494 | 0.0478 | 0.0537 | 0.0529 |
| **Frame** | **6** | **7** | **8** | **9** | **10** |
| Threshold | 0.0478 | 0.0486 | 0.0514 | 0.0482 | 0.0447 |
| **Frame** | **11** | **12** | **13** | **14** | **15** |
| Threshold | 0.0463 | 0.0443 | 0.0416 | 0.0388 | 0.0373 |
| **Frame** | **16** | **17** | **18** | **19** | **20** |
| Threshold | 0.0369 | 0.0337 | 0.0349 | 0.0329 | 0.0337 |

Table 4.1: Computed average thresholds for the dynamic background using the MATLAB function `graythresh`.

The average thresholds that are computed from the dynamic background (without any object in the scene) are small, but they may differ from these results and be larger values depending on the light conditions.

The next table shows the average thresholds right before an object entered in the cameras FOV and when it is in the FOV. Table 4.2 shows the result of the computed average thresholds.

| **Frame** | **1** | **2** | **3** | **4** | **5** |
|---|---|---|---|---|---|
| Threshold | 0.0271 | 0.0267 | 0.0353 | 0.0427 | 0.0518 |
| **Frame** | **6** | **7** | **8** | **9** | **10** |
| Threshold | 0.0624 | 0.0714 | 0.0804 | 0.0902 | 0.1012 |
| **Frame** | **11** | **12** | **13** | **14** | **15** |
| Threshold | 0.1118 | 0.1129 | 0.1133 | 0.1137 | 0.1145 |
| **Frame** | **16** | **17** | **18** | **19** | **20** |
| Threshold | 0.1157 | 0.1173 | 0.1192 | 0.1212 | 0.1227 |

Table 4.2: Computed average thresholds for the dynamic background when object was in FOV, using the MATLAB function `graythresh`.

In the two first frames the object isn't inside the cameras FOV, but from the third frame the object enters it and the average thresholds starts to rise. At frame 7 the whole object is inside the FOV.

From this information the pseudocode was altered so that the code would only recognize an object if the average threshold computed by the `graythresh`-function exceeded a user-defined threshold (UDT). The UDT value couldn't be to small because then the code would detect reflection movement, and it also couldn't be to high since then it wouldn't detect nothing at all. Some experiments where done that concluded that UDT was set to 0.09. The value is higher than the average threshold values computed when a object wasn't in the FOV. Sudden light reflections may affect the average threshold to get even higher values than shown in table 4.1 and by letting UDT be 0.09, it can handle these situations. The algorithm ran for a long period with no objects passing the FOV and it didn't detect any false positives with this UDT value.

### 4.1.1 Ensuring the Whole Object Within the FOV

The frame that is returned in the end of the pseudocode must contain the whole object within the FOV so that later image pre processing applications can extract information about the object. Half an object in the FOV isn't sufficient enough for further feature extraction.
A easy way of implementing this is by using the MATLAB function `imclearborder`. This function removes all objects that touches the image border [5]. If a captured frame with part of an object in the FOV is compared with a previous frame containing no object at all, the average threshold that is computed will increase and may exceed UDT, resulting in returning a image showing only a part of the object. The `imclearborder` function needs an additional application that ensures that the whole object is in the FOV before the algorithm returns the resulting image. This application is a matrix ($M_{rm}$) of the same size as the image (370×705 pixels) containing zeros except for a area that is positioned on the left side in the image. This area contains only ones (1-valued pixels) and is called the reference mask (RM). $M_{rm}$ is shown in figure 4.5.

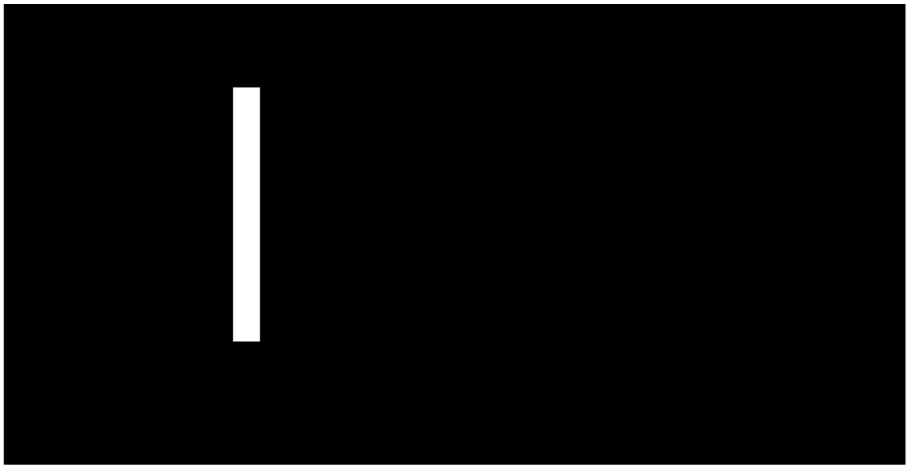


Figure 4.5: Matrix containing a reference mask ($M_{rm}$).

The reference mask is positioned further to the left then to the right because the objects on the conveyor belt enters the FOV from the right in figure 4.5 and leaves the FOV on the left. This ensures that longer objects (in the horizontal direction) is fully inside the FOV. Figure 4.6 shows the reference mask in the cameras FOV.

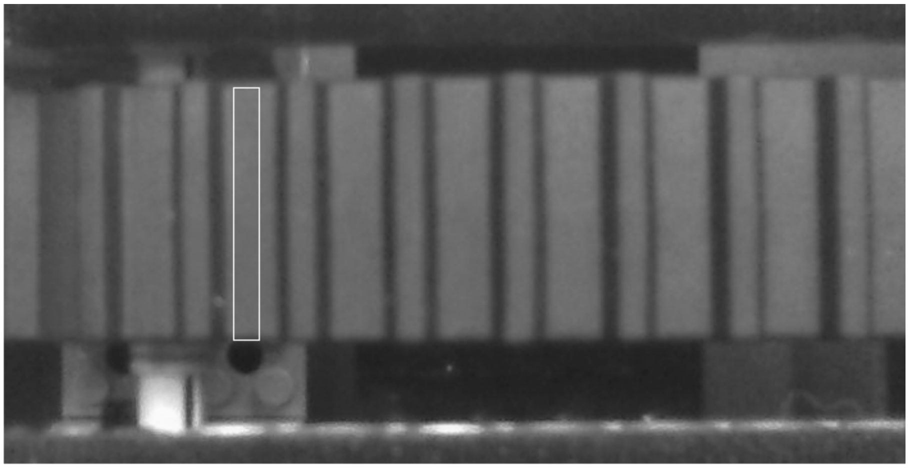


Figure 4.6: The reference mask shown in the FOV.

Every $F_{fd}\{i\}$ (that have been processed by the `imclearborder` function) and the $M_{rm}$ are multiplied together. If the object overlaps RM, the result is a matrix containing 1's where the object overlapped the mask. If the object doesn't overlap the mask, the result is a matrix containing only 0's. Figure 4.7 shows the result of multiplying $M_{rm}$ and a frame differencing image when the object overlaps the mask.

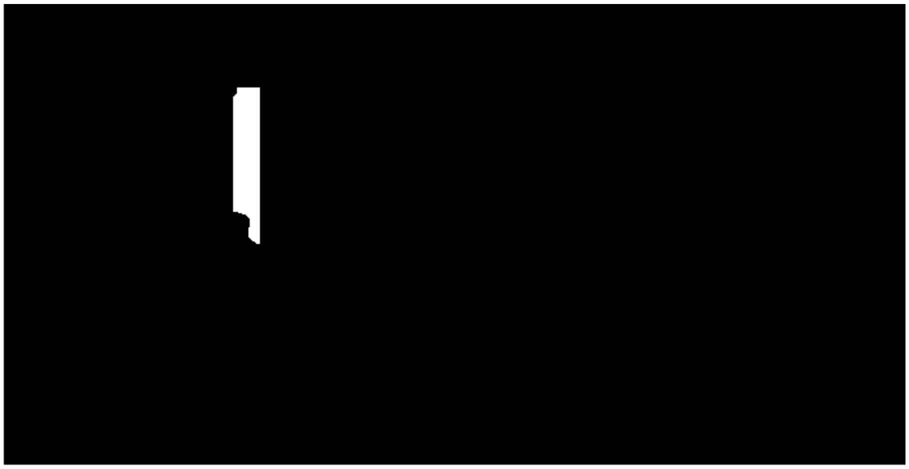


Figure 4.7: The result of multiplying $F_{fd}\{i\}$ and the $M_{rm}$ when object overlaps RM.

The result in figure 4.7 shows that a portion of the object is overlapping RM. To let the algorithm know that the object is either overlapping RM or not, the MATLAB functions `find` and `isempty` are used. The `find` function finds nonzero elements in a matrix and returns the row (r) and column (c) indices for these elements in two vectors. If the matrix contains only 0's the row and column vectors are empty (in MATLAB this is indicated with `[]`). The `isempty` function returns a logical 1 if a vector or matrix contains no nonzero elements and logical 0 otherwise. By inverting this function ($\sim$) it gives a logical 1 if the vectors contains nonzero element and this is used to determine if the object is overlapping the mask.
The area of RM is defined as $M_{rm}(66 : 264, 180 : 200) = 1$.

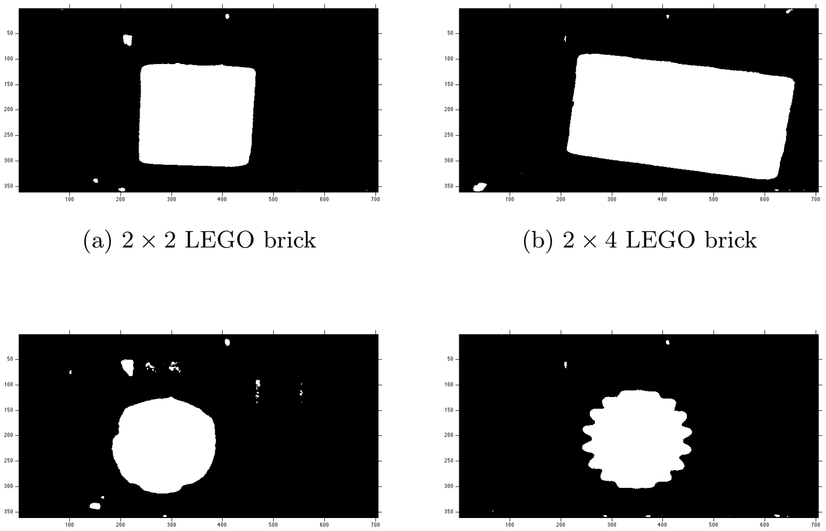
The enhanced pseudocode is shown in algorithm 2

**Algorithm 2** Enhanced Moving Object Detection using BS

```
$i$ & $DET = 1$
while $DET == 1$ do
  get RGB frame from camera
  convert frame to grayscale
  save frame i in F{i}(m, n)
  if $i <= 10$ then
    if $i == 1$ then
      $i \leftarrow 1$
    else
      $F_{fd}(m,n) = F\{i\}(m,n) - F\{i-1\}(m,n)$
      compute T{i-1} from $F_{fd}(m,n)$
      $i \leftarrow 1$
    end if
  else
    $F_{fd}(m,n) = F\{i\}(m,n) - F\{i-1\}(m,n)$
    compute T{i-1} from $F_{fd}(m,n)$
    convert $F_{fd}(m,n)$ to binary image using T
    clear objects that are connected to the border
    $AT\{i-10\} = 0$
    for $j = 0 \rightarrow 9$ do
      AT{i-10} = AT{i-10} + (T{(i-1)-j}/10)
    end for
    multiply $M_{rm}$ and $F_{fd}(m,n)$
    find r and c vectors in $M_{rm} \times F_{fd}(m,n)$
    if !isempty(c) & !isempty(c) & $AT\{i-10\} > UDT$ then
      return $F\{i\}(m,n)$
      $DET = 0$
    else
      $i \leftarrow 1$
    end if
  end if
end while
```

### 4.1.2 Results from the Current Setup

The pseudocode for algorithm 2 was implemented in MATLAB (see Appendix E.11) and tested. The test was to detect different objects when the camera / conveyor belt setup wasn't placed directly below a light source. The next figure shows some results of the MOD.



(a) 2 × 2 LEGO brick

(b) 2 × 4 LEGO brick

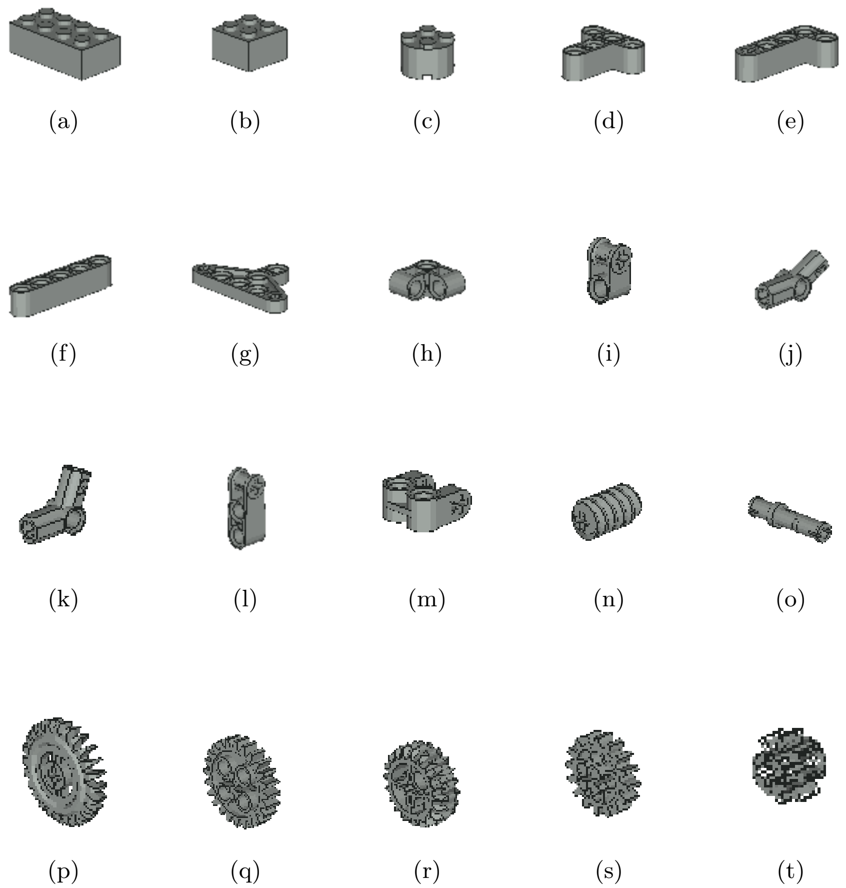
(c) 2 × 2 Round LEGO brick

(d) Technic Gear 16 tooth

Figure 4.8: Some results from the enhanced pseudocode (algorithm 2) implemented in MATLAB.

From the images in figure 4.8 there are some noise present in the frame, but this can be handled in a later stage. The algorithm doesn't return a binary image, but the RGB version to pre-processed further.
The current setup was tested by running the pseudocode for algorithm 2 for 10-20 minutes without having an object passing through under the camera. This was done to check the robustness against the current light conditions in the environment. The result of this test gave no false positive detections.

Another test was also performed to check the pseudocode's ability to detect different types of objects. By letting objects pass through under the camera, the goal was to see how many, of a set of twenty objects, the algorithm could detect on the first try. Figure 4.9 shows the objects that was used in the test.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

(k) (l) (m) (n) (o)

(p) (q) (r) (s) (t)

Figure 4.9: Objects used in the test set. The objects was used to test algorithm 2 failure rate.

Every object was tested five times, so the total number of times the objects passed through, below, the camera was one hundred. Table 4.3 shows the result of the moving object detection for every object.

| **Object** | **(a)** | **(b)** | **(c)** | **(d)** | **(e)** | **(f)** | **(g)** | **(h)** | **(i)** | **(j)** |
|---|---|---|---|---|---|---|---|---|---|---|
| Attempt 1 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Attempt 2 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Attempt 3 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | X | ✓ | ✓ | ✓ |
| Attempt 4 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Attempt 5 | X | ✓ | ✓ | ✓ | ✓ | ✓ | X | ✓ | ✓ | ✓ |
| **Object** | **(k)** | **(l)** | **(m)** | **(n)** | **(o)** | **(p)** | **(q)** | **(r)** | **(s)** | **(t)** |
| Attempt 1 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Attempt 2 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Attempt 3 | X | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Attempt 4 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Attempt 5 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | X | ✓ | ✓ |

Table 4.3: Result of moving object detection on a test set of objects, for algorithm 2.

A checkmark (✓) indicates that the implemented pseudocode detected the object on the first try, and the **X** indicates that it didn't detect the object on the first try.

From this result, the system has a failure rate of 5% with the earlier mentioned light environment, for detecting moving objects. The object identification in table 4.3 refers to the different objects in figure 4.9.

### 4.1.3 Current Illumination Issues

In the current light scenario the camera / conveyor belt setup is not placed directly under a light source. This placement ensures less light reflection and only gives a fail rate of 5% for moving object detection. Otherwise, the room lighting environment is normal fluorescent light (warm).

If the current setup is introduced to a another light source (more intensity or a closer light source) the moving object detection may be affected and detect more false positives. The main enemy for this setup is strong illumination. A pitch black environment is also not desirable.

## 4.2 Reducing the Illumination Problem

One way to reduce the illumination from affecting the system and make it detect false positives, is to generate a background model (BM) based on the running average. This an adaptive approach. Instead of only doing background subtraction with the i-th and (i-1)-th frame, the idea is to generate a BM which can adapt to small changes in the background of an image [7]. This helps both for reducing small changes with illumination and it suppresses some of the movement from the conveyor belt, that can generate false positive detections.
A mathematical expression for BM, based on the running average is as follows,

$$B_i(m,n) = B_{i-1}(m,n) + \alpha\big(F_i(m,n) - B_{i-1}(m,n)\big), \qquad (4.2)$$

where $B_i(m,n)$ is the current background model, $B_{i-1}$ is the previous background model, $\alpha$ is the learning coefficient (also known as the adapt coefficient) and $F_i(m,n)$ is the current frame. This expression is based on formula 11 in [2] and 1 in [18]. When initialized $B_0$ equals $F_0$.

For every new frame that the camera captures, the background model is updated. Afterwards the current frame and the newly updated BM is subjected to frame differencing, as shown in equation 4.1. $B_i(m,n)$ replaces $F_{i-1}$ in this case.
This implementation can handle slow illumination changes and it even makes the system operate under a light source (in the previous sections the system wasn't placed directly under a light source), but it can't handle sudden light changes if a light source is turned off and on again. [2] also proposes a method that does a illumination evaluation that can detect these kind of situations. This report will not rely on or use this method, because the environment the system is intended for should be illuminated at all times with close to normal lighting.

### 4.2.1 The Learning Coefficient - $\alpha$

The learning coefficient as mentioned before is also called the adapt coefficient. This coefficient can have a value between 0 and 1, $\alpha \in [0\ ,1]$ [1].
$\alpha$ determines how fast new changes in newer frames can be implemented in the background model [18]. Large $\alpha$ values allows new changes to be implemented faster in the BM and low values are intended for slow implementation of changes [7] [18]. Both [7] and [18] mentions that $\alpha$ can't be too large because this can produce *artificial tails* behind the moving objects. A typically used value for $\alpha$ is 0.05 [16] and this value is used in the implementation of the BM based on the running average for updating the model, in this report.

[9] uses another algorithm for BM and it's mathematical expression, shown below, is based on [9] (formula 1).

$$B_{i+1}(m,n) = B_i(m,n) + \big(\alpha_1(1 - M_i) + \alpha_2 M_i\big)D_i, \tag{4.3}$$

where $B_{i+1}$ is the current background model, $B_i$ is the previous background model, $M_i$ is the binary moving objects hypothesis mask, $D_i$ is the difference between the current frame and background model and the learning coefficients $\alpha_1$ and $\alpha_2$ are based on an estimate of the rate of change in the background. In [9] these learning coefficients are estimated using Kalman filter, but the article mentions that small values for both $\alpha_1$ and $\alpha_2$ gave good results.

### 4.2.2 The Final Algorithm

In the final algorithm the BM based on the running average is implemented. The algorithm is shown below.

**Algorithm 3** Enhanced Moving Object Detection using BS and BM - `MOD()`

```
$i$ & $DET = 1$
while $DET == 1$ do
  get RGB frame from camera and save frame in $F_{RGB}\{i\}(m,n)$
  convert frame to grayscale and adjust contrast
  save frame i in $F\{i\}(m,n)$
  if $i <= 30$ then
    if $i == 1$ then
      $B\{i\}(m,n) = F\{i\}(m,n)$
    else
      $B\{i\}(m,n) = B\{i-1\}(m,n) + \alpha\big(F\{i\}(m,n) - B\{i-1\}(m,n)\big)$
      $F_{fd}(m,n) = F\{i\}(m,n) - B\{i\}(m,n)$
      compute $T\{i-1\}$ from $F_{fd}(m,n)$
    end if
    $i \leftarrow 1$
  else
    $B\{i\}(m,n) = B\{i-1\}(m,n) + \alpha\big(F\{i\}(m,n) - B\{i-1\}(m,n)\big)$
    $F_{fd}(m,n) = F\{i\}(m,n) - B\{i-1\}(m,n)$
    compute T{i-1} from $F_{fd}(m,n)$
    convert $F_{fd}(m,n)$ to binary image using T{i-1}
    clear objects that are connected to the border and fill holes
    AT{i-30} = 0;
    for $j = 0 \rightarrow 9$ do
      AT{i-30} = AT{i-30} + (T{(i-1)-j}/10)
    end for
    multiply $M_{rm}$ and $F_{fd}(m,n)$ and find r and c
    if !isempty(r) & !isempty(c) & $AT\{i-1\} > UDT$ then
      return $F_{RGB}\{i\}(m,n)$
      $DET = 0$
    else
      $i \leftarrow 1$
    end if
  end if
end while
```

$B\{i\}(m,n)$ is the current background model and $B\{i-1\}(m,n)$ is the previous background model. The learning coefficient $\alpha$ is initialized as 0.05. The purpose of the thirty first frames is to generate the BM and compute the thresholds from the frame differencing expression.

## 4.3 Mechanical Adjustments

The conveyor belt unit shown in figure 4.10 is placed above a dark surface to create less contrast between the moving conveyor belt and it's underlying surroundings.

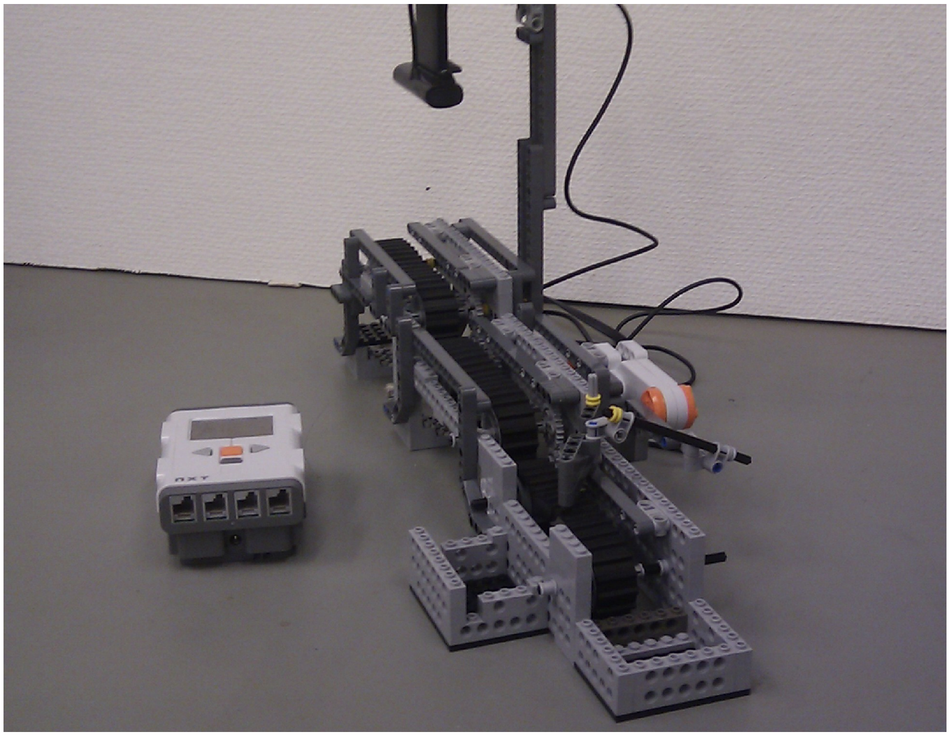
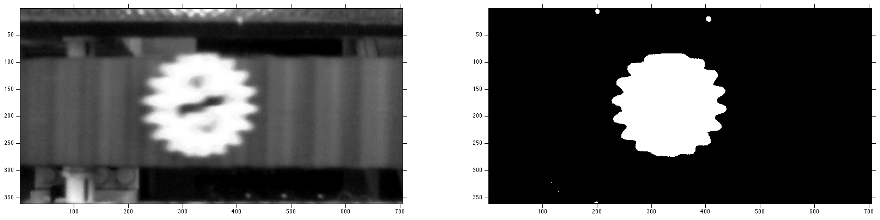


Figure 4.10: The conveyor belt unit.

In the figure 4.10, the surface that the unit is placed on can be to bright in some circumstances, especially if there is a light source right above. Easy mechanical enhancements, like placing the unit on a dark surface, can ensure a better result of detecting an object. Also the beams that is place along the conveyor belt at both sides produces light reflections where the camera is placed. These beams are covered with a dark material to reduces this effect.

## 4.4 Conveyor Belt Speed

The speed of the conveyor belt is controlled by the LEGO Mindstorms motor / servo. The motor properties and settings are implemented in MATLAB using the RWTH - Mindstorms NXT toolbox. To initialize the power / speed of the servo, first create a motor object in MATLAB which contains the motor properties. One of these properties is to set the power of the motor. This can be done by giving the power property a integer from -100 to 100, or 0 to 100%. Positive and negative values determines the direction of rotation (one rotation on the motor is 360°) [13].
If the power property are set to a high value, the camera detects the object, but because of the fast movement the captured and return frame, containing the object, can be bit blurry. An example is shown in the figure below.

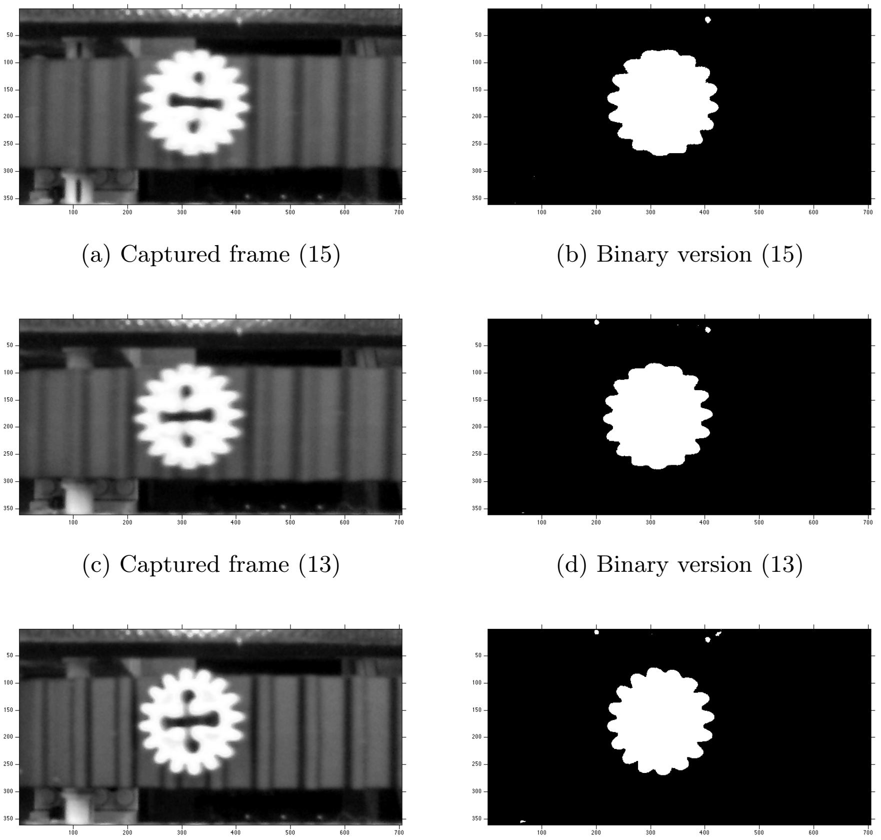


(a) Captured frame

(b) Binary version

Figure 4.11: Resulting images from the moving object detection when the conveyor belt speed is set to 20.

From figure 4.11a the image that is captured are blurry and details from the object shape is lost. Both images in 4.11 shows this. To prevent the result from the MOD to lose object details, the conveyor belt speed is reduced. A simple test was performed to find an optimal parameter for deciding the speed by only reducing it until the result from the detection was satisfying. Figure 4.12 show some results from this test.



(a) Captured frame (15)

(b) Binary version (15)

(c) Captured frame (13)

(d) Binary version (13)

(e) Captured frame (10)

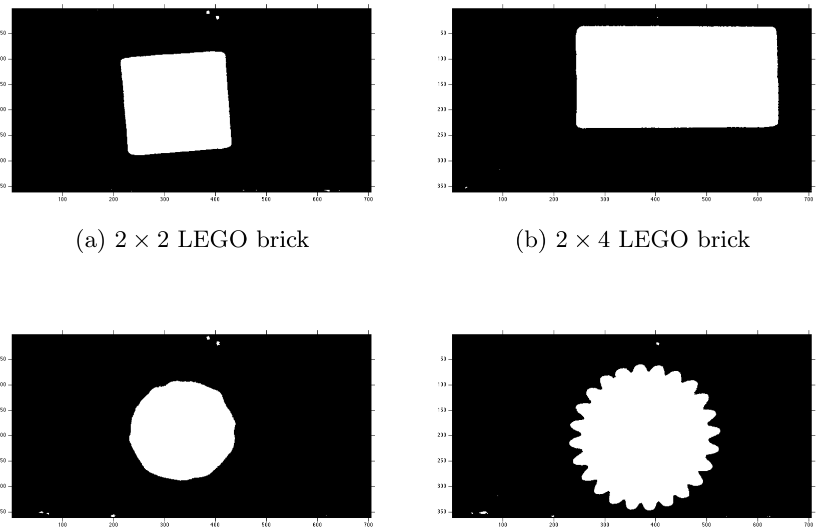
(f) Binary version (10)

Figure 4.12: Resulting images from the moving object detection when the conveyor belt speed is set to 15, 13 and 10.

From figure 4.12 the best result is when the conveyor belt speed is set to 10.

## 4.5 Results from the Final Setup

The pseudocode for algorithm 3 was implemented in MATLAB (see Appendix E.2). Now the camera / conveyor belt setup is placed directly under a light source while it was tested. In figure 4.13 the same objects that was tested in section 4.1.2 is again being looked at.



(a) 2 × 2 LEGO brick

(b) 2 × 4 LEGO brick

(c) 2 × 2 Round LEGO brick

(d) Technic Gear 16 tooth

Figure 4.13: Some results from the final pseudocode (algorithm 3) implemented in MATLAB.

As the images show in the figure above, the background noise is reduced and the objects are more detailed because of the BM and lowering the speed of the conveyor belt.
The UDT value had to be changed in this experiment because of more illumination. A similar test like the one presented in section 4.1 was performed and UDT was set to 0.18.

### 4.5.1 Test: Ability to Detect Objects with the Final Algorithm

Figure 4.9 presented all the objects that was used to test the ability to detect different types of objects on the first try for algorithm 2 . Now the same test set will be used on the final algorithm (3). The results are shown in the table below.

| **Object** | **(a)** | **(b)** | **(c)** | **(d)** | **(e)** | **(f)** | **(g)** | **(h)** | **(i)** | **(j)** |
|---|---|---|---|---|---|---|---|---|---|---|
| Attempt 1 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Attempt 2 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Attempt 3 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Attempt 4 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Attempt 5 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Object** | **(k)** | **(l)** | **(m)** | **(n)** | **(o)** | **(p)** | **(q)** | **(r)** | **(s)** | **(t)** |
| Attempt 1 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Attempt 2 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Attempt 3 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Attempt 4 | ✓ | ✓ | X | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Attempt 5 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Table 4.4: Result of moving object detection on a test set of objects, for algorithm 3.

From table 4.4 there was only one incident were object **(m)** wasn't detected the first time, but on the second. The result is better with the new algorithm and the failure rate has gone from 5 to 1%. It can also be mentioned that the test was performed while the camera / conveyor belt setup was placed directly below a light source and this placement wasn't used when the last test was performed.

# Chapter 5

# Object Representation and Feature Extraction

For a human it is easy to detect and recognize objects in the environment, even if the object is partially hidden behind a obstacle. The human brain keeps track of different objects and knows the difference between them. It's like a hard drive that contains a template for each object, that has been learned from early childhood to the present, and information about them. When a known object is detected, it is matched up against the templates, and the information is given. To implement this in a system would require a database that contain the templates and information about every object that the system should recognize. The system that includes a camera could capture a image of the object and compare it to the templates until the best match is found and then return the stored information about the current object. A simpler way of doing this is to segment the image into regions. These regions consists of connected components, also known as pixels, and could be a object in the image scene. There are two ways of representing these regions and that is by external and internal representation. An external representation focuses on shape and an internal focuses on color and texture [5]. These characteristics are among the first thing a human would look for when recognizing a object. When looking at an apple, the first thing a human may detect is that the apple is close to being round, the color is red or green and the apples texture is smooth. In the same way a system could detect these characteristics from the segmented regions in an image. It could look for corners, length, width, areal, shape and even color and use this to describe an object. The mentioned characteristics is also called features, and can be extracted from the regions and used to describe and represent the object.

## 5.1 Preparing the Image Received from MOD

When the MOD detects an object the function will end and return the frame containing the whole object within the FOV. This frame is a RGB version and it's converted to a binary image because further image pre processing, in this report, are based on binary mathematical morphology (MM) operations.

A RGB image contains three components; Red, Green and Blue. These components can be extracted from a RGB image and produce three new grayscale images that separately contain information about the three colors. An algorithm was developed to determine which component had the most information about the object color in the frame and was best suited to extract the object from. This algorithm receives the RGB frame from `MOD()` and finds the best possible image-component that can extract the object from the background. It compares the computed thresholds (uses `graythresh`) from each component and returns the binary version of the image-component with the highest threshold value.
Algorithm 4 shows the pseudocode of this function.

**Algorithm 4** Segmenting Object from Background using Information about RGB Components - `rgbComp()`.

```
Input: $F_{RGB}$
for $i = 1 \rightarrow 3$ do
    $Comp\{i\} = F_{RGB}(:, :, i)$
    $T[i] = \text{graythresh}(Comp\{i\})$
end for
find the index ($i_{max}$) for the largest threshold value in $T[i]$
convert $Comp\{i_{max}\}$ to binary using $T[i_{max}]$
return binary image
```

$Comp\{i\}$ is a cell containing each component and $thr[i]$ is a vector containing each components computed threshold. Algorithm 4 was implemented in MATLAB and can be found in E.3.

After some testing it was found that adjusting the contrast in each component before computing the threshold gave the best result. Some of the RGB images with an object that had a yellow color, contained much of the same color information as the background (the three components also contained very similar color information) and had the largest computed threshold value. When this image was chosen, it couldn't extract the object from the background and the result was meaningless. This didn't happen every time, but it is dependent on the illumination and it's implemented as a safeguard. When the contrast is adjusted the object stands more out in the image and it is more brighter than the background, making it easier to extract the object without including noise from the background. Table 5.1 shows computed threshold values for each component for four different objects (shape and color) with and without adjusting the image contrast.

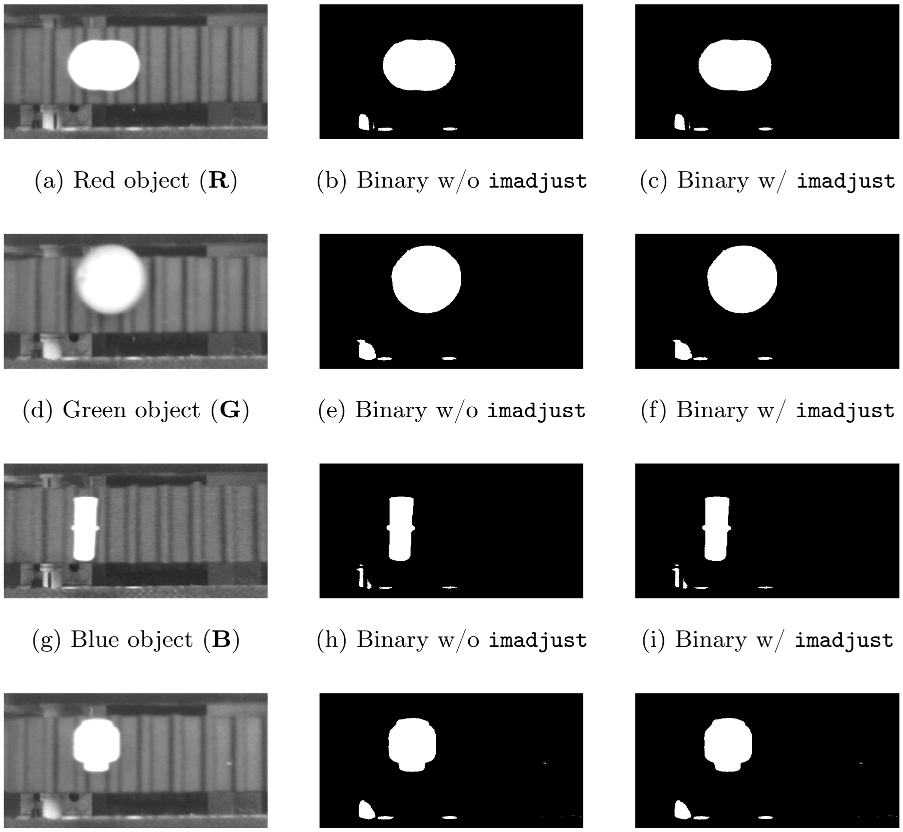
| **Object color** | **Threshold w/o** `imadjust` | | | **Threshold w/** `imadjust` | | |
|---|---|---|---|---|---|---|
| | **R** | **G** | **B** | **R** | **G** | **B** |
| RED | 0.6784 | 0.3882 | 0.3451 | 0.6078 | 0.3549 | 0.4196 |
| GREEN | 0.3608 | 0.6627 | 0.3686 | 0.4118 | 0.5843 | 0.3784 |
| BLUE | 0.3608 | 0.3961 | 0.6667 | 0.4412 | 0.2961 | 0.6000 |
| YELLOW | 0.6823 | 0.6823 | 0.5804 | 0.6098 | 0.6078 | 0.5020 |

Table 5.1: Computed threshold values for each R, G and B component with and without adjusting the image contrasts for four different color and shaped objects. The MATLAB function `imadjust` adjusts the contrast.

The gray cells in the table indicates the maximum threshold values computed by the MATLAB function `graythresh` for each component.

When comparing all the threshold values in table 5.1, it shows that all the maximum computed thresholds are lower when the image is contrast adjusted. In the case with the yellow object the R and G component are equal when the image isn't exposed to contrast adjustment. In some cases (different light environment) with a yellow object all the components can be of approximately the same value and especially the B component contains a lot of the background information. By using the threshold values computed by

each contrast adjusted component, rgbComp() extracted and separated the object from the background in every experiment.
Figure 5.1 shows the best RGB components and the binary version of it, w/ and w/o adjusting the contrast, from the objects in table 5.1.



(a) Red object (**R**) (b) Binary w/o imadjust (c) Binary w/ imadjust

(d) Green object (**G**) (e) Binary w/o imadjust (f) Binary w/ imadjust

(g) Blue object (**B**) (h) Binary w/o imadjust (i) Binary w/ imadjust

(j) Yellow object (**R**) (k) Binary w/o imadjust (l) Binary w/ imadjust

Figure 5.1: Objects extracted from **RGB** components w/ and w/o adjusting the contrast.

Algorithm 4 was implemented because the result from converting a RGB image to grayscale, compute the threshold value from this image and then convert it into a binary image gave varying results. The results sometimes contained just a part of the object. Other times the reflection from the conveyor belt surrounding the physical object could be added as part of the object in the binary image. In the experiments performed on algorithm 4, the object was always extracted without being affected by the varying illumination.

After algorithm 4 has returned the binary image with the best result, the next stage is to prepare the image for further preprocessing by doing noise reduction and labeling the object of interest (OOI). Figure 5.2 shows an image component that have been selected by the function in E.3 to be further pre processed.

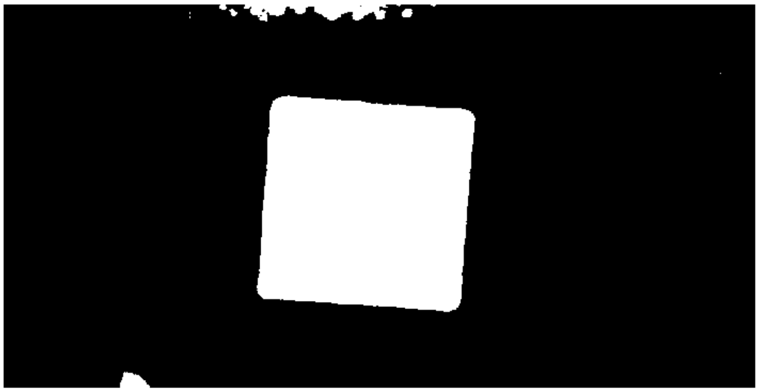


Figure 5.2: Image received from `rgbComp()`.

When the image in figure 5.2 is received it is first processed by two MATLAB functions. The first one is the `imfill` function that ensure to close/fill holes in an binary image. A hole in a binary image is defined as a dark area surrounded by lighter pixels [5]. This function is intended to close the object surface and make it whole if the object contains holes.
The second function is used to clear objects that are on the image border and it is called `imclearborder`. This function is also mentioned in section 4.1.1 where it was used to ensure that the whole object was in the cameras FOV. Figure 5.3 shows the result after the `imfill` and `imclearborder` functions have been used on the image.

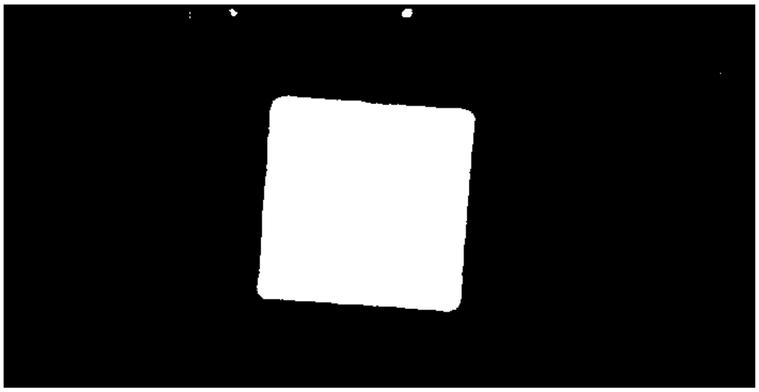


Figure 5.3: Result of closing holes and clearing unwanted objects on the image border.

From the figure above there are still some noise left in the image. The next section explains how to remove or reduce this noise.

### 5.1.1 Noise Reduction

In binary images, objects are represented by ones (white) and the background are represented by zeros (black), so that each pixel in a binary image has the value 1 (foreground pixel) or 0 (background pixel) [5]. Every foreground pixel can be considered as an object or as noise in the image. Often single foreground pixels or small groups of foreground pixels are noise.
Morphological opening is used to reduce the noise. The OOI is assumed to be larger than noise objects. The structural element (SE), used in the opening, is a diamond shaped element with a radius equal to 4 pixels. This element reduces the noise and doesn't deform the OOI's contour too much. Figure 5.4 shows the result of opening.

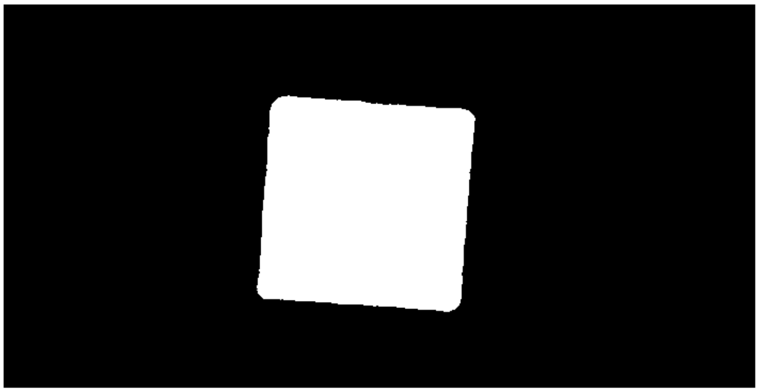


Figure 5.4: Result of morphological opening on the test image.

From the image above all noise have been removed and only the OOI remains, but this doesn't have to be the case every time. It depends on the illumination. A bright environment causes more reflection from the conveyor belt and this can affect the noise in the image. Due to illumination variations the opening function can't remove all noise, but the function reduces it.
After reducing noise in the image, erosion is performed on the object to smooth the contour. A smaller diamond shaped SE are used when performing the erosion (radius = 1 pixel).
The functions for performing morphological opening and erosion in MATLAB are `imopen` and `imerode`.

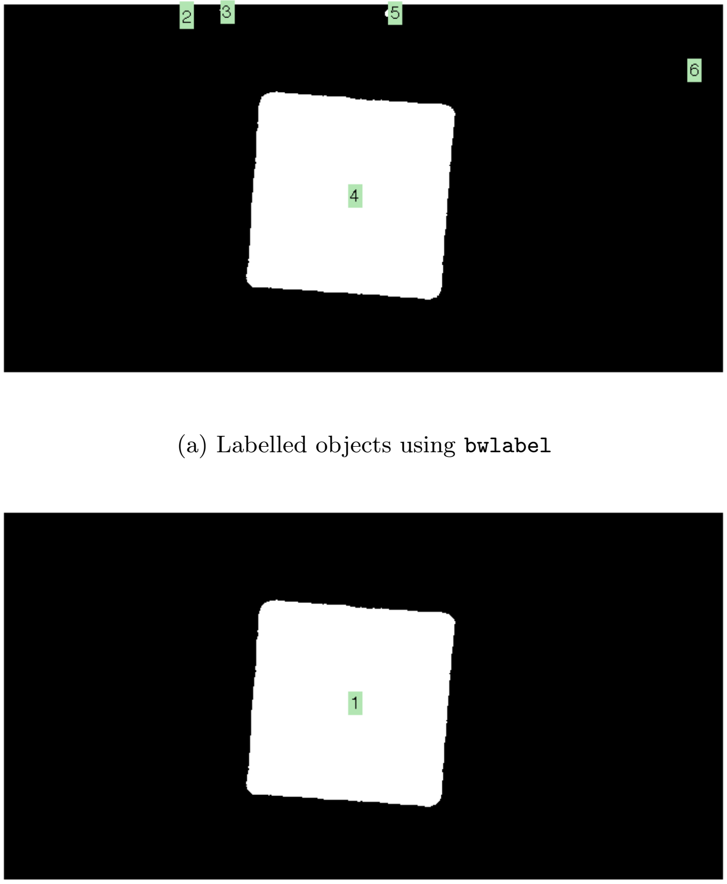
### 5.1.2 Labeling the Object

When performing labeling, the idea is to give an object in a image a reference or name. As mentioned in section 2.3.5 a object is a set of connected components. The connectivity between the components can be either a 4- or 8-connected neighborhood. When labeling objects in MATLAB a function called `bwlabel` are used. This function uses 8-connected connectivity as default and returns the number of objects found in a binary image and a label matrix ($LM$) containing the reference to every object.

If an image containing the OOI also contains noise, it will affect the labeling. Noise, that's in the image, will be perceived as an object and `bwlabel` will find more than one object. If there is more than one object the algorithm has to find the OOI and this can be done by the MATLAB function `bwselect` and earlier information about the area of the reference mask (RM) as defined in section 4.1.1.
The steps listed below explains how the algorithm finds and extracts the OOI.

- `bwselect` finds a object in a user defined coordinate and returns a binary matrix containing only this object.
- The function searches for a object in two different coordinates in and around RMs area.
- Coordinates (n, m): 180, 160 (in RM) and 220, 160 (to the right of RM).
- Worst scenario: object is detected by MOD when the right part of the object overlaps n-coordinate 180. The object is then on it's way out of RM and out of the image on the left side.
- Best scenario: MOD detects the object when it's left side is overlapping n-coordinate 200. Then the object has only just overlapped RM.
- Objects with uneven contours (gear) might not overlap the coordinate 200, 160. Therefor the n-coordinate is set to 220.
- If the result from the first coordinate does not contain the OOI (contains only zeros), the next coordinate are checked.

Figure 5.5 shows two images. The first image shows a labelled image without using `bwlabel` and the second images shows the result of using `bwselect`.



(a) Labelled objects using `bwlabel`

(b) Labelled object using `bwselect`

Figure 5.5: Result from using `bwlabel` and `bwselect`.

Figure 5.5a shows six labelled objects in the image and figure 5.5b shows just the labelled OOI after using `bwselect` on the mentioned coordinates.

## 5.2 Representation of the Object

MATLABs interpretation of a binary image is a matrix that contains zeros (background) and ones (objects and/or noise). Except from this information MATLAB doesn't know much about the OOI because the object isn't represented. As mentioned in section 2.4 internal and external characteristics can describe the OOI. These characteristics can be used as object features and provides MATLAB with data describing the OOI. The object representation can also help MATLAB find more features.

### 5.2.1 Finding Internal Characteristics

To represent a object with internal characteristics the MATLAB function `regionprops` is used. The syntax for use of `regionprops` in MATLAB is as follows [11]:

```
Data = regionprops(LM, properties)
```

This function finds properties in image regions [11].
The `regionprops` function needs the labelled matrix ($LM$) as an input parameter and the user can define what kind of data the function will return as a second input parameter. If only $LM$ is used as a input parameter, `regionprops` returns a structure array containing object area, centroid and bounding box. The structure array containing object data will have more indices than one if $LM$ contains two or more objects.
The parameter `properties` can be defined as `'all'`, `'basic'` or ,as mentioned before, not be defined at all. When `'all'` is defined, `Data` returns all shape measurements [11]. A list of measurements can be found in [5] (page 643) or in [11] (Image Processing Toolbox → `regionprops`).
Defining `'basic'` or not defining the `properties` parameter at all does the same thing and makes the function return only area, centroid and bounding box. The internal characteristics that are interesting in this report will be the area and the centroid, which is the coordinate in the objects center of gravity. The bounding box is a vector containing the upper left coordinate for $n$ and $m$, and the length and width of the box. The bounding box represents the smallest rectangle containing a object [5]. Figure 5.6 shows the

bounding box containing an object.

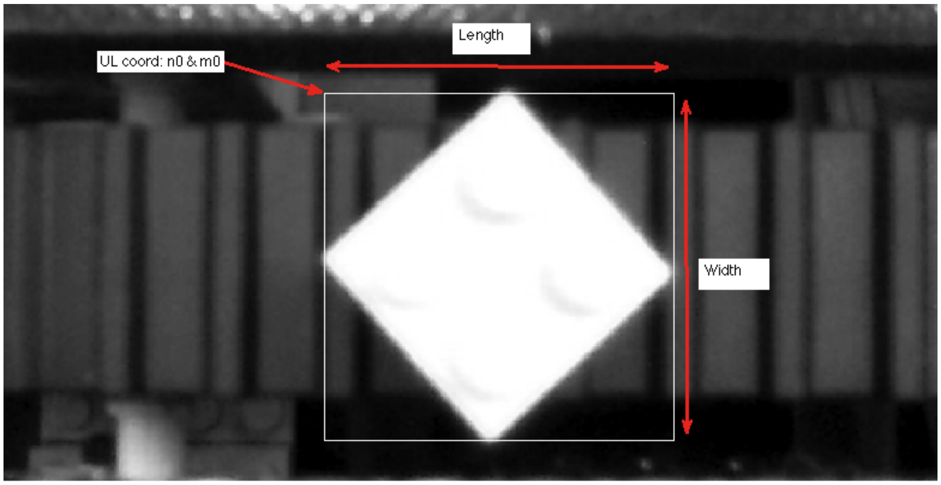


Figure 5.6: Image containing the objects bounding box.

### 5.2.2 Finding External Characteristics

The object perimeter, or boundary, can be an useful external characteristic. This information and the coordinate for the centroid is used when finding the objects signature. The signature operation will be explained in section 5.2.3 and this section will focus on how to find the perimeter / coordinates. The MATLAB function `bwperim` is used to find and extract an object perimeter and the result is a binary image that only contains the perimeter (white pixels) [11]. An example is shown in figure 5.7.

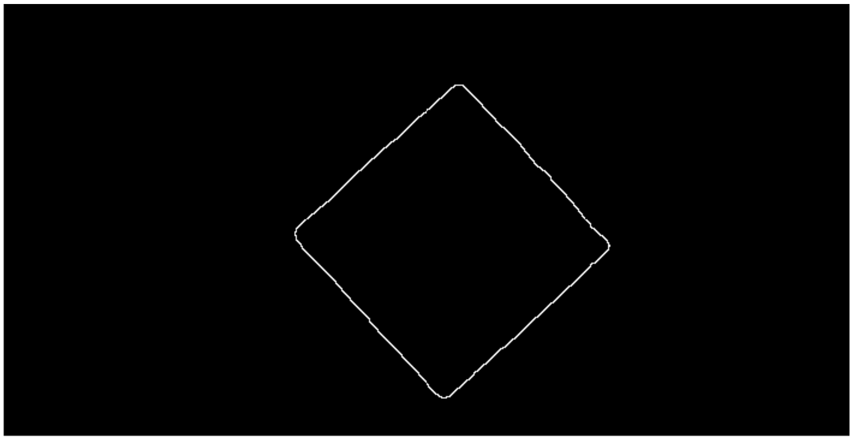


Figure 5.7: Result of using the MATLAB function `bwperim` on a binary image containing an object.

The function defines a pixel as part of the perimeter if the value of it is nonzero and it's connected to at least one zero valued pixel [11].

The figure above shows the result from using `bwperim` on a binary image, but it doesn't return the coordinates of the boundary. To extract the coordinates from this result, another MATLAB function is needed. This function is called `bwboundaries` and it traces an object perimeter [11]. The result is an Q-by-2 vector containing all the coordinates for the object perimeter. Q is the number of coordinates found on the perimeter.
The general syntax for using this function in MATLAB can be written as,

```
BC = bwboundaries(IBW, conn, options)
```

where `BC` is the Q-by-2 vector, `IBW` is the binary image containing object(s), `conn` specifies the connectivity (4- or 8-connected neighborhood) between perimeter pixels and `options` is an optional parameter that the user defines as either `'holes'` or `'noholes'`. If an object has a hallow area (containing only 0 valued pixels) inside it's own perimeter and this area contains another object like in figure 5.8, the outer object is defined as the parent object and the inner object is the child.

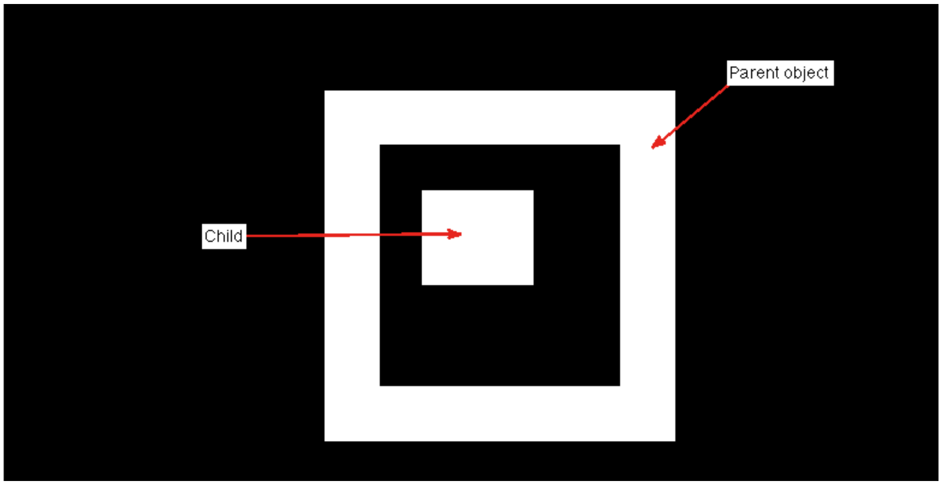


Figure 5.8: Binary image with an parent object and a child.

If the option ’holes’ is defined, bwboundaries will return the coordinates for the outer and inner parent object perimeter. Are ’noholes’ defined it will only return the outer perimeter of the parent and the child. If the object surface contains no holes and does not have a child, both ’holes’ and ’noholes’ can be used, but [11] prefers the second option because it provides better performance than the option ’holes’.

Vector BC is organized so that BC(Q, 1) contains all m-points (y axis) and BC(Q, 2) all n-points (x axis) of the perimeter. The function signature.m needs both boundary coordinates and the coordinate of the objects centroid as inputs when it computes the signature and it requires BC to be organized so that all n-points are in BC(Q, 1) and m-points are in BC(Q, 2). This is done by extracting each column in BC separately and rearrange them so that it meets signature.m requirements, as shown below.

```
Iperimeter = bwperim(IBW, 8);
BC = bwboundaries(Iperimeter, 'noholes');
m = BC{1}(:, 1);
n = BC{1}(:, 2);
BC(:, 1) = n;
BC(:, 2) = m;
```

#### 5.2.2.1 Algorithm for the Object Of Interest

The following algorithm was developed for the OOI and implemented in MATLAB. The implementation can be found in appendix E.4.

**Algorithm 5** Finding the Object Of Interest - `ooi()`.

```
Input: $Comp\{i_{max}\}$
Fill any holes in the binary image using imfill.
Clear unwanted objects on image border using imclearborder
Perform opening and then erotion using imdilate and imerode
$[LM\ junk] = \texttt{bwlabel}(\texttt{Comp}\{\texttt{i}_{\texttt{max}}\})$
$img = \texttt{bwselect}(\texttt{LM}, 180, 160)$
if $img == M0$ then
  $img = \texttt{bwselect}(\texttt{LM}, 220, 160)$
  if $img == M0$ then
    $img = M0$
    $OData = [\,]$
    $BC = [\,]$
  else
    $OData = \texttt{regionprops}(\texttt{img} == 1)$
    $BC$ = object border coordinates
  end if
else
  $OData = \texttt{regionprops}(\texttt{img} == 1)$
  $BC$ = object border coordinates
end if
return $img$, Odata and $BC$
```

The algorithm returns a binary image containing only the OOI, object data and the coordinates of the boundary.

### 5.2.3 Object Signature

In [5] the authors presents a function called `signature.m`, which is based on the theory in section 2.4.2.1. The m-file is shown in appendix E.5.
The MATLAB syntax [5] for the `signature` function is as follows,

```
[sign, angle] = signature(BC, n0, m0)
```

where `sign` and `angle` is a 360-by-1 vector containing the distances and angles, `BC` is the earlier mentioned coordinates of the object border and `n0` / `m0` is the coordinate of the objects center of gravity.

The function extracts the Cartesian coordinates from `BC` by defining them as `xcart = BC(:, 2)` and `ycart = -BC(:, 1)`, or in other words $xcart = m$ and $ycart = -n$ and then converts them to polar coordinates using the MATLAB function `cart2pol`. This function returns the signature and angle for all distances. The angles are then converted to degrees and some safety measurements are done so that duplicate angles are deleted [5].

An example of converting Cartesian coordinates to polar is shown in appendix C.1.

## 5.3 Extracting Features

All features extracted from an object will be stored in a feature vector $\underline{x}$ and be used by a pattern classifier. Chapter 6 conducts experiments on three different classifiers with three different combinations of features. The data gathered to this experiment contained four different descriptors in the feature vector; area, mean value of signature, standard deviation of signature and the number of vertices / peaks in the signature. This 4-dimensional vector is defined as follows,

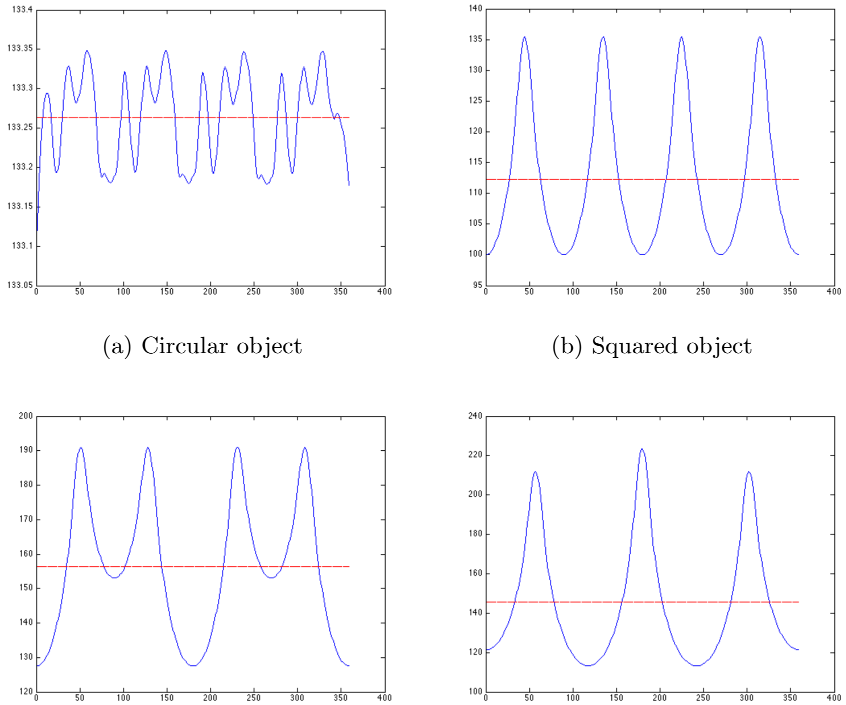
$$\begin{aligned}\underline{x} &= \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}^T \\ &= \begin{bmatrix} A & M & Std & NoP \end{bmatrix}^T \end{aligned} \tag{5.1}$$

where $A$ is the area, $M$ and $Std$ is the mean and std value of the signature and $NoP$ is the number of peaks in the signature. The area is computed by the `regionprops` function, the mean and standard deviation is computed by the two MATLAB function `mean` and `std2`. The number of vertices is found by analyzing the signature.

The dimensionality of the final feature vector depends on which combination of features that gives the lowest classification error rate. The combinations of features are presented in Chapter 6.

### 5.3.1 Mean and Standard Deviation

The values computed from the `mean` and `std2` can describe a signatures form and variation.
The mean values gives an impression of the average distance from an objects center to it's boundary. For objects that have a similar geometric shape like the rectangular object in figure 5.9c, the mean value would be higher than an more compact object that is as long as it is wide.



(a) Circular object

(b) Squared object

(c) Rectangular object

(d) Triangular object

Figure 5.9: The figure shows the signature for four different objects. In addition the mean is marked with a red line in each signature.

Very low std values indicates that the distance from the object center to the boundary is similar for all 360 increments and that the object is near to having a circular shape, like the signature in figure 5.9a. In table 5.2 the values of these objects can be found and the std value of an circular object is very low.

If an object is not circular shaped, but is still compact indicating that the object is approximately as high as it is wide, like a square, the std value is still low if comparing it to the rectangular or even the triangular shape.

| | Objects | | | |
|---|---|---|---|---|
| | Circular | Squared | Rectangular | Triangular |
| **Mean** | 133.2631 | 112.2056 | 156.3376 | 145.8192 |
| **Std.** | 0.0548 | 11.6228 | 19.6697 | 31.3234 |

Table 5.2: Mean and std values for the objects in figure 5.9.

### 5.3.2 Finding Vertices in Signature

Peaks / vertices in the signature is also a source of information about an object. If an object has a geometric shape the number of vertices in the signature can indicate what kind of object it is. The table below shows the most likely shape of an object based on the amount of vertices in the signature.

| **Vertices** | **Object Description** |
|---|---|
| 3 | Most likely a triangular shape. |
| 4 | Most likely a square shape. If the distance between each peak is approx. 90°, the shape is more likely a square than a rectangle. |
| 5 | Pentagon. |
| $\geq 6$ | Hexagon / octagon, etc., but it could also be a circle because the boundary isn't perfect and is varying a lot (low variation, but many vertices). |

Table 5.3: Object description based on the amount of vertices in a signature for geometric shaped objects.

Beside from these shapes, where the amount of peaks on the signature indicates which geometric shape the object has, the amount of peaks can give information about other objects as well. E.g the number of teeth on a mechanical gear.
To detect peaks in signatures algorithm 6 was developed.

**Algorithm 6** Detect Peaks in Signature - `peaks()`.

```
Input: sign, angle
$i \leftarrow 2$
**while** $i < length(angle)$ **do**
  $sign(i)$
  **if** $sign(i) > sign(i - 1)$ & $sign(i) > sign(i + 1)$ **then**
    **if** $sign(k) > sign(i)$ **then**
      $peak \leftarrow 1$
      $p(peak) = angle(i)$
    **end if**
  **end if**
  $i \leftarrow 1$
**end while**
**if** $sign(1) > sign(p(1))$ **then**
  $peak \leftarrow 1$
**end if**
$NoP = peak$
**return** $NoP$
```

This algorithm compares the ith element with the (i-1)-th and (i+1)-th element in the `sign` vector. Only if the i-th element is bigger than both the (i-1)-th and (i+1)-th element a peak is detected.

Objects on the conveyor belt could be placed like the object in figure 5.6 and then the first and last element in the `sign` vector indicates a peak. The MATLAB function `signature` always starts computing the distance from an objects centroid to the boundary on the left side (angle = 0°) of the object in an image.

Algorithm 6 always checks the first element in the vector and compares it to the first peak found in position $p(1)$. If the value of this element is bigger than $sign(p(1))$, it indicates a peak and *peak* is incremented. The signature of the object in figure 5.6 is shown in figure 5.10.

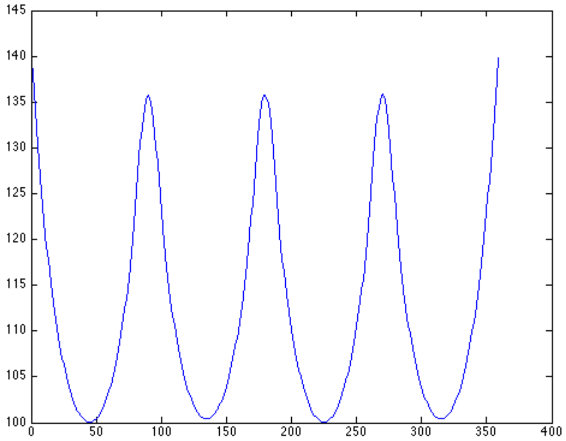


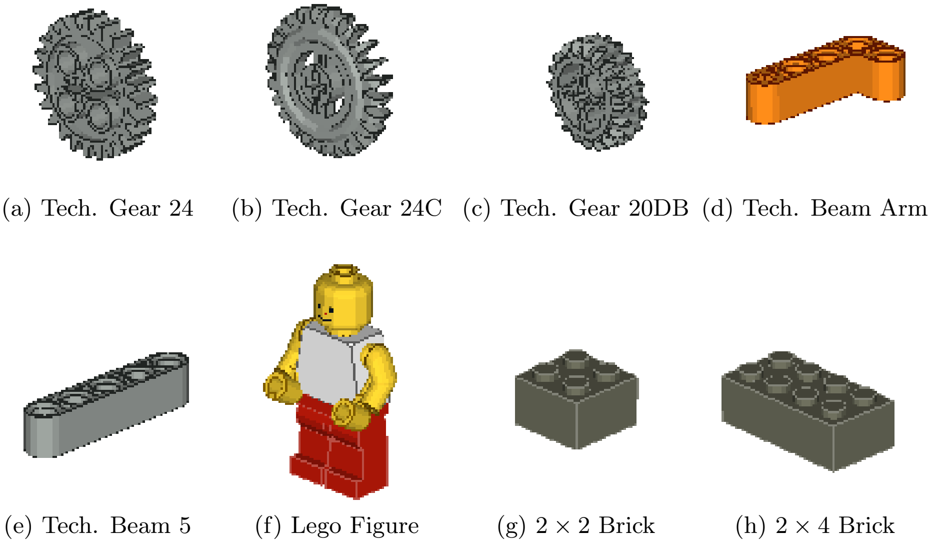
Figure 5.10: The signature to the object in figure 5.6.

From the figure above, both the first and last element in the signature are bigger then the first peak, indicating a fourth peak.

Algorithm 6 is implemented in MATLAB and the code can be found in appendix E.6.

## 5.4 Results from `ooi()`

Figure 5.11 shows a set of objects that was used to test the `ooi()` function.



(a) Tech. Gear 24 (b) Tech. Gear 24C (c) Tech. Gear 20DB (d) Tech. Beam Arm

(e) Tech. Beam 5 (f) Lego Figure (g) 2 × 2 Brick (h) 2 × 4 Brick

Figure 5.11: Test objects for the OOI function.

The data in table 5.5 shows the result from the `ooi()` on all the objects in figure 5.11. These objects was only tested once. The data is stored in the feature vector `X`. Also shown are the computational time it took MATLAB after the object was detected by `MOD()` to the object was represented and the data was stored in the feature vector.

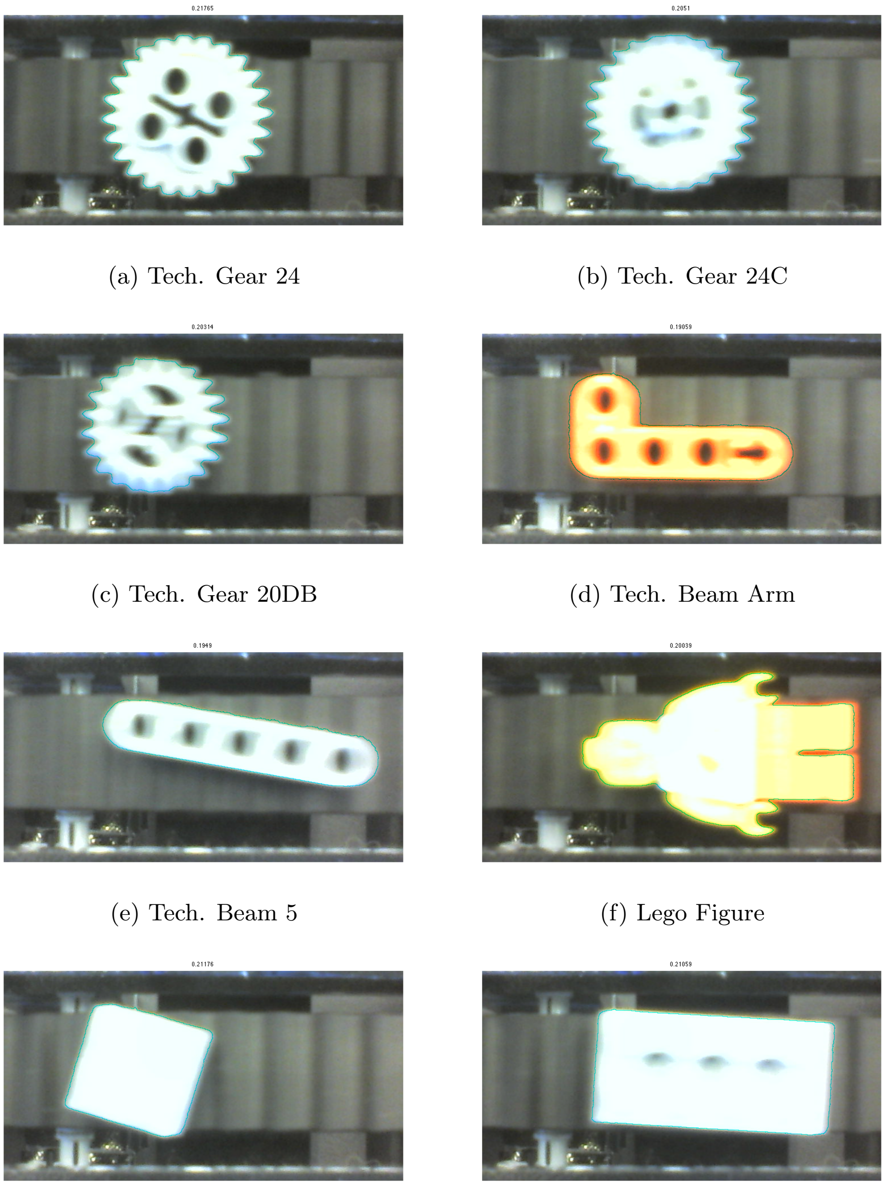
| **Object** | **Area** | **Mean** | **Std.** | **NoP** | **Computational Time** |
|---|---|---|---|---|---|
| **a)** | 61378 | 139 | 7 | 23 | 3.1431* |
| **b)** | 60190 | 138 | 5 | 20 | 0.6305 |
| **c)** | 43876 | 117 | 7 | 19 | 0.4485 |
| **d)** | 46995 | 106 | 60 | 4 | 0.4569 |
| **e)** | 44337 | 102 | 66 | 2 | 0.4333 |
| **f)** | 83876 | 157 | 42 | 8 | 0.4391 |
| **g)** | 43160 | 116 | 12 | 4 | 0.4460 |
| **h)** | 83537 | 155 | 48 | 4 | 0.4484 |

Table 5.5: Results from the OOI on the objects presented in figure 5.11.

* The computational time for object **a)** is a lot longer because it's the first object that is introduced to the system and it has to run through all the functions for the first time. After this test the system was stopped and then started again. A object was placed on the conveyor belt and registered by the system. The time it took was closer to the computational times found in table 5.5 (except for **a)**).

Figure 5.12 shows all eight object of interest with an outlined border.

Figure 5.13 shows their respective signatures.



(a) Tech. Gear 24

(b) Tech. Gear 24C

(c) Tech. Gear 20DB

(d) Tech. Beam Arm

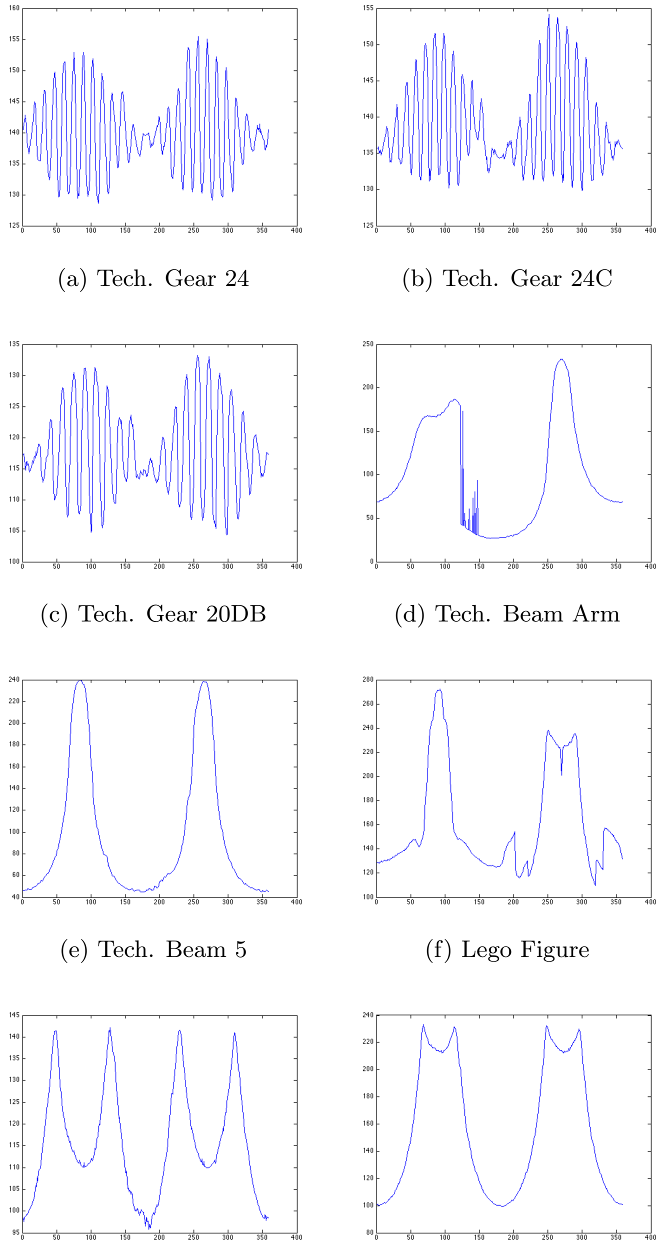
(e) Tech. Beam 5

(f) Lego Figure

(g) 2 × 2 Brick

(h) 2 × 4 Brick

Figure 5.12: The result from algorithm 6 on each object in figure 5.11. The object of interest in each image is outlined.



(a) Tech. Gear 24

(b) Tech. Gear 24C

(c) Tech. Gear 20DB

(d) Tech. Beam Arm

(e) Tech. Beam 5

(f) Lego Figure

(g) 2 × 2 Brick

(h) 2 × 4 Brick

Figure 5.13: Computed signatures for each object from the test set in figure 5.11.

The borders in figure 5.13a, 5.13b and 5.13c, upper and lower part, are diffuse and does not contain all the information about the objects real contour.

This system was also used when the data from the Technical Gear 24 (class 1) and the Technical Gear 24C (class 2) was gathered. In all, 140 feature vectors ($1 \times 4$) from each object was gathered. This data is used in the next Chapter to train and test three different classifiers.
When the data was collected the system had 7 incidents of false positive detection due to reflections on the conveyor belt. These incidents gave a failure rate of 2.5% which is close to the failure rate of 1% found in section 4.5.1.

# Chapter 6

# Feature Classification and Object Recognition

In this Chapter all experiments and results from the classification problem are presented. Data are logged from the physical objects, Technic Gear 24T and Technic Gear 24TC, and are labelled as class 1 and 2. This data is stored in a `.mat`-file called `Data`.
The conducted experiments examines three classifiers based on different techniques; Maximum Likelihood (ML) estimation, Parzen Window and $k_n$-Nearest-Neighbor (kNN) technique. The training technique that is used is called *Cross Validation* and this is presented in section 6.2.
From the collected data, different combinations are used to train the classifiers. Raw and normalized data with three different dimensions (2-, 3- and 4-dimension) of data are compared.

## 6.1 Data Sets

For both objects that are classified, it is done an experiment where the feature vector was logged 140 times. This produced two $4 \times 140$ vectors containing data for both objects/classes (`Data`).
For each class the data is randomly picked and divided into four datasets of equal size, $d \times 35$ $(0 < d \leq 4)$. Each dataset (D1 for class 1 and D2 for class 2) contains different data and a single data point do not occur in two or more datasets. The way the data is randomly picked and divided into four different datasets is done in MATLAB with the help of two functions called `randn` and `sort`. The function `randn` generates 140 different numbers and the function `sort` sorts the numbers in ascending order and stores a vector containing the original index of each sorted number. This vector is then used to pick 35 random data points from a class and produce four dataset. Figure 6.1 shows a schematic layout of the division of datasets.

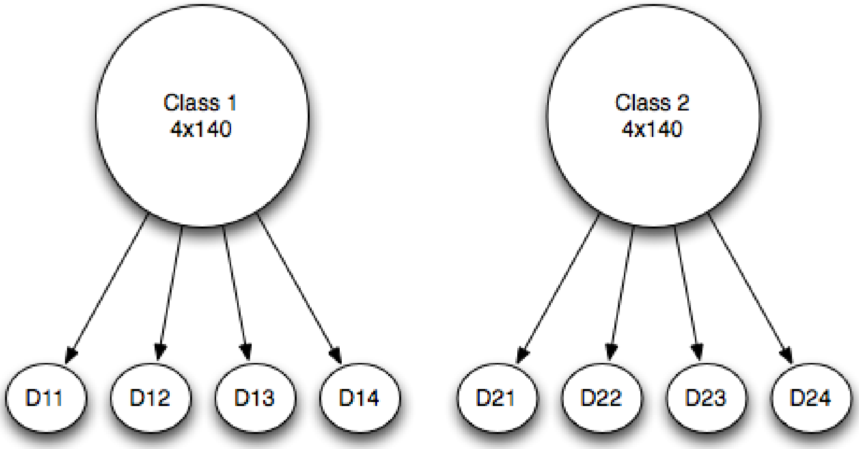


Figure 6.1: The figure shows an illustration of how the data from class 1 and 2 are divided into 8 different datasets. The size of each dataset is $d \times 35$.

As mentioned before, class 1 and 2 contains four different features extracted from the OOI, and these can be combined in many combinations when generating the randomly picked datasets. Table 6.1 shows different combinations for 2-, 3- and 4-dimensional data.

| Dimensionality (d) | Combinations |
|---|---|
| 2D | Area, Mean<br>Area, Std<br>Area, NoP<br>Mean, Std<br>Mean, NoP<br>Std, NoP |
| 3D | Area, Mean, Std<br>Area, Mean, NoP<br>Area, Std, NoP<br>Mean, Std, NoP |
| 4D | Area, Mean, Std, NoP |

Table 6.1: Combination of different features extracted from the OOI.

This report only focuses on one combination for each dimension to determine the best classifier. The combinations are:

1) Std, NoP (2D)
2) Mean, Std, NoP (3D)
3) Area, Mean, Std, NoP (4D)

The 4-dimensional data containing all the features extracted from the OOI was a natural choice to do experiments on. Since the 2D data contains *Std* and *NoP* it was important that these two features also was a part of the 3D data. All three data sets contain common data and are more valid to compare.
Because of limited time, it was not possible to conduct an experiment on each combination.

### 6.1.1 Determining Test Sets and Training Sets

After class 1 and 2 are divided into eight different datasets, like in figure 6.1, the next step is to define which dataset is for training and which is for testing. The way this has been done is to define four different training sets and four different test sets of the four data sets for each class. The four test sets are like the four data sets; D11, D12, D13 and D14 for class 1, and D21, D22, D23 and D24 for class 2. The four trainings sets (TS1 for class 1 and TS2 for class 2) then become a combination of the other three data sets. A overview of the combinations are listed in table 6.2.

| **Class** | **Test set** | **Size (test)** | **Training set** | **Size (train.)** |
|---|---|---|---|---|
| 1 | D11 | $d \times 35$ | { D12, D13, D14 } | $d \times 105$ |
| 1 | D12 | $d \times 35$ | { D11, D13, D14 } | $d \times 105$ |
| 1 | D13 | $d \times 35$ | { D11, D12, D14 } | $d \times 105$ |
| 1 | D14 | $d \times 35$ | { D11, D12, D13 } | $d \times 105$ |
| 2 | D21 | $d \times 35$ | { D22, D23, D24 } | $d \times 105$ |
| 2 | D22 | $d \times 35$ | { D21, D23, D24 } | $d \times 105$ |
| 2 | D23 | $d \times 35$ | { D21, D22, D24 } | $d \times 105$ |
| 2 | D24 | $d \times 35$ | { D21, D22, D23 } | $d \times 105$ |

Table 6.2: An overview of test and training sets for class 1 and 2.

Each training set has three times more data than each test set and this achieves the criteria given in Chapter 9 in [3], where the portion of data that is declared as a test set should be less then half of all data. This criterion is mainly used for a method called *Cross validation*. This method will be introduced in section 6.2.

### 6.1.2 Normalization of Data

The values of the raw data (features) in the feature vector lies within different ranges/scales in both classes. The range for each class is listed below.

- Area: 50000 - 70000
- Mean: 129 - 150
- Std: 4 - 9
- NoP: 14 - 26

According to [12] a feature with a larger range, or scale/value, will contribute more than a feature that's in a smaller range. This will suppress the smaller

valued features and make them less important when creating a classifier and classifying the feature vector.
By normalizing the data, the contribution from each feature are equalized [12]. Normalization achieves equal scales among different data with zero mean and unit variance. There are several different ways to normalize data, but this report only focuses on the common normalization method. The mathematical expression for this normalization is shown below

$$\underline{\hat{X}}_k = \frac{\underline{X}_k - \underline{\mu}_k}{\underline{\sigma}_k} \tag{6.1}$$

, where $\underline{\hat{X}}_k$ is the normalized data, $\underline{X}_k$ is the raw data, $\underline{\mu}_k$ is the calculated expectation from the raw data and $\underline{\sigma}_k$ is the calculated standard deviation from the raw data. 68% of all values in the normalized data will lie within the [-1, 1] range [12].

### Example of normalization for a test set and it's corresponding training set

From table 6.2, the test set D11 will be classified to {D12, D13, D14} (TS11) and {D22, D23, D24} (TS21). The mean (M) and standard deviation (S) is computed from TS11 and TS21 in MATLAB. M and S are used to normalize TS11 and TS21, and the training set D11. The MATLAB code below shows how this is done.

```
% Test set.
D11;

% Training set
TS11 = {D12 D13 D14};
TS21 = {D22 D23 D24};

T = [TS11 TS21];

% Compute M and S.
M = mean(T,2);
S = (std(T'))';

% Normalize training and test sets.
TS11 = (TS11-repmat(M,1,105))./repmat(S,1,105);
TS21 = (TS21-repmat(M,1,105))./repmat(S,1,105);
D11 = (D11-repmat(M,1,35))./repmat(S,1,35);
```

Both raw and normalized data will compared for the three classifiers based on Maximum Likelihood, Parzen and $k_n$-Nearest-Neighbor techniques.

## 6.2 Cross Validation

Cross validation is an technique that is introduced in Chapter 9 in [3]. First the dataset containing the two labelled classes is randomly divided into two parts. The first part is used for training and adjusting model parameters in the classifier and the second part (test set), also known as the validation set, is used to estimate the error rate of the classifier. Also the data in the training set is classified and the error rate from this reclassification is compared to the error rate from the validation set. The purpose is to train the classifier until the error rate from the validation set is low and the generalization between the validation set and training set is good. Good generalization, in this report, is defined as the difference between the validation and training error, and this difference should not exceed 2%.
Figure 6.2 shows an example of cross validation.

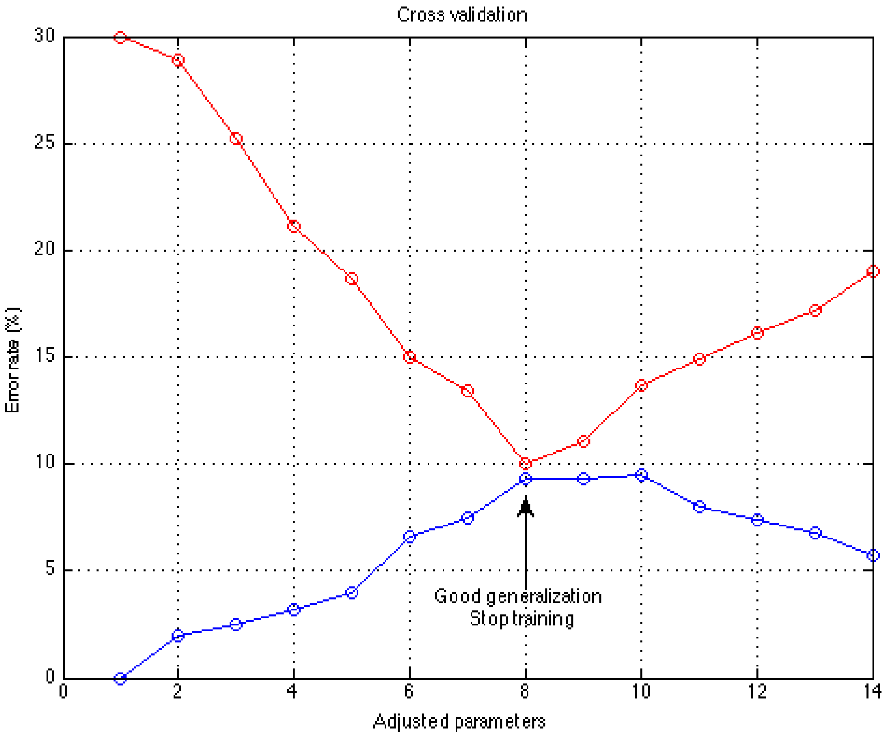


Figure 6.2: An example of cross validation. The arrow in the figure points to the best generalization between the validation and training error, at a given parameter.

The behaviour of the training and validation error varies from problem to problem. A thumb rule is to stop the training of the classifier where a good generalization is achieved at the lowest validation error before it starts increasing again. In figure 6.2, after the indication of good generalization, the validation error increases and this can indicate that the classifier is overtrained [3].

According to [3], cross validation can be applied to almost every classification method. An example can the Parzen Window technique where the adjusting parameter that trains the classifier is the window width parameter, h1.
This method will be applied on each of the three classifier with raw and normalized data to determine the best classifier. Except from only using good generalization as a criterion for choosing the best classifier, some other criterias have to be emphasized as well. In section 6.2.2 all criterias will be

listed and explained. The next section will explain how the validation and training errors are found, and how they are calculated.

### 6.2.1 Finding the Validation and Training Error

Each of the three classifiers are implemented in MATLAB as functions with inputs and outputs.

For the ML and Parzen Window classifier the inputs are the test sets and training sets for class 1 and 2. The training sets are used to generate the classifier, but they are also used for reclassification. Each test set and training set (for class 1 and 2) are then classified individually. For each set thats being classified, the classifier computes the discriminant value for every datapoint in the set for class 1 and 2. The discriminant value for each class is then compared against each other, e. q. the first discriminant value for class 1 is compared to the first discriminant value in class 2, and so on. The result is a $1 \times N$ vector containing the integers 1 or 2, where $N$ is the length of the set thats being classified ($N = 35$ for test sets and $N = 105$ for training sets).

If the integer is 1, the discriminant value for a datapoint is bigger for class 1 than class 2, and the point is classified to class 1. Opposite if the integer is 2. If a random set belongs to class 1 (f. ex. D11), the number of wrong classifications (NOWC) can be found by adding up each element in the $1 \times N$ vector which belongs to class 2 (each element that contains 2). Then the number of correct classifications (NOCC) is $N - NOWC$. An example for 2D data is shown in figure 6.3.

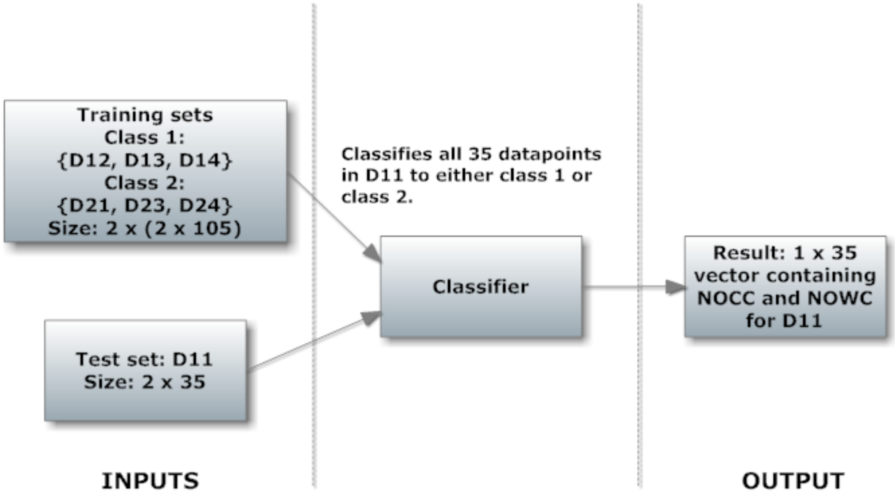


Figure 6.3: Finding NOCC and NOWC for random test set. D11 is classified to both training sets for class 1 and 2. The result is a $1 \times 35$ vector containing 1s and 2s. The number 1 indicate that a sample in D11 is classified to class 1, and the number 2 indicate that a sample is classified to class 2.

The other classifications for this example is shown below.

| **Classification set** | **Training set (generates the classifier)** | $N$ |
|---|---|---|
| D21 | {D12, D13, D14} & {D22, D23, D24} | 35 |
| {D12, D13, D14} | {D12, D13, D14} & {D22, D23, D24} | 105 |
| {D22, D23, D24} | {D12, D13, D14} & {D22, D23, D24} | 105 |

Table 6.3: Classifications for random 2D test and training sets.

The result is four different $1 \times N$ vectors containing the NOCC and NOWC for each set. To compute the validation and training error a matrix called the *Confusion Matrix* is used. The confusion matrix is used to evaluate the performance of a classification system [17]. The confussion matrix is defined like this,

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \tag{6.2}$$

where

- $A_{11}$ is the NOCC for class 1, given class 1
- $A_{12}$ is the NOWC for class 1, given class 2
- $A_{21}$ is the NOWC for class 2, given class 1
- $A_{22}$ is the NOCC for class 2, given class 2

The correct classification rate for the classifier, also known as *Overall Accuracy (Ac)* [17], is calculated as

$$Ac = \frac{A_{11} + A_{22}}{A_{11} + A_{12} + A_{21} + A_{22}} \tag{6.3}$$

and the test and/or training error is then calculated as

$$E = 1 - Ac \tag{6.4}$$

The error is calculated for every validation and training set (four times), at each adjusted parameter, and the mean of these four errors is the mean validation and mean training error, which is compared in the cross validation. Appendix C.2 gives an example of calculating these errors from a set of data.

### 6.2.2 Criterias for Choosing the Best Classifier

There are three criterias that have to be achieved before determining which classifier that is best. The criterias are

1) Good generalization (max 2%)
2) Low mean error rate
3) Low spread in data (low standard deviation)

**Good generalization**
As mentioned before, good generalization should not exceed 2%. For every classifier with raw and normalized data a plot of the validation and training error will be produced, like in figure 6.2. In every plot there may be several points where the generalization is equal or lower than 2%. Each point refers to a adjusted parameter in the classifier and this parameter is different for each classifier.

*ML Classifier*
For the ML classifier the parameter is the covariance matrix. Adjusting this parameter is done by adjusting the elements in the matrix. The covariance matrix is calculated from the training data and it will be of size $2 \times 2$, $3 \times 3$ or $4 \times 4$ depending on the dimensionality of the actual data (2D, 3D or 4D), and this is the first parameter used in the cross validation. In the second parameter the non diagonal elements are set equal to zero and in the third parameter the covariance matrix is a unit matrix. Below is an example of the three different parameters for 2D data.

$$\sum = \begin{pmatrix} \sigma_{11}^2 & \sigma_{12}^2 \\ \sigma_{21}^2 & \sigma_{22}^2 \end{pmatrix} \quad \sum = \begin{pmatrix} \sigma_{11}^2 & 0 \\ 0 & \sigma_{22}^2 \end{pmatrix} \quad \sum = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

General Diagonal Unit

*Parzen Window Classifier*
For the Parzen Window classifier the window width parameter, h1, of the Gaussian window is the parameter that will be adjusted. For each cross validation the test starts with a very low value of h1, which produces a narrow window, and often gives 100% correct classifications for the reclassification of the training set. This implies that the classifier is overtrained for the training set. The parameter is adjusted until good generalization is achieved.

*$k_n$-Nearest-Neighbor Classifier*
For the $\mathrm{k_n}$-Nearest-Neighbor classifier the $\mathrm{k_n}$ parameter is adjusted. This parameter can only be a integer and the lowest value possible is 1.

**Low mean error rate**
The difference between the mean validation error and the mean training error decides if the generalization is good or bad. In the points where this difference is equal or lower than 2%, the point (adjusted parameter) which has the lowest mean validation error should be picked as the best solution. This error is the lowest error rate for the classifier when good generalization is achieved, but this choice may be affected by the next criteria.

**Low spread in data**
From the four different validation errors (both class 1 and 2) at a parameter, the mean validation error is calculated. These four values can also be used to calculated the standard deviation at the parameter. If the value of the

standard deviation is high the spread in data is also high and more uncertain. This can be examined by plotting a 95% confidence interval for the mean validation error. This confidence interval gives a 95% probability that the interval contains the mean validation error [6]. This interval is plotted by using the following mathematical expression

$$\bar{E} \pm 1.96SD(E) \tag{6.5}$$

, where $\bar{E}$ is the mean validation error and $SD(E)$ is the standard deviation for this point. The number 1.96 is a key factor for calculating a 95% confidence interval [6] (table for normal distribution - $N(0,1)$).

If the value of the standard deviation is low, the interval is smaller and the spread in data is less, and this gives an indication of more certain data. If this is taken into account when choosing the lowest error rate, it can affect this choice. Consider two error rates within good generalization. One mean error rate is bigger than the other, and the highest valued error rate has the lowest standard deviation. Since the data is more certain (less spread) for the higher valued mean validation error, this one will be picked.
Every cross validation plot for each classifier will also contain a plot of the 95% confidence interval for the mean validation error.

## 6.3 Comparing Cross Validations & Gather Data

In this experiment, all three classifiers are analyzed with cross validation on raw and normalized 2D, 3D and 4D data. The purpose is to find the best possible classifier with good generalization, low error rate (mean validation error rate) and low spread in data for each classifier. In this section all cross validations of each classifier is presented with plots. In the each plot the 95% confidence interval is also plotted (marked as - - -).
All the data that has been collected from each classifier experiment is presented in appendix F.

### 6.3.1 Maximum Likelihood Estimation

In this section, all cross validations for the ML estimation are presented with plots. Also a table is presented in the end which lists up the criterias for the best classifier (best found parameter) for each type of data.

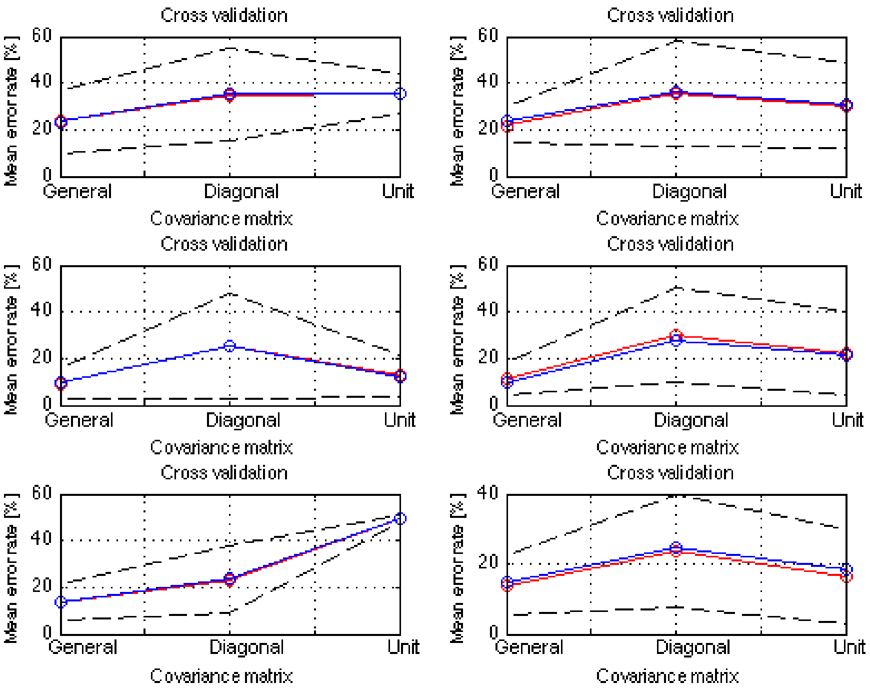


Figure 6.4: Cross validation for the ML classifier with 2D, 3D and 4D data. Each row shows the cross validation for 2D, 3D and 4D data. Each column shows the cross validation for raw and normalized data. The red graph represents the validation, the blue graph represents the training and the black dotted lines represent the confidence interval.

The table below lists up the best classifier from each ML experiment.

| **Data** | **Best parameter (cov. matrix)** | **Mean val. error (%)** | **Data spread (%)** | **Generalization (%)** |
|---|---|---|---|---|
| 2D | Unit | 35.36 | 44.02 - 26.79 | 0.12 |
| N 2D | Unit | 30.36 | 48.57 - 12.15 | 0.47 |
| 3D | General | 9.29 | 15.95 - 2.63 | 0.36 |
| N 3D | General | 11.43 | 18.29 - 4.57 | 1.43 |
| 4D | General | 13.57 | 21.65 - 5.50 | 0.12 |
| N 4D | General | 13.93 | 22.59 - 5.27 | 0.95 |

Table 6.4: Gathered data from each ML classifier experiment. The data is gathered from the best adjusted parameter in each cross validation.

### 6.3.2 Parzen Window Technique

In this section, all cross validations for the Parzen Window technique are presented with plots. Also a table is presented in the end which lists up the criterias for the best classifier (best found parameter) for each type of data.

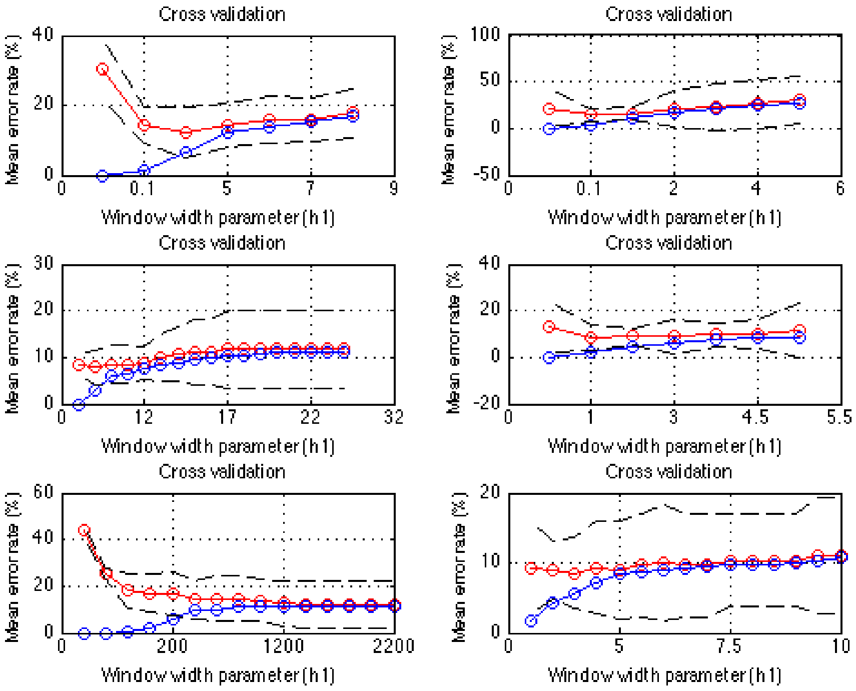


Figure 6.5: Cross validation for the Parzen Window classifier with 2D, 3D and 4D data. Each row shows the cross validation for 2D, 3D and 4D data. Each column shows the cross validation for raw and normalized data. The red graph represents the validation, the blue graph represents the training and the black dotted lines represent the confidence interval.

The table below lists up the best classifier from each Parzen Window experiment.

| **Data** | **Best parameter (h1)** | **Mean val. error (%)** | **Data spread (%)** | **Generalization (%)** |
|---|---|---|---|---|
| 2D | 7 | 16.07 | 22.28 - 9.86 | 0.59 |
| N 2D | 3 | 22.86 | 48.10 - (-2.38) | 1.79 |
| 3D | 12 | 8.93 | 12.46 - 5.4 | 1.19 |
| N 3D | 4 | 9.64 | 14.42 - 4.86 | 1.78 |
| 4D | 2000 | 12.14 | 22.49 - 1.79 | 0.59 |
| N 4D | 9 | 10.36 | 16.98 - 3.74 | 1.55 |

Table 6.5: Gathered data from each Parzen Window classifier experiment. The data is gathered from the best adjusted parameter in each cross validation.

### 6.3.3 $\mathbf{k_n}$-Nearest-Neighbor

In this section, all cross validations for the kNN technique are presented with plots. Also a table is presented in the end which lists up the criterias for the best classifier (best found parameter) for each type of data.

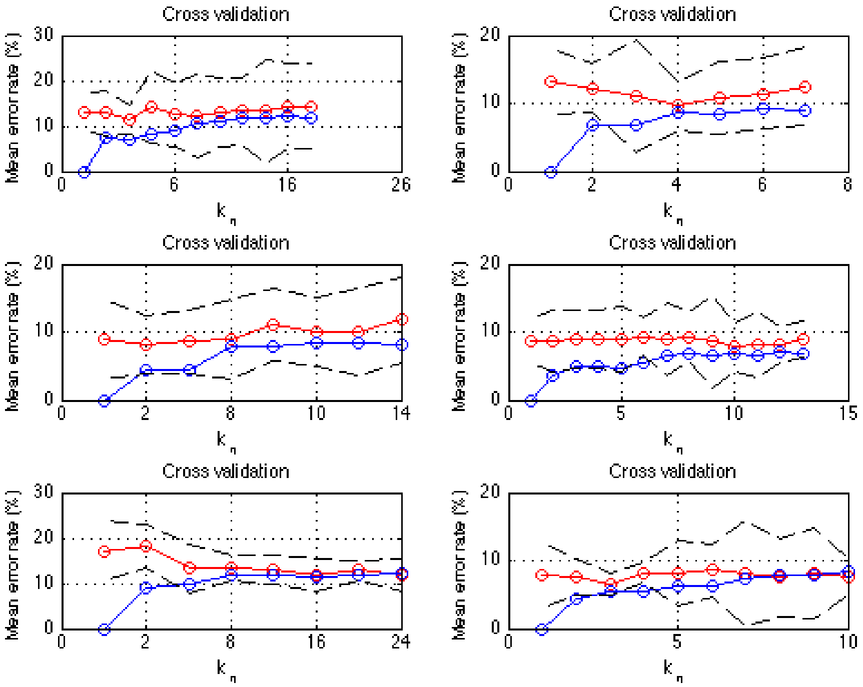


Figure 6.6: Cross validation for the kNN classifier with 2D, 3D and 4D data. Each row shows the cross validation for 2D, 3D and 4D data. Each column shows the cross validation for raw and normalized data. The red graph represents the validation, the blue graph represents the training and the black dotted lines represent the confidence interval.

The table below lists up the best classifier from each kNN experiment.

| Data | Best parameter ($k_n$) | Mean val. error (%) | Data spread (%) | Generalization (%) |
|---|---|---|---|---|
| 2D | 12 | 13.21 | 20.58 - 5.84 | 1.54 |
| N 2D | 4 | 9.64 | 13.17 - 6.11 | 0.95 |
| 3D | 10 | 10.00 | 15.12 - 4.88 | 1.67 |
| N 3D | 12 | 8.21 | 10.90 - 4.82 | 1.19 |
| 4D | 20 | 12.86 | 15.15 - 10.57 | 0.96 |
| N 4D | 3 | 6.43 | 8.04 - 4.82 | 0.95 |

Table 6.6: Gathered data from each kNN classifier experiment. The data is gathered from the best adjusted parameter in each cross validation.

## 6.4 Conclusion

Based on the given criterias, presented in section 6.2.2, the best classifier for each method is picked out and in the end compared against each other.

The data for the ML classifier in table 6.4 gives an indication that the classifier for the raw 3D data, with a general covariance matrix, is the best solution for this experiment. It has the lowest mean validation error and least spread in data, which indicates less uncertainty in the data.

In the case, with the Parzen Window classifier data in table 6.5, the data suggests that the classifier with raw 3D data (h1 = 12) is the best solution. Also, in this case, the classifier for the raw 3D data have the lowest mean validation error and least spread in data. For every experiment done with normalized data, the window width parameter, h1, is lower than for the experiments done with raw data.

In the last case, with the kNN experiments, the classifier for the normalized 4D data stands out from the rest ($k_n = 3$). Of the three criterias mentioned in section 6.2.2, the kNN classifier for the normalized 4D data has the best generalization, the lowest mean validation error and least uncertain data, compared to all other experiments done with kNN classifiers.

When comparing and choosing the best classifier from the best result in each method, the same criterias are used as before. The ML classifier has the highest mean validation error and the most uncertainty in data compared to the Parzen and kNN classifiers. Even if this is the simplest method to implement and the less time consuming algorithm, it will not be chosen.
The kNN classifier has lower mean validation error, better generalization and least spread in data, compared to the Parzen classifier for the raw 3D data. All of the three criterias are better for the kNN classifier than for the Parzen, and the kNN classifier will be implemented in the system.

Since the kNN classifier with normalized 4D data gave the best result, the `ooi()` algorithm will not be changed, and will still find four features from an object.

## 6.5 Development of the kNN Classifier

The kNN classifier is implemented in MATLAB on the bases of algorithm 7.

**Algorithm 7** kNN Classifier for 4D features

```
Input: D, $\underline{x}$, $k_n$, C
for $i = 1 \rightarrow$ length(D) do
    $\delta = \underline{x} - D(:, i)$
    $R(1, i) = \delta^T \times \delta$
end for
$[junk\ idx] = sort(R)$
$g(1,1) = sum(C(1, idx(1 : kn)) == 1)/length(find(R \leq R(1, idx(kn))))$
$g(1,2) = sum(C(1, idx(1 : kn)) == 2)/length(find(R \leq R(1, idx(kn))))$
$[junkCx] = max(g)$
return $Cx$
```

**Explanations of variables:**

- $D$ is the collected data from class 1 and 2. $D$'s size is $4 \times 280$, where the first 140 samples belong to class 1, and the last 140 samples belongs to class 2.

- $\underline{x}$ is the feature vector.
- $k_n$ is the nearest neighbor parameter.
- $C$ is a vector that contains the labels for $D$. The first 140 indices are 1 and this indicates to the program that the first 140 samples in $D$ belongs to class one. The last 140 indices in $C$ are 2.
- $\delta$ is the distance between each feature in $\underline{x}$ and in $D$. This distance is computed for every sample in $D$ to $\underline{x}$.
- $R$ contains the Euclidian distance between $\underline{x}$ and every sample in $D$.
- $sort(R)$ sorts the Euclidean distances in ascending order. $idx$ contain the original indices to each of the sorted distances.
- $g(1,1)$ and $g(1,2)$ computes the a posteriori probability for class 1 and for class 2.
- $Cx$ is 1 if $g(1,1) \geq g(1,2)$ or 2 if $g(1,2) > g(1,1)$.

$R$ contains each Euclidean distance from $\underline{x}$ to every sample in $D$. The order of the distances in $R$ is the same as for every sample in $D$. This means that the Euclidean distance for the first sample in $D$ is stored in the first index in $R$, and so on. When $R$ is sorted in ascending order, the vector $idx$ contains the original index of each distance in $R$.

$idx(1 : k_n)$ finds the original indices to the $k_n$-nearest neighbors and $sum(C(1, idx(1 : k_n)) == 1)$ computes which of the $k_n$-nearest neighbors that belongs to class 1. $sum(C(1, idx(1 : k_n)) == 2)$ computes which of the $k_n$-nearest neighbors that belongs to class 2.

$length(find(R \leq R(1, idx(kn))))$ finds the total amount of $k_n$-nearest neighbors to $\underline{x}$.

The a posteriori probability for class 1 and for class 2 is computed by using the equation 2.47, presented in section 2.5.5.2.

The MATLAB implementation of algorithm 7 can be found in appendix E.7.

The MATLAB implementation of the ML and Parzen Window classifier, used in the experiments, are shown in appendix E.10. Since the Parzen Window classifier is based on a Gaussian window, the ML classifier are implemented in the Parzen as well.

# Chapter 7

# Explanation of System Functionalities

In this Chapter, the functions `MOD()`, `ooi()` and `kNN_classifier` are implemented in `main.m`. `main.m` is the main program that runs the whole system. A part from the mentioned functions, other functionalities will be implemented as well. Section 7.1 presents `main.m` and it's functionalities.

The other sections presents all the functions that is included in `main.m`. Each function/program are presented with flowcharts. Each step in all the flowcharts are numbered and explained.

## 7.1 Main Function - main()

As mentioned the main program `main.m` runs the system and contains all the functions that are developed. This section presents `main.m` and explains it's structure.

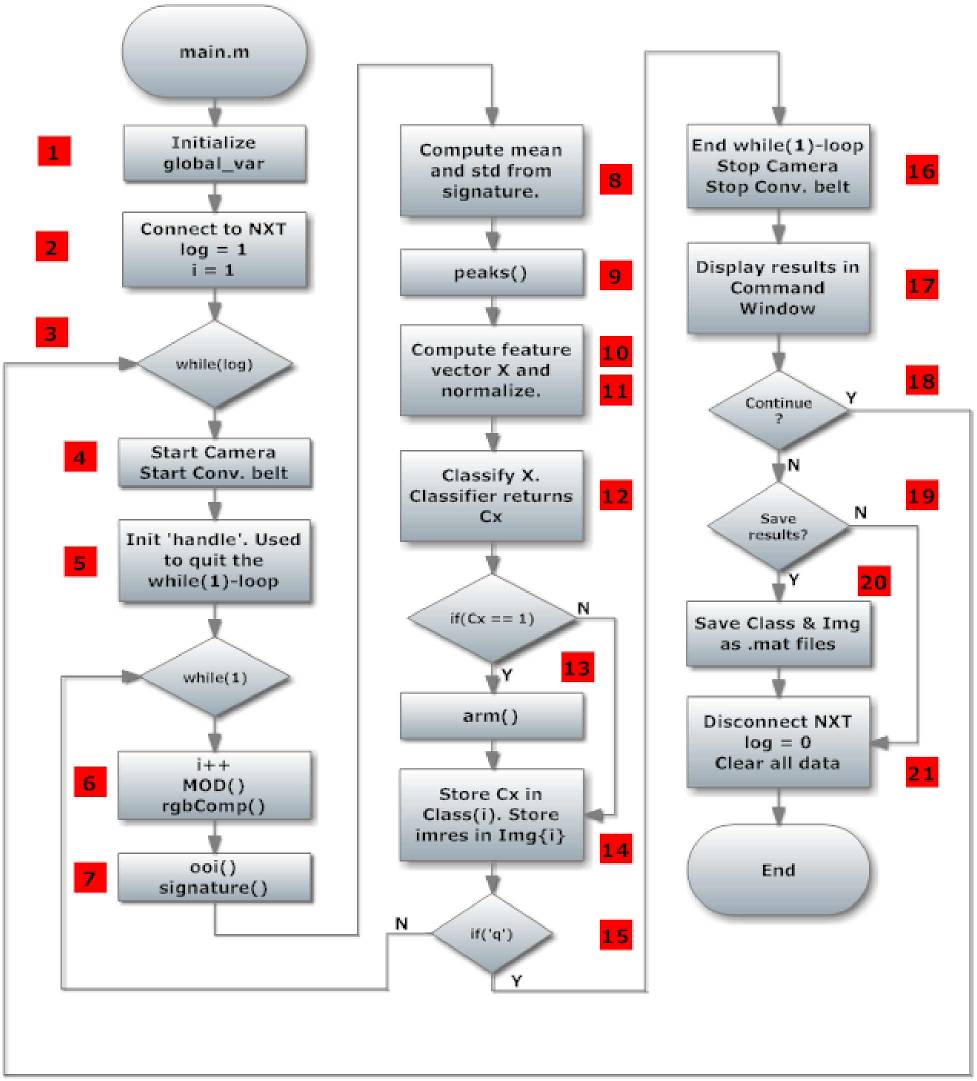

main.m
1
Initialize global_var
2
Connect to NXT log = 1 i = 1
3
while(log)
4
Start Camera Start Conv. belt
5
Init 'handle'. Used to quit the while(1)-loop
while(1)
6
i++ MOD() rgbComp()
7
ooi() signature()
8
Compute mean and std from signature.
9
peaks()
10
11
Compute feature vector X and normalize.
12
Classify X. Classifier returns Cx
if(Cx == 1)
N
Y
13
arm()
Store Cx in Class(i). Store imres in Img{i}
14
if('q')
N
Y
15
16
End while(1)-loop Stop Camera Stop Conv. belt
17
Display results in Command Window
18
Continue ?
Y
N
19
Save results?
N
Y
20
Save Class & Img as .mat files
Disconnect NXT log = 0 Clear all data
21
End

Figure 7.1: Flowchart for `main.m`.

| Step | Elaboration |
|---|---|
| 1 | `global_var` initializes global variables that the functions use. It creates a video input object and connects the web camera to MATLAB. It also creates NXT motor objects for the conveyor belt and the sorter arm. The collected data that feature vectors are classified to are also initialized and normalized in this m-file. `global_var.m` can be looked at in appendix E.9. |
| 2 | Connects to NXT. The variables `log` is set to 1 and `i` is set to 0. While `log` is 1, the system runs. The variable `i` is a help variable that is increments each time a object has been detected and classified. `Class` and `Img` uses this variable. |
| 3 | **WHILE**-loop. Runs as long as `log` equals 1. |
| 4 | The camera starts logging frames and motor object `mA` is transferred to NXT, and the conveyor belt starts moving. |
| 5 | `'handle'` is a figure that opens when it's initialized. This figure is used to exit the `while(1)`-loop. If the `'q'` button is pressed after an object is classified, MATLAB interprets this as closing the figure and MATLAB jumps out of the `while(1)`-loop by executing the `break` command. |
| 6 | The `MOD()` function runs until a object is detected on the conveyor belt. The `rgbComp()` function converts the RGB frame captured by `MOD()` to a binary image. It computes the threshold from every RGB component and converts the component having the highest threshold to a binary image. |
| 7 | The `ooi()` function finds and extracts the object of interest (OOI) in the binary image received from `rgbComp()`. It also finds some of the objects internal (area, centroid) and external (border and border coordinates) characteristics. The function returns an binary image containing only the OOI and the internal and external data from the object. The `signature()` function computes the signature of the objects border using the objects centroid and border coordinates. |
| 8 | The mean (`MSign`) and standard deviation (`StdSign`) of the signature is computed with the MATLAB commands `mean` and `std`. |
| 9 | The `peaks()` functions computes the amount of peaks/vertices (`NoP`) in the signature. |
| 10 | The feature vector for the OOI, `X`, is initialized. The features are the OOIs area, `MSign`, `StdSign` and `NoP`. |

| | |
|---|---|
| 11 | X is normalized by using the mean and standard deviation computed from the collected data (line 80-81 in E.9). |
| 12 | X is classified with $k_n$-nearest-neighbor technique (kNN_classifier()). The classifier returns Cx to main. Cx is 1 if the OOI belongs to class 1. Else it's 2 and the OOI belongs to class 2. |
| 13 | **IF**-loop. If Cx equals 1, the arm is moved across the conveyor belt and the object is pushed off the belt and into a tray. |
| 14 | It saves the i-th OOIs class number and the RGB frame containing the OOI in Class(i) and in Img{i}. |
| 15 | **IF**-loop. Checks if 'q' is pressed. If it is not pressed, the while(1)-loop restarts. |
| 16 | 'q' is pressed. The camera stops logging and the conveyor belt stops. |
| 17 | Displays the number of objects that have passed through the system and how many objects that are classified to class 1 and class 2. This is presented in MATLABs command window. |
| 18 | The program then asks if the user wants to continue and restart the system without deleting i, Class(i) and Img{i}. The user needs to enter Y or N. If the users wants to continue, the program starts over and jumps back to step 4. |
| 19 | User doesn't want to continue. The program ask if the user wants to save Class(i) and Img{i}. |
| 20 | If the user wants to save the results, they are saved in the current MATLAB workspace as two separate .mat files. The current date and time are added to the filenames. |
| 21 | The program disconnects from NXT and clears all variables in the workspace. |

## 7.2 Connect to NXT - `con2NXT()`

This function opens a bluetooth connection to the NXT. If the connection fails it will try the USB connection, if the cable is connected.

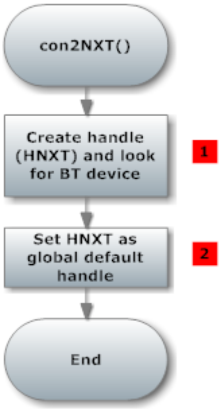

con2NXT()
Create handle (HNXT) and look for BT device
1
Set HNXT as global default handle
2
End

Figure 7.2: Flowchart for `con2NXT()`.

| Step | Elaboration |
|---|---|
| 1 | Looks for and connects to the NXT. In the process it creates the handle HNXT. |
| 2 | Sets HNXT to a global handle. HNXT is used by all the RWTH functions to communicate with the NXT. |

## 7.3 Moving Object Detection - MOD()

This functions detects an object on the moving conveyor belt and returns the image containing the whole object within the FOV.

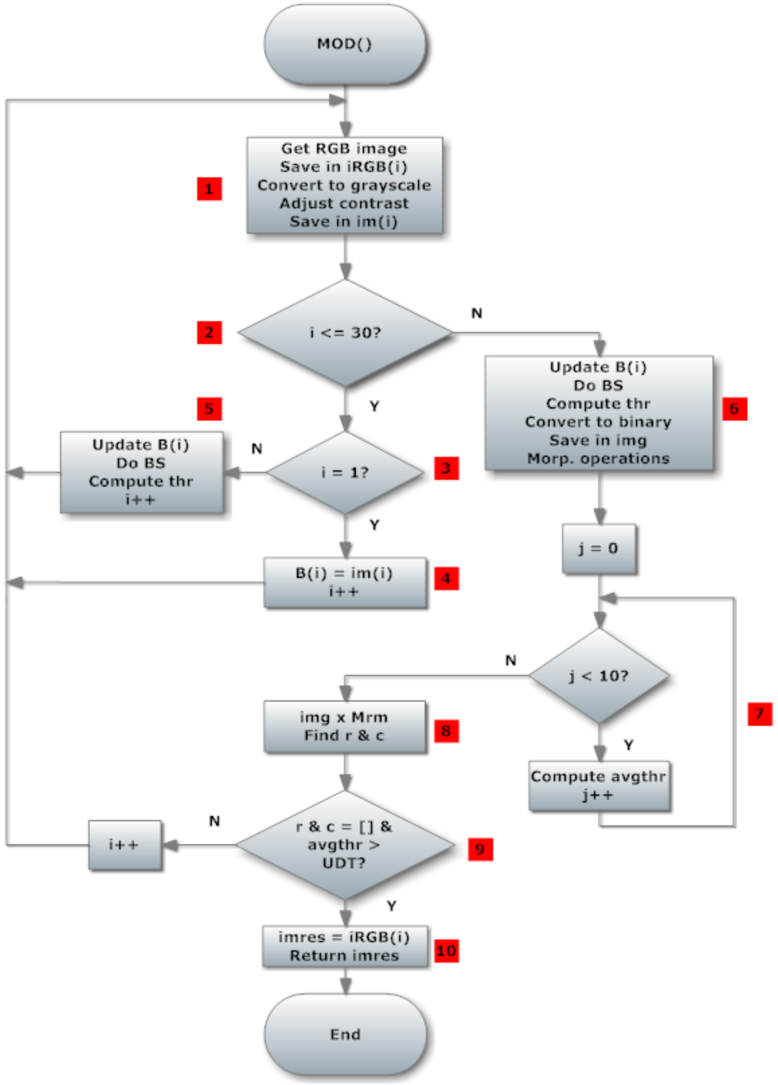


Figure 7.3: Flowchart for MOD().

| Step | Elaboration |
|---|---|
| 1 | First the camera takes a snapshot of the conveyor belt and saves the image in `iRGB(i)`. Then it converts the image to grayscale, adjusts the contrast and saves this version in `im(i)`. |
| 2 | **IF**-loop. Goes through this loop 30 times. |
| 3 | Another **IF**-loop that only is executed one time. |
| 4 | IF *i* equals 1, the first background image `B(1)` is the the first grayscale image `im(1)`. |
| 5 | For *i* equals 2 to 30 the background model is updated using equation 4.2. Then it computes the background subtraction between `im(i)` and `B(i)`. This is done to generate a BM before starting to detect objects. For every BS the threshold is also computed using the MATLAB function `graythresh`. |
| 6 | In this segment the function starts to look for changes in the result from BS. The BM is still updated and the threshold is computed from the BS result and stored in `thr(i)`. The image computed from the BS is converted to a binary image and stored in `img(i)`. Every binary image is exposed to a morphological operation that clear objects on the image border and fill any gaps in remaining objects. |
| 7 | A **FOR**-loop that computes the average threshold for the last ten computed `thr(i)`. The result is stored in `avgthr(i-30)`. This result is used in step 9. |
| 8 | Every `img(i)` is multiplied with `Mrm`. `Mrm` is a matrix (same size as `img`) containing only zeros except in a area called RM, where the elements have value 1. If an object is present in `img(i)` and is overlapping RM, the result would be a matrix that only contains value 1 where the object overlapped RM. By using the MATLAB function `find` on this result will return a row and a col vector that contains indexes for all 1-valued pixels. If an objects doesn't overlap RM the result is a zero matrix, and row and col would be empty. |
| 9 | If row = col = [ ] (empty) the function starts over. If they are not empty and `avgthr(i-30)` is bigger than UDT, the function continues. |
| 10 | The variable `imres` is set equal to `iRGB` and is returned to main. The function then ends. |

## 7.4 Extract and Separate Object from Background - rgbComp()

This function analyses each R, G and B component in the image and determines which component the object is extracted from.

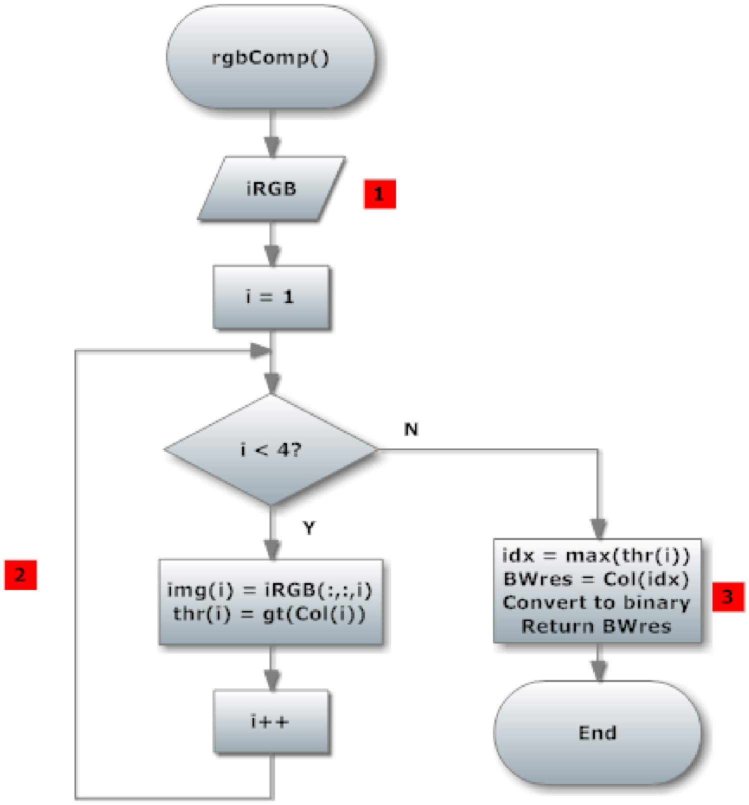


Figure 7.4: Flowchart for rgbComp().

| Step | Elaboration |
|---|---|
| 1 | **Input image:** `iRGB` received from `MOD()`. |
| 2 | **FOR**-loop. Extracts each RGB component from `iRGB` and stores them in `img(i)`. For each component the threshold value is computed using the MATLAB function `graythresh`, but each components intensity level is adjusted (`imadjust`) beforehand to make the object stand more out in all components. Each threshold is saved in the 1-by-3 vector `thr(i)`. |
| 3 | The MATLAB function `max` returns the index of the highest valued threshold in `thr(i)` and uses this to determine which component that has the best result of extracting the object from the background. The best component is converted to the binary image `BWres` and returned to main. |

## 7.5 Object of Interest - ooi()

The ooi() function finds the object of interest in a binary image and reduces noise. It then returns the object perimeter as an binary image, the object border coordinates and object data.

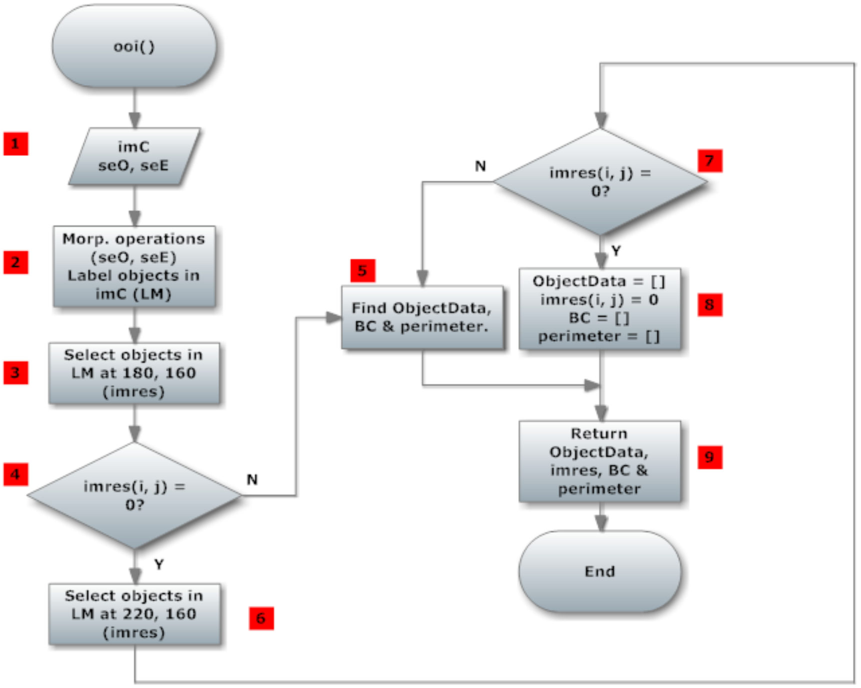


Figure 7.5: Flowchart for ooi().

| Step | Elaboration |
|---|---|
| 1 | **Inputs:** Binary image - imC and structural element for morphological opening and erosion - seO and seE. |
| 2 | Reduce noise by morphological opening and smooth the object contour by erosion. Label (result: LM) remaining objects with the MATLAB function bwlabel. |
| 3 | Select object that overlaps coordinate (n, m): 180, 160. The result is stored in imres. |
| 4 | **IF**-loop. Compares imres with a zero matrix (of the same size). |

| | |
|---|---|
| 5 | If they are not equal, the object is extracted from `LM`. Then object data is found from the object using the MATLAB function `regionprops` (area, centroid and bounding box), the perimeter is extracted from `imres` using `bwperim` and the boundary coordinates are computed using `bwboundaries`. |
| 6 | If they are equal, the object isn't extracted from `LM` and the function tries to select the object at coordinates 220, 160 (in `LM`) instead. `imres` is overwritten. |
| 7 | **IF**-loop. Compares `imres` with a zero matrix (of the same size). Not equal: See step 5. |
| 8 | Equal. No object is in `LM` and the result from `MOD()` is a false positive detection. All values are set empty / zero. |
| 9 | Returns `ObjectData`, `imres`, `BC` (boundary coordinates) and `perimeter` to main. The function then ends. |

## 7.6 Signature - signature()

Computes an objects signature and returns all distances and angles.

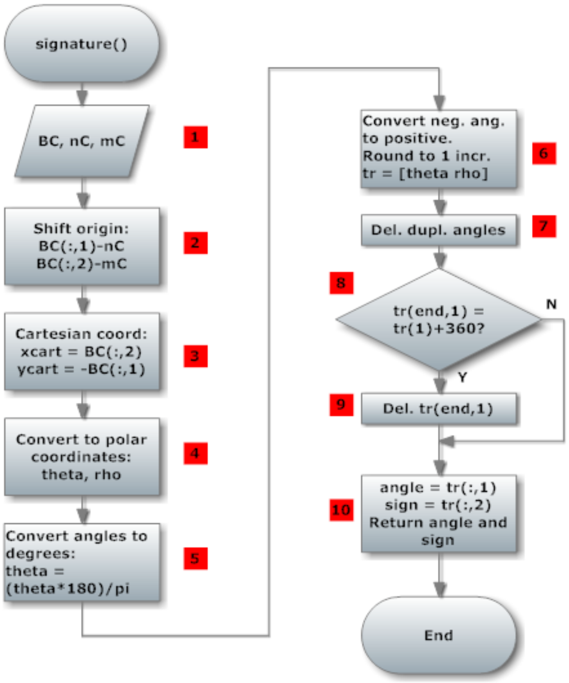


Figure 7.6: Flowchart for `signature()`.

| Step | Elaboration |
|---|---|
| 1 | **Inputs:** Boundary coordinates - `BC` (np-by-2 vector) and object centroid coordinates - `nC` and `mC`. |
| 2 | Sets `nC` and `mC` as the origin to all boundary coordinates in `BC`. |
| 3 | Converts the coordinates in `BC` to Cartesian coordinates (`xcart` and `ycart`) that MATLAB uses. |
| 4 | Converts `xcart` and `ycart` to Polar coordinates by using the MATLAB `cart2pol`. Result: `theta` (angle) and `rho` (distance). |
| 5 | Converts `theta` from radians to degrees. |
| 6 | All angles in `theta` is converted to nonnegative angles. All angles are also rounded so that there is 1 increment (degree) between them. `theta` and `rho` is stored in `tr`. |

| | |
|---|---|
| 7 | If some of the angles in `theta` is equal the duplicates are deleted. |
| 8 | **IF**-loop. Checks if the first and last angle in `theta` is equal. |
| 9 | Equal. Deletes last angle in `theta`. |
| 10 | `angle = tr(:, 1)` and `sign = tr(:, 2)`. Returns `angle` and `sign`. |

## 7.7 Detect Vertices in Signature - peaks()

The function **peaks()** analysis the signature and returns the number of peaks detected.

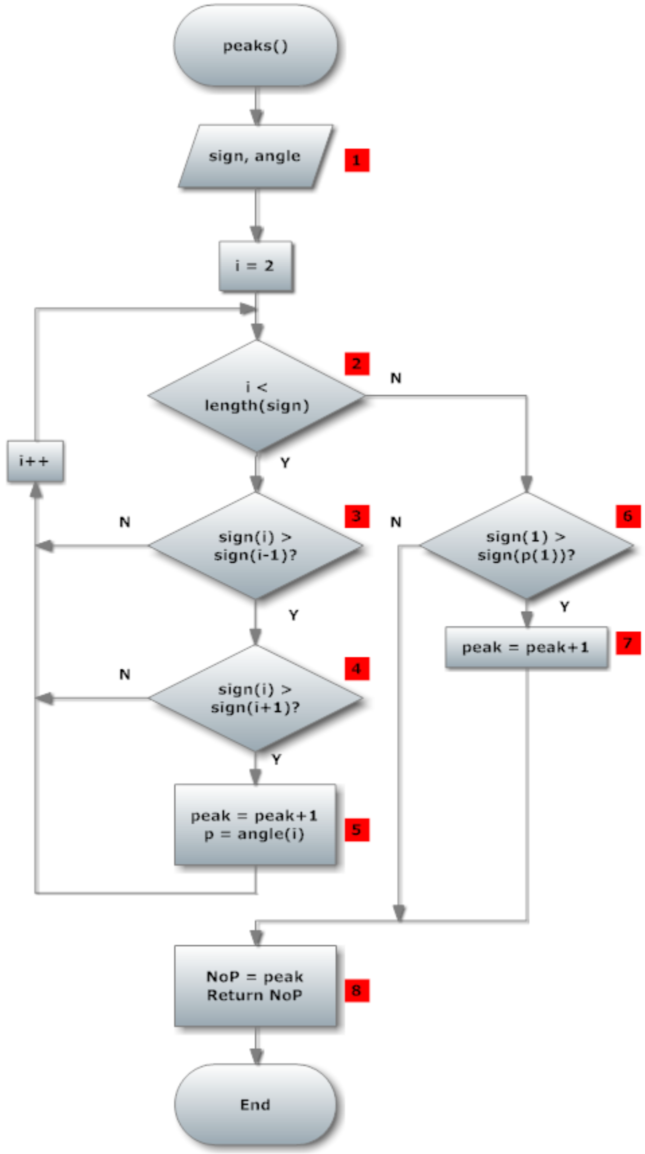


Figure 7.7: Flowchart for peaks().

| Step | Elaboration |
|---|---|
| 1 | **Inputs:** `sign` and `angle` computed by `signature()`. |
| 2 | **WHILE**-loop. Continues until all values in the `sign` vector have been analyzed. |
| 3 | **IF**-loop. Is `sign(i)` bigger than `sign(i-1)`? If not, increment *i* and start over. |
| 4 | **IF**-loop. Is `sign(i)` bigger than `sign(i+1)`? If not, increment *i* and start over. |
| 5 | `sign(i)` is bigger than both `sign(i-1)` and `sign(i+1)`. Increment peak and store the peaks position in `p(peak)`. Increment and start over if `i < length(sign)`. |
| 6 | **IF**-loop. Compares the value in the first element in `sign` with the value of the first peak. If `sign(1)` is smaller than `sign(p(1))`, jump out of the **IF**-loop. |
| 7 | `sign(1)` is larger than `sign(p(1))`. Increment `peak`. |
| 8 | `NoP` equals `peak`. Return `NoP` and end function. |

## 7.8 The kNN Classifier - `kNN_classifier()`

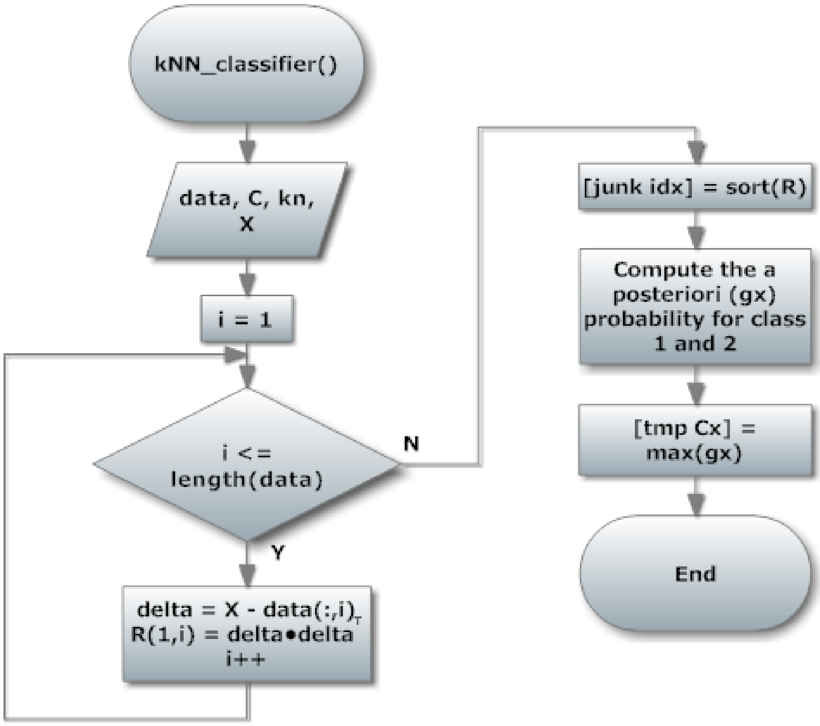


Figure 7.8: Flowchart for `kNN_classifier()`.

The explanation for this flowchart is explained in Chapter 6, section 6.5.

## 7.9 Sorter Arm - arm()

This function controls the arm that pushes objects of the conveyor belt.

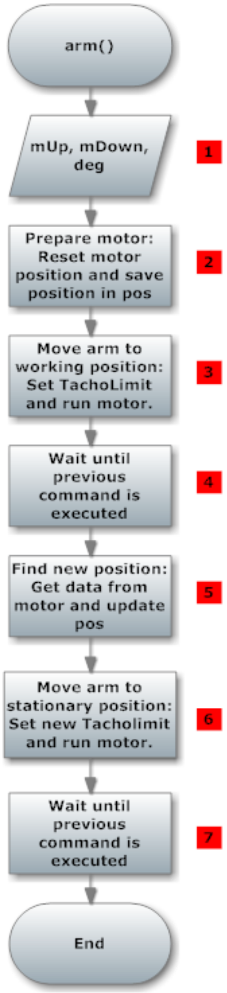

arm()
mUp, mDown, deg
1
Prepare motor: Reset motor position and save position in pos
2
Move arm to working position: Set TachoLimit and run motor.
3
Wait until previous command is executed
4
Find new position: Get data from motor and update pos
5
Move arm to stationary position: Set new Tacholimit and run motor.
6
Wait until previous command is executed
7
End

Figure 7.9: Flowchart for arm().

| Step | Elaboration |
|---|---|
| 1 | **Inputs:** `mUp` and `mDown` are motor objects created from the NXT motor B (line in 52-55 E.9). `mUp` positions the arm in stationary position and `mDown` positions the arm in working position. The variable `deg` is short for degrees and is set to 90. |
| 2 | When reseting the motor position, the current position of the motor is automatically set to 0. This value is saved in `pos`. |
| 3 | Motor object `mDown` is sent to the NXT motor to move the arm across the conveyor belt. The movement of the motor is decided by the RWTH function `TachoLimit`, which is set to `deg + pos`, in this case 90°. |
| 4 | The RWTH function `WaitFor` makes sure that MATLAB waits until the motor is across the conveyor belt and has stopped before taking on a new task. |
| 5 | Finds the new position of the motor and saves this in `pos`. |
| 6 | Updates `TachoLimit` (set equal to `pos`) and sends the motor object `mUp` to NXT, which moves the arm to stationary position. |
| 7 | Waits until the arm is in stationary position and the motor has stopped before ending the function. |

The MATLAB implementation can be found in appendix E.8.

# Chapter 8

# Experiments & Results

In this Chapter, the kNN classifier is implemented in `main.m` and both the software (MATLAB) and the physical system are tested, with class 1 and class 2 objects. Issues that have arisen in the course of these experiments are also presented. Results are presented in tables for every experiment. The systems overall experiment are discussed in the last section in this Chapter.

## 8.1 Implement the kNN Classifier to the System

The collected data (`Data`) included data from class 1 and 2, and it contained 280 different samples (140 samples per class). The data for each class was divided into four random data sets (D). A combination of three of these D's was defined as the training set (TS) and the remaining D, that is left out of TS, was defined as the test/validation set. This was used when performing cross validation experiments on the different classifiers.

From section 6.4 it was concluded that the kNN classifier with normalized 4D data and $k_n$ equal 3 gave the best result from the cross validation experiements, because all criterias, presented in section 6.2.2, was met. Compared to other classifiers that met the criterias, it had the lowest mean validation error and least spread in data.
All collected data from class 1 and 2 was used when implementing the final kNN classifier in the system. The features from an object that passed through the system was normalized using the mean and standard deviation computed from `Data`. The feature vector is then classified to `Data`, with the kNN classifier.

### 8.1.1 Results from Current Setup

To test the final kNN classifier, the following steps was decided.

- A minimum of two separate experiments
- The object from each class will be classified 30 times
(total attempts: 60)
- Objects will be classified in random order
- Number of correct and wrong classifications (NOCC and NOWC) are logged
- After the experiment, the error rate are computed

The expectation before the experiments was to have a error rate close to 6-10%. Results from the cross validation performed on the kNN classifier indicated a mean validation error of 6.43% when 35 data features was classified to 2 × 105 (class1 and class 2). In the system, one feature is classified to the 280 previously logged samples in Data, and this will affect the result. At the same time the current illumination will also affect the result. Light and reflections can either distort or better the captured frame containing the object of interest, and affect the feature values.

The table below shows the result from two separate experiments performed on the final kNN classifier that was implemented in the system.

| **Experiment** | **NOWC for class 1** | **NOWC for class 2** | **Error Rate [%]** |
|---|---|---|---|
| 1 | 18 | 0 | 30.00 |
| 2 | 23 | 0 | 38.33 |

Table 8.1: Two separate experiments performed on the kNN classifier for normalized 4D data and $k_n$ equal 3. Features are classified to Data.

The error rate is a lot higher then expected for both experiments. One reason for this may be illumination. The light environment were most likely different when the experiments was conducted compared to when the data was collected and stored in Data.mat. This can affect the system and give a higher error rate because the values in the computed feature vector is different from the values in Data.

It can also be mentioned that it is only the class 1 object that is wrongly classified. This can indicate that the data from class 1 and 2 are very similar, but also that the data from class 2 are more spread then for class 1. When a feature computed from the class 1 object is classified to Data, the probability of classifying it to class 2 is higher because of the spread.

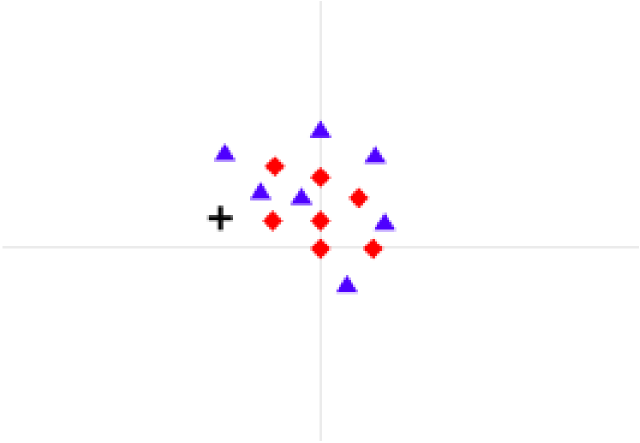


Figure 8.1: Example: kNN with $k_n$ equal 3. The feature vector indicated by a cross belongs to class 1 and are being classified. The red samples are class 1 and the blue samples are class 2. Because of more spread in the class 2 samples, the feature vector is classified to class 2.

In the next section, new and previously collected data are compared and examined.

### 8.1.2 Comparing New Captured Features with the Current Data Set (`Data.mat`)

Ten new features from both classes was collected and was compared to `Data`. The reason for this was to see if the current features had changed compared to previously collected data. The comparison is shown in figure 8.2.

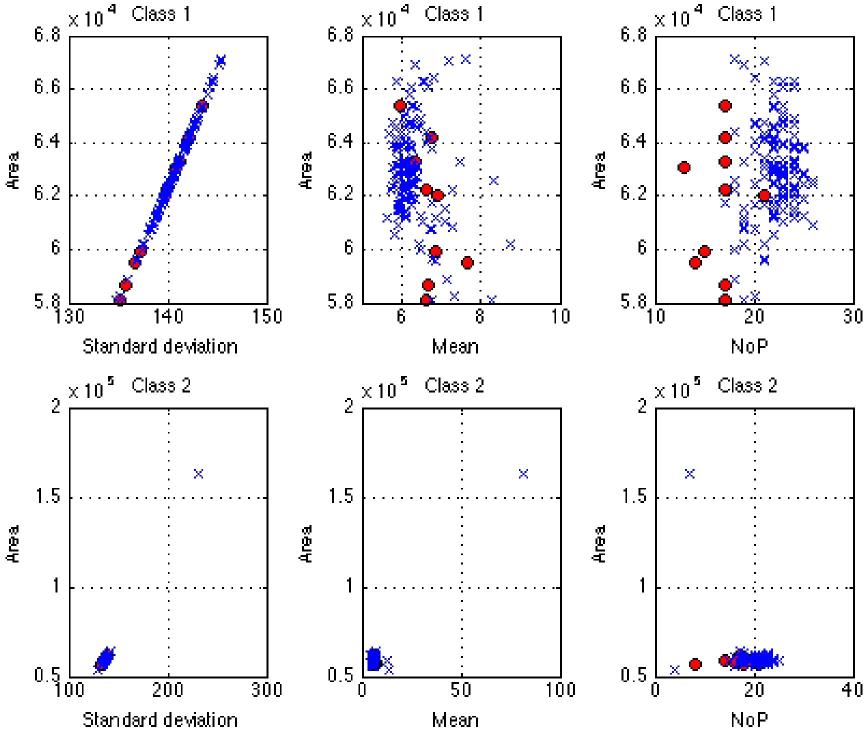


Figure 8.2: Comparison of new and old data. All images shows 2D data. The axes are organized as follows: [Area, Mean], [Area, Std] and [Area, NoP]. The top three images are for class 1 and the bottom three images are for class 2. The red dots are the new features.

The images in figure 8.2 shows that the new data is similar to the old data. There are some small deviations and this is because of varying illuminations. For class 2 there are two outliers in the data set. The first outlier is shown in all images for class 2. The area exceeds $1.5 \times 10^4$. The second outliers is shown in the last images for class 2 ([Area, NoP]). The number of peaks detected are less then 5.

The two outliers in `Data` are most likely data calculated from a false positive detection in `MOD()`. A change in light may have been reflected by the conveyor belt and been detected by `MOD()`. The area of the false positive detected object is part of the conveyor belt were the reflection is brightest. As the outliers show, all feature values except from NoP becomes much larger than normal.

### 8.1.3 Removing Outliers from the Current Data Set

Since the two outliers, detected in class 2, have been part of training sets that have been used when training classifiers with cross validation, they may have affected the result. To examine this, the outliers are removed from `Data` and the kNN classifier are tested again. The data for each class has been reduced to $4 \times 136$ (four features are removed from each class). The new and reduced data set is called `Data2.mat`. The $\mathrm{k_n}$ parameter is still set to 3. This parameter value gave the best result when cross validation was performed in section 6.3.3, on the kNN classifier with normalized 4D data.

Two separate experiments was performed on the system with `Data2`. The experiments was performed with the same steps as for the test in section 8.1.1. The results is shown in the table below.

| **Experiment** | **NOWC for class 1** | **NOWC for class 2** | **Error Rate [%]** |
|---|---|---|---|
| 1 | 17 | 0 | 28.33 |
| 2 | 19 | 0 | 31.67 |

Table 8.2: Results from the kNN experiments with `Data2` and $\mathrm{k_n}$ equal 3.

The results are a little better after removing the outliers, but still not good enough. Since the data in `Data2` is reduced, compared to `Data`, and the $\mathrm{k_n}$ parameter isn't changed, a new cross validation was performed to check if another parameter value could improve the system. Figure 8.3 shows the cross validation for the kNN classifier with normalized 4D data.

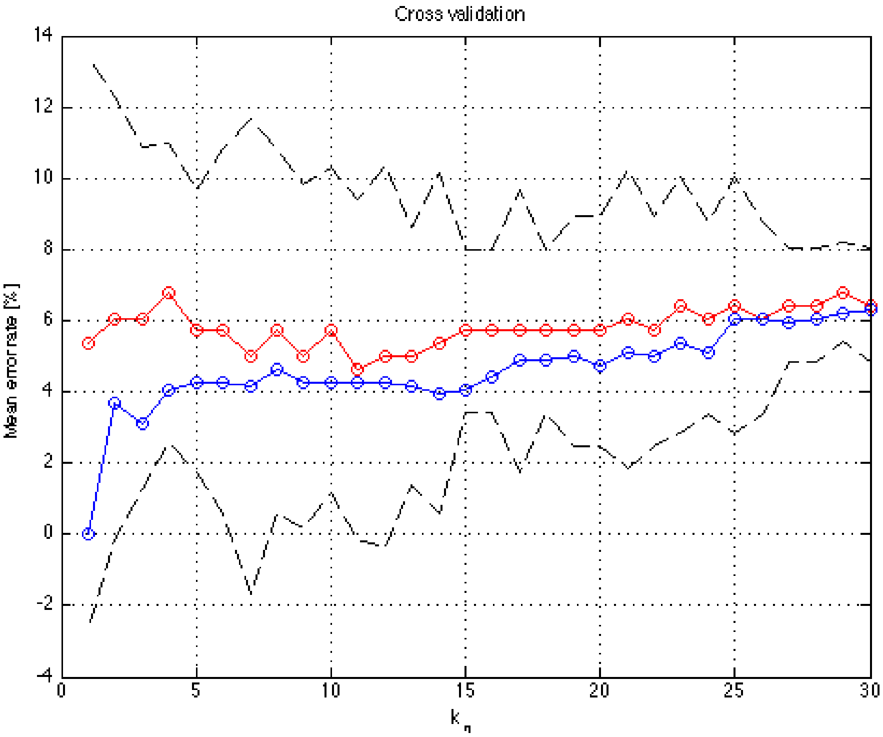


Figure 8.3: Cross validation for the kNN classifier and `Data2` (normalized 4D).

From the cross validation, $k_n$ equals 29 was chosen. It has good generalization, but not the lowest mean validation error. This parameter was chosen since the data has least uncertainty for this parameter value. The kNN classifier was tested with the new $k_n$ value. The result is shown in the table below.

| **Experiment** | **NOWC for class 1** | **NOWC for class 2** | **Error Rate [%]** |
|---|---|---|---|
| 1 | 12 | 0 | 20.00 |
| 2 | 15 | 0 | 25.00 |

Table 8.3: Results from the kNN experiments with `Data2` and $k_n$ equal 29.

By changing the $k_n$ value, the result has improved, but not as much as expected. This suggests that the deviation between new and old data, presented in figure 8.2, could affect the result.

### 8.1.4 Collect New Data

Since the error rate of the classifier implemented in the system didn't produce a satisfying result, it was decided to collect new data from both classes. The amount of data collected was 52 features from each class. This data set is called `Data3.mat`. The reason for collecting new data was to confirm that the deviation between data (new and old) in figure 8.2 really affected the classifier and produced a higher error rate than expected.

#### 8.1.4.1 Cross Validation

`Data3` is pre processed and divided into randomly data sets. It was done in the same manner as described in section 6.1. The cross validation was only performed on the kNN classifier for normalized 4D data. Each of the data sets that is randomly picked from `Data3` is $4 \times 13$ in size. Figure 8.4 shows the result from the cross validation.

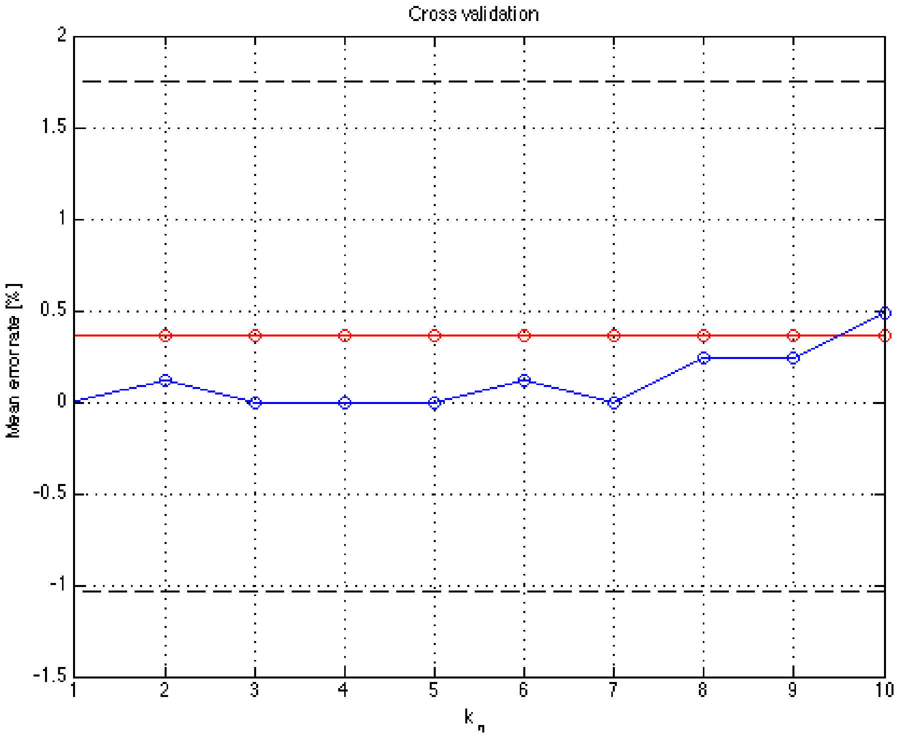


Figure 8.4: Cross validation for the kNN classifier and `Data3` (normalized 4D).

For every $k_n$ parameter the cross validation indicates constant uncertainty in the data. The mean validation error is also constant and the generalization is acceptable in every case. $k_n$ equals 6 was chosen as the parameter.

#### 8.1.4.2 Result

This experiment was done four times with the same starting point as all the other experiments. The result is shown in the table 8.4.

| Experiment | NOWC for class 1 | NOWC for class 2 | Error Rate [%] |
|---|---|---|---|
| 1 | 4 | 0 | 6.67 |
| 2 | 2 | 0 | 3.33 |
| 3 | 6 | 0 | 10.00 |
| 4 | 6 | 0 | 10.00 |

Table 8.4: Results from the kNN experiments with `Data3` and $k_n$ equal 6.

The results from the kNN classifier with the new data set has significantly improved from previously experiments with `Data` and `Data2`. This indicates that the deviations in data from figure 8.2 has affected the classifier. The error rate isn't constant for each experiment, but this is due to illumination that affects the values computed in the feature vector. As long as the illumination varies, so will the values. The system is placed in a room that has low illumination variations. The windows are blinded with dark curtains, but there is still changes in light.
To reduce the effect of light variations, the system have to be placed in a room that always have constant illumination and preferably no windows. Another measure is to improve the moving object detection algorithm and make it more robust against light and reflections.

`Data`, `Data2` and `Data3` has one thing in common. The data for the class 2 object is more spread than the data for class 1, in both sets. It's only the class 1 object that is wrongly classified in every case for both data sets.

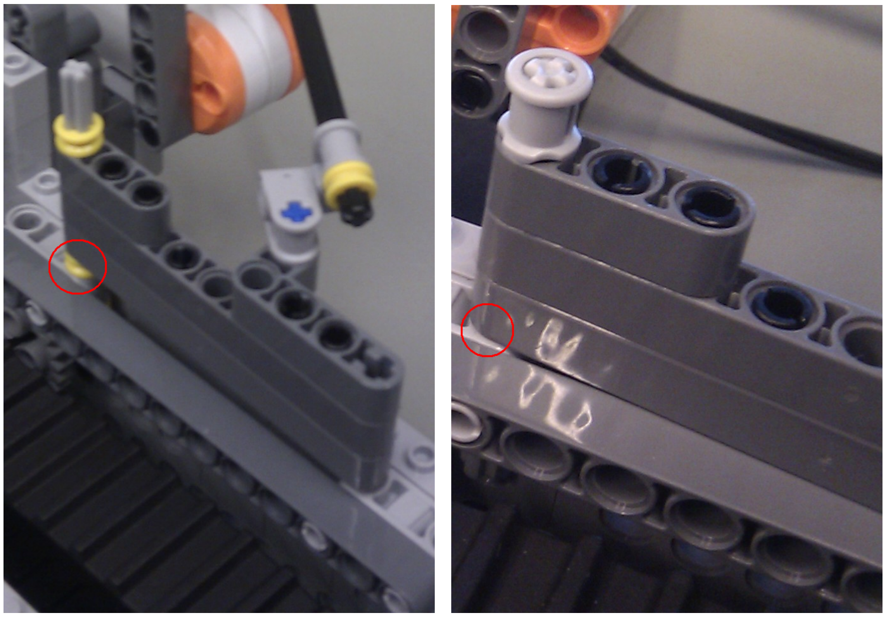
The kNN classifier with $k_n$ equal 6 are implemented in the system with `Data3`.

## 8.2 Mechanical Adjustments

There were made three mechanical adjustments on the physical system while performing the final testing.

### Sorter Arm

The first adjustment was to the sorter arm. When the arm was in it's working position across the conveyor belt, both objects went under the arm. The vertical position of the arm was too high. This was adjusted and the figure below shows the arm before and after the adjustment.



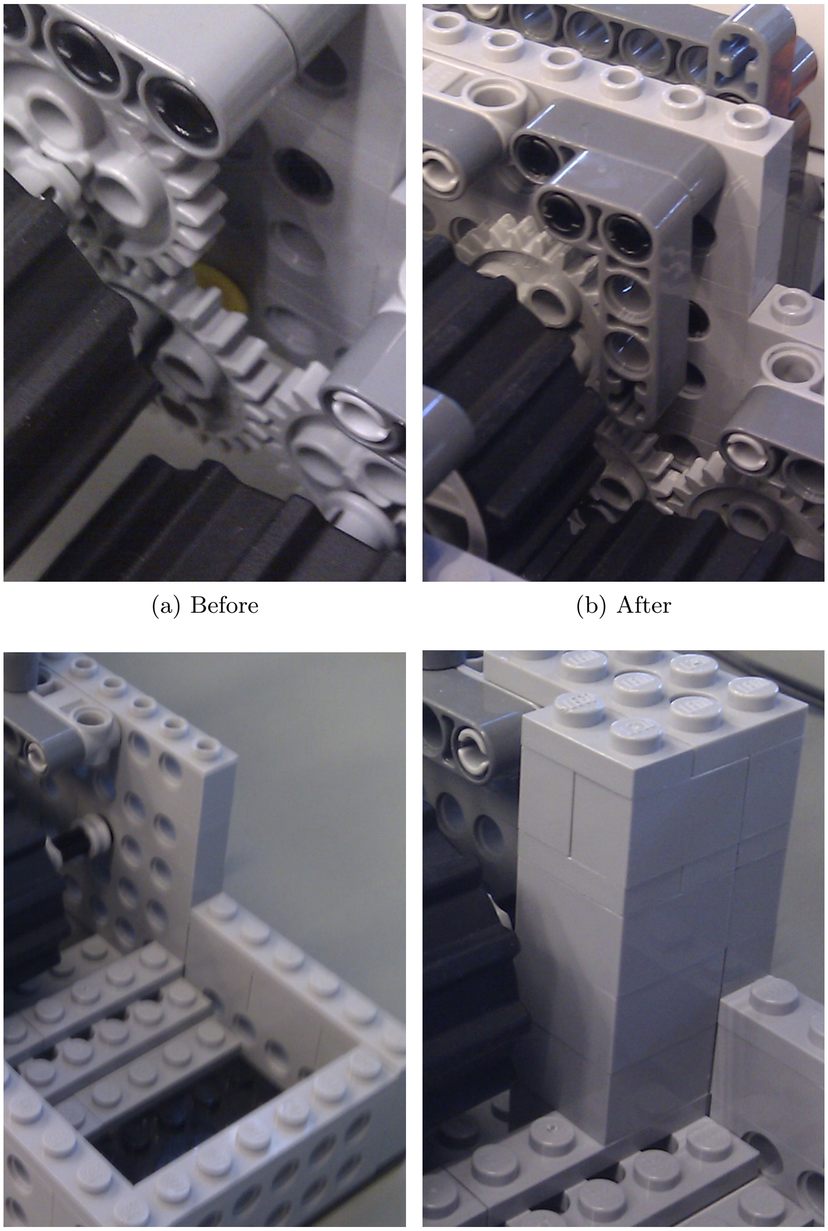
(a) Before

(b) After

Figure 8.5: The sorter arm before and after adjusting it's vertical position. The red ring, in both images, indicates the part that was adjusted.

### Seal holes/openings surrounding the conveyor belt

The two last adjustments was to seal holes and openings surrounding the conveyor belt. Both objects was transported the whole length of the belt to look for areas were they could slip through a opening and fall outside. Figure 8.6 shows two areas on the conveyor belt that was adjusted.



(a) Before

(b) After

(c) Before

(d) After

Figure 8.6: Mechanical adjustments made on areas surrounding the conveyor belt. This prevents the objects from falling off the conveyor belt and possibly get stuck in gears that runs the conveyor belt forward.

## 8.3 Timing the Sorter Arm

When the class 1 object is detected and classified, the sorter arm will move to it's working position and push the object off the conveyor belt, and into tray 1.

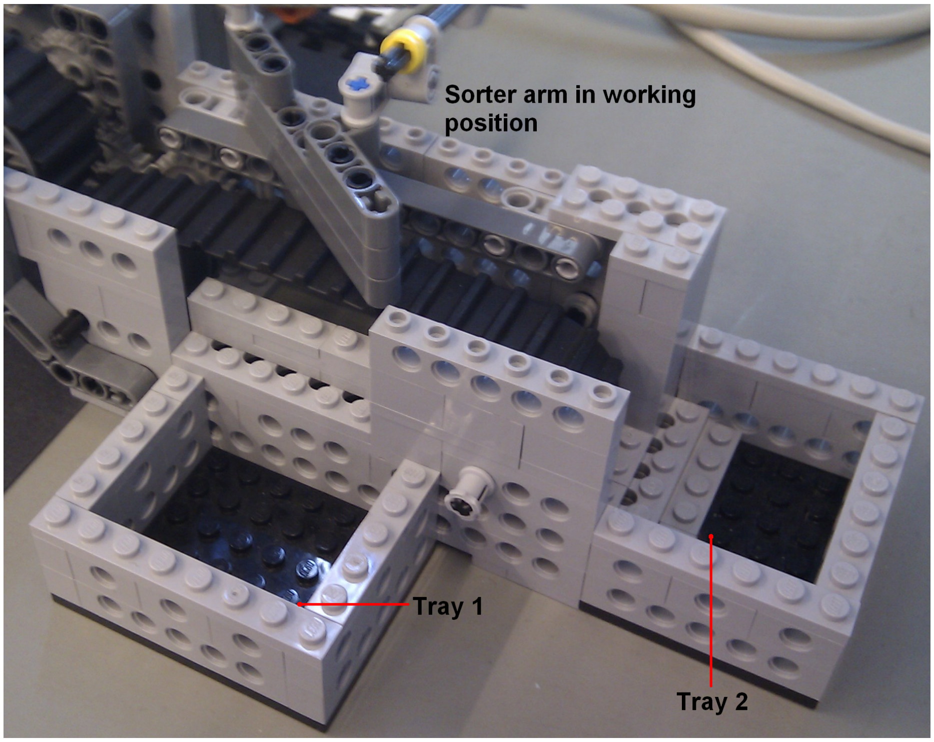


Figure 8.7: The image show tray 1 and 2, which collects the sorted objects. When the sorter arm is in it's working position, class 1 objects are pushed into tray 1. Class 2 objects are collected in tray 2.

When a class 1 object is recognized and classified, the sorter move to it's working position. It needs to stay out long enough so that the object can move along the conveyor belt and get pushed into tray 1. The solution was to use a pause function in MATLAB, which is invoked after the arm is moved across the belt. This time delay is set to 7 seconds . After the delay, the arm is moved back to it's stationary position. The delay is added between step 5 and 6 in the sorter arms flowchart, which is presented in section 7.9.

## 8.4 Overall Performance

For every experiment carried out in this Chapter, the system has not had any false positive detections. The illumination from the lights have been constant, but because of windows in the room, the overall illumination has varied. This shows that the work and development of the moving object detection algorithm (`MOD()`) has made it more robust against light and reflections from the conveyor belt. The algorithm is still affected by the light. The computed feature values are different from one experiment to another because of it, but also because an object will never be placed in the exact same place on the conveyor belt.
A part from not having any false positive detections during the experiments, MATLAB have crashed a couple of times. It's mainly the camera and MATLABs video input that stops logging frames. A quick restart solves the problem. The crashes may occur after detecting and classifying 50-100 objects. These implications seem to be random. It may be the web camera that don't always respond to MATLABs inquiry to start logging frames.

The last cross validation experiment, presented in section 8.1.4.1, gave a mean validation error below 0.5%, but this isn't the case when it was tested. The result show a varying error rate between 3 and 10%. It is still an acceptable result. The illumination affects these results as well, but the deviations between the data in `Data3.mat` and the features extracted from the passing objects isn't as big as it was while using `Data.mat`.

Results and experiments from the moving object detection and for finding / extracting the object of interest is presented in Chapter 4 and 5.

# Chapter 9

# Conclusion & Further Development

## 9.1 Conclusion

The physical sorter system was designed in the preliminary project *Objekt sorterer - Egenskaps uttrekning* (MIK110), but it was made three important mechanical adjustments to the system in this report.
The first one was to reduce the effect of reflection around the conveyor belt where the web camera is located, to avoid false positive detections. This was done by using black tape on the areas where the reflection was high and affected the resulting frame containing the object of interest in the form of noise. The second adjustment was to secure / seal areas where objects could fall off the conveyor belt along the entire transport stage. The last adjustments involved lowering the sorter so that the objects didn't travel beneath it.
These three adjustments made the system both more secure and stable. The mechanical reduction of reflection especially made it easier to pre process captured frames in MATLAB.

The development of the moving object detection was one of the most crucial and most important tasks performed in this report. The research area concerning moving object detection is big and introduced me to several solutions. This field is constantly evolving, but a final solution seems yet to be found. There was no tangible theory to commit to. One common factor for all research, was of course, the problem concerning illumination. This was no exception in my work. Typically, in a industrialized situation, illumination is avoided by placing systems handling detection of objects, sorting packages, etc. in a space that is not exposed to constantly change in light. The illumination is constant and other light sources, that can affect the system, is excluded. For my part, this was not possible. The system had to tolerate small light changes. This was made possible by using two methods called background subtraction (BS) and background modeling (BM). Both BS and BM are basic methods that is used for moving object detection. Beside from these two methods, the developed moving object detection algorithm compares computed thresholds from a grayscale version from the BS, to determine if the object is in the frame field of view (FOV). Experiments showed that this threshold value became bigger when a object entered the FOV. This was then utilized as security measurement to confirm that the object of interest actually was present in a frame. It also supports the BM in reducing the effect of reflection, which produces false positive detections.

All in all, the results that are presented in this report are satisfying. The main theory behind the solution is based on the image processing and pattern recognition subjects, as intended. The report also focuses on moving object detection which I found out was a relatively interesting part of image processing theory. Further research within this field of work could be very beneficial for future students, as well as for my self. Working with methods based on the pattern recognition subject was also intriguing. When I was introduced to this subject in the spring of 2011, I couldn't imagine how to transform the theory behind it into a practical example. I wasn't the only one having this problem. The work behind this reports classification problem has really been a eye opener and has given me more understanding about this field and it's potential. It has been an obligatory subject for earlier students attending Information Technology (IT) at the University of Stavanger. In my time at IT it was only a optional course, but as for next semester (fall 2012) it will once again be obligatory and I truly support this decision.

The solution also gives a good fundament for further development. There is still a lot that can be improved and there's the possibility of scaling up or expand the system. It could be used as an practical example on how image processing and pattern recognition can be used in real life. Students can use the model and work on it as an assignment or project.

## 9.2 Further Development

This section mention some suggestions for further development.

### 9.2.1 Improving the Moving Object Detection Algorithm

The moving object detection algorithm, `MOD()`, is based on background subtraction (BS) and background modeling (BM). The BM and the user defined threshold (UDT) is used to adapt to gradual illumination changes, and prevent illumination from causing false positive detections. The BM only adapts to slow changes in illumination. It's robustness against dynamic backgrounds is also a issue.

[4] mentions several different background models that are more adaptable to dynamic backgrounds. One of the models mentioned is the Mixture of Gaussian (MoG). The pixels in a frame containing a dynamic background varies from another frame. MoG models these variations with K[1] Gaussian distribution, for each captured frame and compares them to existing models at the same location in the frame. Matched models are updated with a learning factor and unmatched models are discarded, and replaced by new Gaussian models [4]. MoG has been used to model complex dynamic backgrounds.

[2] mentions a method to reduce the affect of sudden illumination problems. It's based on entropy and exploits the fact that dark image scenes have low entropy whereas light image scenes have high entropy.

### 9.2.2 Expand Number of Classes

If new objects that can be classified and sorted from each other are found and the number of classes in the classification exceeds two, an expansion

[1] Usually K is between 3 and 5.

is needed on the physical system. Figure 9.1 and 9.2 shows two possible expansions of trays.

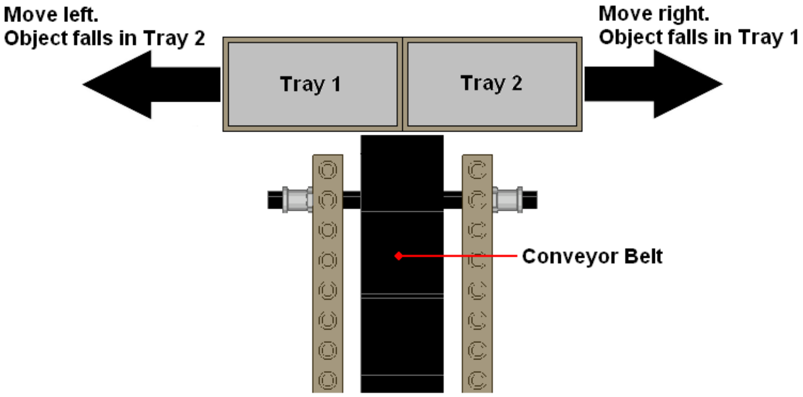


Figure 9.1: In this expansion, two trays are mounted at the end of the conveyor belt. The trays are controlled by a NXT servo and a gear mechanism that can slide the trays left and right.

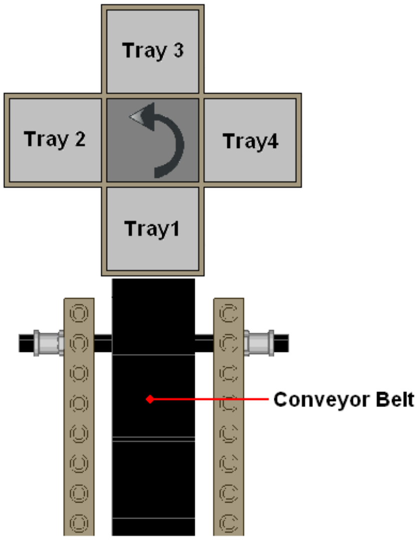


Figure 9.2: In this expansion, four trays are mounted at the end of the conveyor belt. The trays are controlled by a NXT servo that rotates the tray cluster 360°. If tray 2, 3 or 4 is placed at the end of the conveyor belt, the NXT servo have to rotate either 90°, 180° or 270°.

In addition to expand the physical system, new features may be needed as well. New objects require new cross validation experiments and the features that are used in this report might be inoperable to classify the new objects. Other than the traditional classifiers that have been presented in this report may also be considered.

A part from this, a new MATLAB function that controls the added trays need to be programmed. If the total amount of servos are more than three after the expansion, a second NXT needs to be implemented into the system. The RWTH Toolbox supports multi-NXT use, but it requires two bluetooth handles in MATLAB.

# Bibliography

[1] Balthasar, D., Erdmann, T., Pellenz, J., Rehrmann, V., Zeppen, J. & Priese, L. (2001) 'Real-time Detection of Arbitrary Objects in Alternating Industrial Environments', *Proceedings Scandinavian Conference on Image Analysis (2001)*, pp. 321-328. Available from: `http://www.uni-koblenz.de/~lb/publications/Balthasar2001RDO.pdf` (Obtained: 10.01.2012).

[2] Cheng, F-C., Huang, S-C. & Ruan S-J. (2011) 'Illumination-Sensitive Background Modeling Approach for Accurate Moving Object Detection', *IEEE Transaction On Broadcasting*, 57(4), pp. 794-801. [Online] DOI: `http://dx.doi.org/10.1109/TBC.2011.2160106` (Obtained: 23.01.2012).

[3] Duda, R. C., Hart, P. E. & Stork D. G. (2000) *Pattern Classification*. 2nd ed. New York: Wiley Interscience.

[4] Elhabian, S. Y., El-Sayed, K. M., Ahmed, S. H. (2008) 'Moving Object Detection in Spatial Domain using Background Removal Techniques - State-of-Art', *Recent Patents on Computer Science*, 1(1), pp. 32-54. Available from `http://www.benthamscience.com/cseng/samples/cseng1-1/Elhabian.pdf` (Obtained: 18.01.2012)

[5] Gonzalez, R. C., Woods, R. E. & Eddins S. L. (2009) *Digital Image Processing using MATLAB*. 2nd ed. USA: Gatesmark Publishing.

[6] Hagen, P. C. (2007) *Innføring I Sannsynlighetsregning Og Statestikk*. 5th ed. Oslo: Cappelen Akademiske Forlag as.

[7] Heikkilä, J. & Silvén, O. (2004) 'A Real-Time System for Monitoring of Cyclists and Pedestrians', *Image and Vision Computing*, 22(7), pp. 563-570. [Online] DOI: `http://dx.doi.org/10.1016/j.imavis.2003.09.010` (Obtained: 30.01.2012).

[8] Karasulu, B. & Korukoglu, S. (2012) 'Moving Object Detection and Tracking by Using Annealed Background Subtraction Method in Videos: Performance Optimization', *Expert Systems with Applications: An International Journal*, 39(1), pp. 33–43. [Online] DOI: `http://dx.doi.org/10.1016/j.eswa.2011.06.040` (Obtained: 30.02.2012).

[9] Koller, D., Weber, J. & Malik, J. (1993) 'Robust Multiple Car Tracking with Occlusion Reasoning', *ECCV '94 Proceedings of the third European conference on Computer vision*, 1, pp. 189-196. Available from: `http://www.cse.unr.edu/~bebis/CS791E/FinalPresPapers/CarTracking.pdf` (Obtained: 12.02.2012).

[10] *Mac - NXT communication* (2009) Available from: `http://sites.google.com/site/sienamatlabnxt/getting-started/mac` (Obtained: 13.10.2011).

[11] *MathWorks: Support* (2012) Available from `http://www.mathworks.se/support/` (Obtained: 07.02.12).

[12] Pham, P. T., Prostov, Y. I. & Suarez-Alvarez, M. M. (2012) 'Statistical approach to normalization of feature vectors and clustering of mixed datasets'. [Online] DOI: `http://dx.doi.org/10.1098/rspa.2011.0704` (Obtained: 20.04.2012).

[13] *RWTH - Mindstorms NXT Toolbox: Functions by Category* (2012) Available from `http://www.mindstorms.rwth-aachen.de/documents/downloads/doc/version-4.04/categories.html` (Obtained: 20.01.12).

[14] *RWTH - Mindstorms NXT Toolbox* (2012) Available from `http://www.mindstorms.rwth-aachen.de/` (Obtained: 07.02.2012).

[15] Stauffer, C. & Grimson, W. E. L. (1999) 'Adaptive background mixture models for real-time tracking', *IEEE*

*Comput. Soc. (1999)*, 2(c), pp. 246-252. Available from `http://www.ai.mit.edu/projects/vsam/Publications/stauffer_cvpr98_track.pdf` (Obtained: 19.01.2012).

[16] Su, S-T. & Chen, Y-Y. (2008) 'Moving Object Segmentation Using Improved Running Gaussian', *Computing: Techniques and Applications (2008). DICTA '08. Digital Image*, pp. 24-31. [Online] DOI: `http://dx.doi.org/10.1109/DICTA.2008.15` (Obtained: 15.02.2012).

[17] Theodoridis, S. & Koutroumbas, K. (2008) *Pattern Recognition.* 4th ed. USA: Academic Press.

[18] Yi, Z. & Liangzhong, F. (2010) 'Moving Object Detection Based on Running Average Background and Temporal Difference', *International Conference on Intelligent Systems and Knowledge Engineering (2010)*, pp. 270-272. [Online] DOI: `http://dx.doi.org/10.1109/ISKE.2010.5680866` (Obtained: 01.02.2012).

# Appendix A

# Content on CD

| **Folder** | **Content** |
|---|---|
| *Bibliography* | The folder contains some of the referred papers in Bibliography. |
| *Collected Data* | Contains `Data.mat`, `Data2.mat` and `Data3.mat`. |
| *Master Thesis* | Contains a digital copy (.pdf) of the master thesis. |
| *MATLAB Implementations* | Contains all m-files that are developed in MATLAB. |
| *RWTH Toolbox* | Contains a digital copy (.zip) of the RWTH toolbox v4.04.<br>Used in this report. |
| *Video* | A short video of the system when it's running. The video is also on YouTube:<br>`http://www.youtube.com/watch?v=AAGNuTu7Tfk&feature=plcp`. |

The CD is located on the last page of the report (back cover).

# Appendix B

# Set Theory

This chapter looks at basic concepts behind set theory. Equations 2.2, 2.3, 2.4 and 2.5 in chapter 2 are based on set theory.

## B.1 Basic Concepts

Consider the following two sets, A and B.

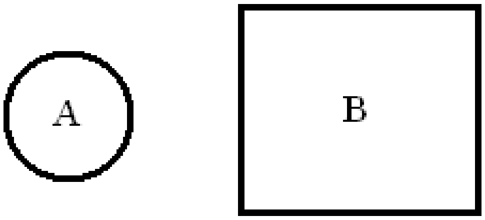
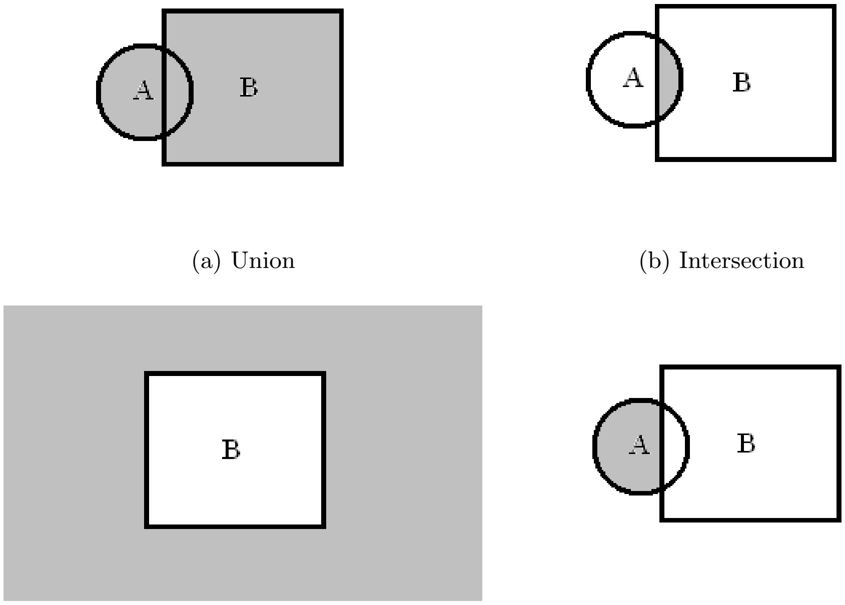


Figure B.1: Set A and B.

| Definition | Notation | Explanation |
|---|---|---|
| Union | $A \cup B$ | A and B. |
| Intersection | $A \cap B$ | A or B. |
| Complement | $(A)^c$ or $(B)^c$ | Not A / Not B. |
| Difference | $A - B$ | A minus B. |
| Empty set | $\emptyset$ | Contains no elements. |
| Reflection | $\hat{A}$ or $\hat{B}$ | Reflects A or B. |
| Translation | $(A)_z$ or $(B)_z$ | Move the sets origin. |

Table B.1: Set theory definitions [5].



(a) Union

(b) Intersection

(c) Complement

(d) Difference

Figure B.2: Four basic concepts in set theory.

# Appendix C

# Calculations

## C.1 Example: Converting Cartesian Coordinates to Polar

Consider the complex number $z = a + jb$.
Definitions:

$$\begin{array}{ll} Re(z) = a & a = r \cdot cos(\theta) \\ Im(z) = b & b = r \cdot sin(\theta) \\ Arg(z) = \theta = tan^{-1}\left(\frac{b}{a}\right) & |z| = r\underbrace{\left(cos(\theta) + jsin(\theta)\right)}_{e^{j\theta}} \end{array}$$

**Cartesian form:** $r = \sqrt{(x - x_1)^2 + (y - y_1)^2}$, origin: $(x_1,\, y_1)$
**Polar form:** $z = re^{j\theta}$

Table C.1 show some border coordinates from an object, calculated from the MATLAB function `bwboundaries`.

| **x** | `BC(x, 1)` **- n-points** | `BC(x, 2)` **- m-points** |
|---|---|---|
| 1 | 243 | 191 |
| 2 | 244 | 190 |
| 3 | 244 | 189 |
| 4 | 244 | 188 |
| 5 | 245 | 187 |

Table C.1: Some border coordinates computed by the function `bwboundaries`.

Centroid coordinate (`n0`, `m0`): 373.9, 198.1.
Shift origin for each n- and m-point: `BC(x, 1/2) = BC(x, 1/2) - n0/m0`.
Table C.2 shows the result.

| **x** | `BC(x, 1)` **- n-points** | `BC(x, 2)` **- m-points** |
|---|---|---|
| 1 | -130.9 | -7.1 |
| 2 | -129.9 | -8.1 |
| 3 | -129.9 | -9.1 |
| 4 | -129.9 | -10.1 |
| 5 | -128.9 | -11.1 |

Table C.2: Coordinates after the origin is shifted.

To use the MATLAB function `cart2pol` the image coordinates have to be converted to a coordinate system that MATLAB uses. The conversion is as follows

$$xcart = BC(x, 2) = m \tag{C.1}$$

$$ycart = -BC(x, 1) = -n \tag{C.2}$$

Convert the Cartesian coordinates to polar:

| **Polar form** | | |
|---|---|---|
| x | $r$ | $\theta$ |
| 1 | 131.00 | 0.0542 |
| 2 | 130.15 | 0.0623 |
| 3 | 130.22 | 0.070 |
| 4 | 130.29 | 0.0776 |
| 5 | 129.38 | 0.0852 |

Table C.3: Calculated polar coordinates.

$\theta$ is in radians. The function `signature` converts $\theta$ to degrees and saves the first angle as 0 and the last angle as 359 (1° increment). Table C.4 shows the computed $r$ for $x \epsilon \{1, 2, \ldots, 5\}$.

| x | **Computed $r$ in MATLAB** |
|---|---|
| 1 | 129.8991 |
| 2 | 129.9288 |
| 3 | 130.9885 |
| 4 | 131.1357 |
| 5 | 130.2747 |

Table C.4: Computed distances ($r$) in MATLAB.

## C.2 Calculating the Validation and Training Error

The data in the tables below is from a Parzen Window classification experiment. The first table shows the number of correct classifications (NOCC) and the number of wrong classifications (NOWC), for the test sets. The second table shows NOCC and NOWC for the training sets. D1 and D2 are the test sets for class 1 and 2, and TS1 and TS2 are the training sets for class 1 and 2.

| **Parzen Window Classification on normalized 2D data (h1 = 1)** | | | | |
|---|---|---|---|---|
| i | NOCC in D1{i} | NOWC in D1{i} | NOCC in D2{i} | NOWC in D2{i} |
| 1 | 30 | 5 | 24 | 11 |
| 2 | 34 | 1 | 31 | 4 |
| 3 | 31 | 4 | 29 | 6 |
| 4 | 29 | 6 | 26 | 9 |

Table C.5: Data from test sets.

| **Parzen Window Classification on normalized 2D data (h1 = 1)** | | | | |
|---|---|---|---|---|
| i | NOCC in TS1{i} | NOWC in TS1{i} | NOCC in TS2{i} | NOWC in TS2{i} |
| 1 | 92 | 13 | 82 | 23 |
| 2 | 99 | 6 | 92 | 13 |
| 3 | 93 | 12 | 79 | 26 |
| 4 | 91 | 14 | 80 | 25 |

Table C.6: Data from training sets.

Equations 6.3 and 6.4, extracted from the *Confusion Matrix*, is used to calculate the error rate of each test and training set.

## C.2 Calculating the Validation and Training Error

The next two points show how to calculate the error rate for the first test set (D1{1} and D2{1}):

**1. Find the confusion matrix:**

$$A = \begin{pmatrix} 30 & 5 \\ 11 & 24 \end{pmatrix} \tag{C.3}$$

**2. Calculate Ac and E from the confusion matrix:**

$$Ac = \frac{30 + 24}{70} \approx 0.7714$$
$$E = 1 - 0.7714 = 0.2286$$

The error rate for the first test is 22.86%.

The error rate is calculated for all the other test and training sets in the same way. The table below shows the result.

| **Correct classification (Ac) and error (E) rates (%)** | | | | |
|---|---|---|---|---|
| i | $\text{Ac}_{\text{test}}(i)$ | $\text{E}_{\text{test}}(i)$ | $\text{Ac}_{\text{train.}}(i)$ | $\text{E}_{\text{train.}}(i)$ |
| 1 | 77.14 | 22.86 | 82.86 | 17.14 |
| 2 | 92.86 | 7.14 | 92.38 | 7.62 |
| 3 | 85.71 | 14.29 | 81.9 | 18.10 |
| 4 | 78.57 | 21.43 | 81.42 | 18.58 |

Table C.7: The calculated Ac and E for each test and training set.

Mean validation error:

$$\bar{E}_{\text{test}} = \frac{1}{4}\sum_{i=1}^{4} E_{\text{test}}(i) \approx 16.43\%$$

Mean training error:

$$\bar{E}_{\text{train.}} = \frac{1}{4}\sum_{i=1}^{4} E_{\text{train.}}(i) \approx 15.72\%$$

# Appendix D

# Implementation of Software and NXT

This appendix presents the setup that have to be done to achieve communication between MATLAB and NXT. The setup was implemented on the Mac OS X platform.
The guidelines shown in this appendix are based on a step by step guide found on the internet [10], but it is shown in more detail in the coming pages, in this report.

## D.1 Add the RWTH Toolbox in the MATLAB Environment

The RWTH toolbox is available for free download via [14] (*Download*). There are several versions of the toolbox, but the version(s) marked with a star are recommended, since these versions is the latest stable release.

The following guidelines shows how to add the RWTH toolbox to the MATLAB environment.

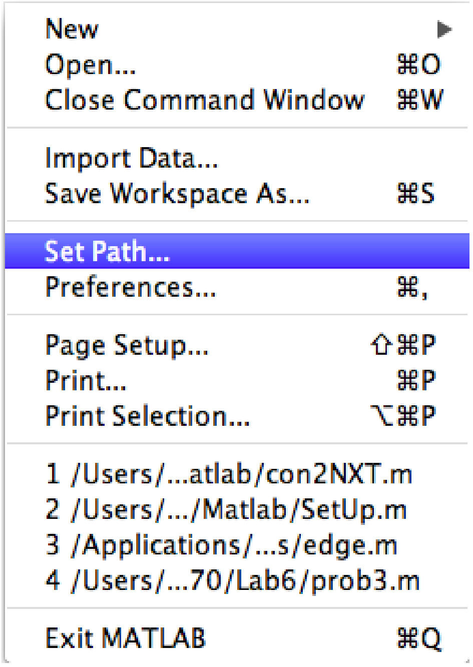
1) Download the version of the toolbox marked with a star as mentioned

before (v4.04 is used in this setup).

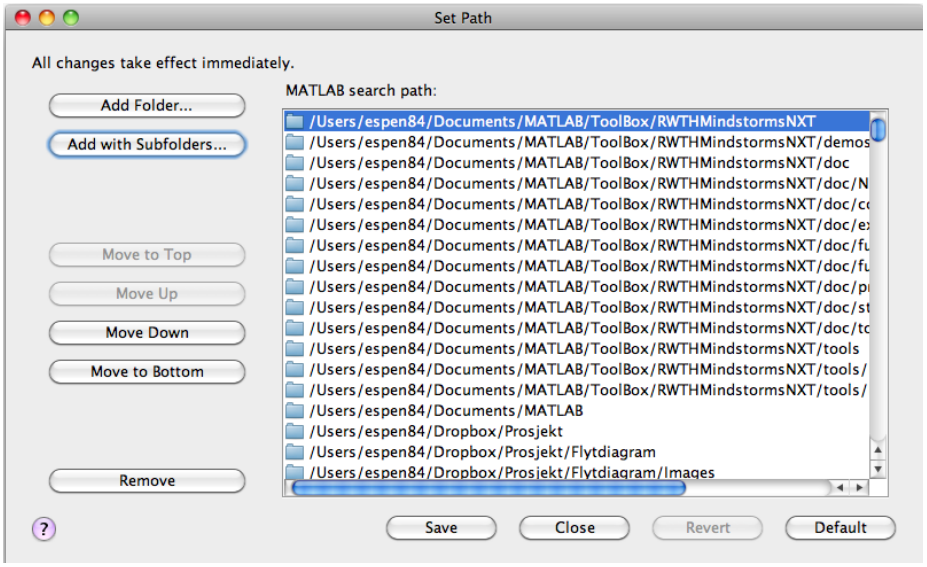
2) Use a compression program, like WinRAR, to extract the downloaded zip-compressed toolbox. The unzipped file is a folder containing the toolbox.

3) In the MATLAB folder, under **Documents**, make a new folder called **Toolbox**. Place the RWTH toolbox in this folder.

4) Open MATLAB and select **Set path...** from the drop-down file menu.



5) Then choose **Add with Subfolders...** and find the folder containing the RWTH toolbox. Mark the folder and click **Open**.



All changes take effect immediately.

MATLAB search path:

/Users/espen84/Documents/MATLAB/ToolBox/RWTHMindstormsNXT
/Users/espen84/Documents/MATLAB/ToolBox/RWTHMindstormsNXT/demos
/Users/espen84/Documents/MATLAB/ToolBox/RWTHMindstormsNXT/doc
/Users/espen84/Documents/MATLAB/ToolBox/RWTHMindstormsNXT/doc/N
/Users/espen84/Documents/MATLAB/ToolBox/RWTHMindstormsNXT/doc/c
/Users/espen84/Documents/MATLAB/ToolBox/RWTHMindstormsNXT/doc/e
/Users/espen84/Documents/MATLAB/ToolBox/RWTHMindstormsNXT/doc/fu
/Users/espen84/Documents/MATLAB/ToolBox/RWTHMindstormsNXT/doc/fu
/Users/espen84/Documents/MATLAB/ToolBox/RWTHMindstormsNXT/doc/p
/Users/espen84/Documents/MATLAB/ToolBox/RWTHMindstormsNXT/doc/st
/Users/espen84/Documents/MATLAB/ToolBox/RWTHMindstormsNXT/doc/t
/Users/espen84/Documents/MATLAB/ToolBox/RWTHMindstormsNXT/tools
/Users/espen84/Documents/MATLAB/ToolBox/RWTHMindstormsNXT/tools/
/Users/espen84/Documents/MATLAB/ToolBox/RWTHMindstormsNXT/tools/
/Users/espen84/Documents/MATLAB
/Users/espen84/Dropbox/Prosjekt
/Users/espen84/Dropbox/Prosjekt/Flytdiagram
/Users/espen84/Dropbox/Prosjekt/Flytdiagram/Images

6) Check the **MATLAB search path** and confirm that the path to the toolbox are added to the list. If it isn't, repeat from step 4. When the toolbox is found in the list, click **Save** and then **Close**.

When step 6 is done, the toolbox should be added to the MATLAB environment. One way of checking this is to look in MATLABs help function for the toolbox, or write `help COM_OpenNXT` in the command window. If MATLAB prints the help text for this NXT function in the command window the toolbox is added to the MATLAB environment.

## D.2 USB Connection

The first connection between NXT and the Mac platform is the USB connection. This is used to set up the NXT and prepare the bluetooth connection. Follow the next steps to initialize the USB.

1) Download the **Fantom Driver** from `http://mindstorms.lego.com/en-us/support/files/Driver.aspx`.

**Remember** to choose the Mac version!

2) Open the downloaded file, *legodriver.pkg*, and follow the installation guidelines.

3) When this installation is completed the USB connection is ready for use.

## D.3 Update NXT Firmware

Before preparing the bluetooth connection the NXT firmware **should** be upgraded to the newest version. The next steps shows how to update the NXT firmware.

1) Turn on the NXT by holding the orange button down.

2) Push the left arrow two times until the ***Settings*** icon is shown in the NXT display and then push the orange button to access this menu.

3) Push the left arrow two times until the ***NXT Version*** icon is shown in the NXT display and then push the orange button to access this menu.

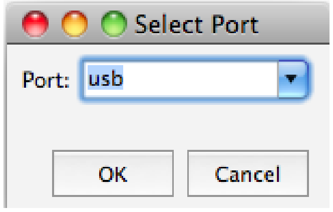
4) Now the firmware version will be unveiled on the display.

5) If there is a newer firmware version it can be downloaded from `http://mindstorms.lego.com/en-us/support/files/Firmware.aspx`.

6) Regardless of whether the firmware version must be updated or not, a program called **NeXT Tools** have to be downloaded from `http://bricxcc.sourceforge.net/utilities.html`.

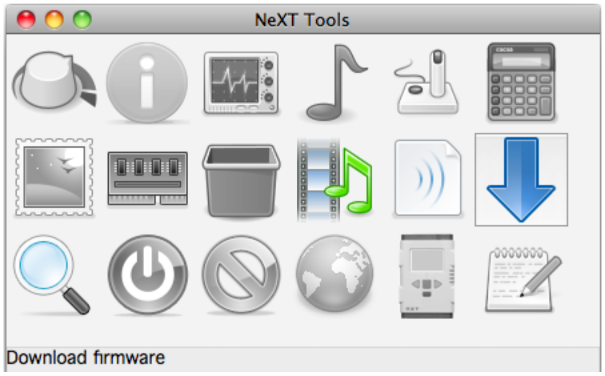
7) Connect the NXT to the Mac with a USB cable.

8) Open **NeXT Tools** and choose ***usb*** when the program prompts **Select Port**.



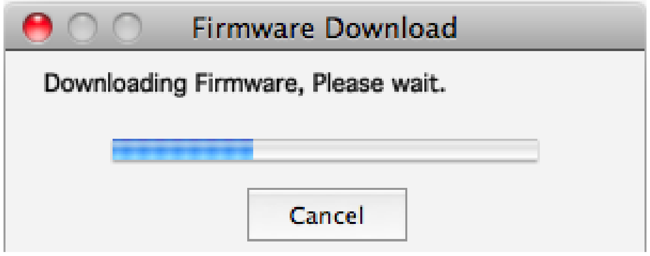
Port: usb

9) Choose ***Download firmware*** from the menu.



10) Find the downloaded firmware file and click **Open**.

11) Now the new firmware will be installed on the NXT and the window below should appear.



Downloading Firmware, Please wait.

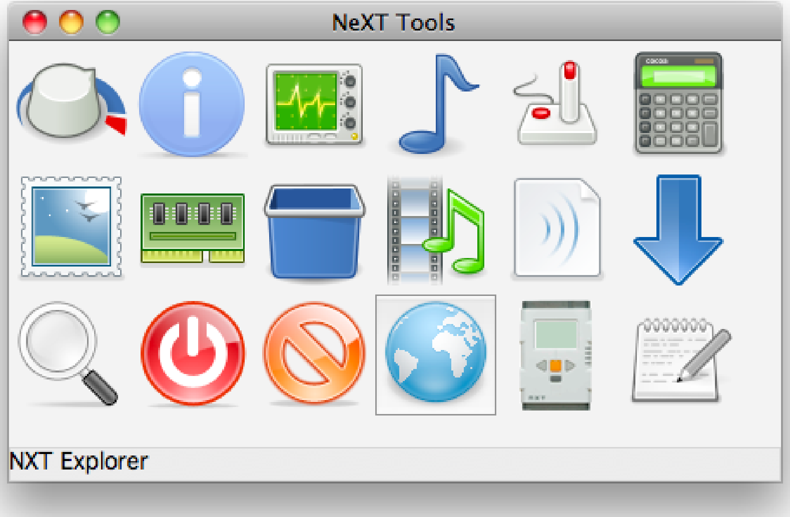
12) When the new firmware is installed, go over step 1 to 4 again and confirm it.

## D.4 Installing *Motor Control* on NXT

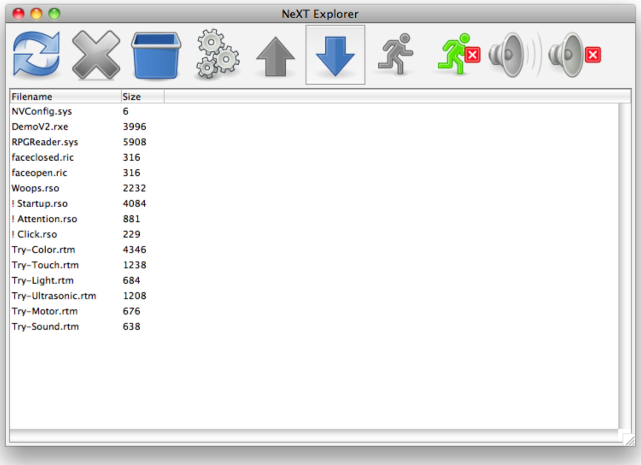
*Motor Control* is a program that have to be transferred to the NXT via NeXT Tools. While a MATLAB program is running the Motor Control

program receives commands from MATLAB. Then it controls the motors / servos precisely while the MATLAB program continues to run [14]. The Motor Control program is included in the RWTH toolbox.
The following steps show how to install the Motor Control on NXT.

1) Connect the NXT to the Mac with a USB cable.
2) Turn on NXT.
3) Open **NeXT Tools** and choose ***usb*** as mentioned before.
4) Click on the ***NXT Explorer*** from the menu (globe icon).

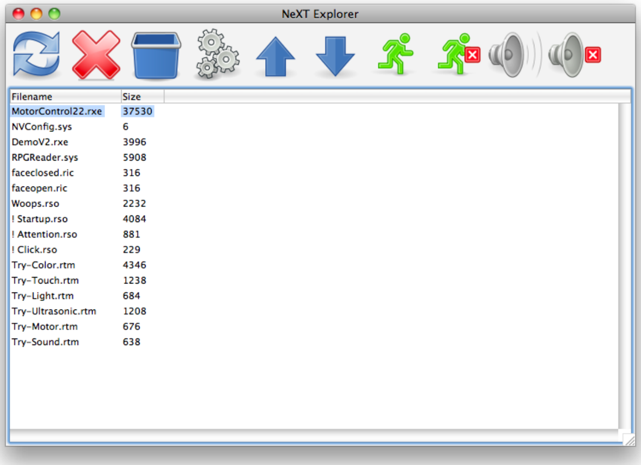


5) Click on ***Download selected files to the NXT*** as shown in the next figure and find the file *MotorControl22.rxe* in the RWTH toolbox folder (RWTHMindstormsNXT/tools/Motor- Control/).



| Filename | Size |
|---|---|
| NVConfig.sys | 6 |
| DemoV2.rxe | 3996 |
| RPGReader.sys | 5908 |
| faceclosed.ric | 316 |
| faceopen.ric | 316 |
| Woops.rso | 2232 |
| ! Startup.rso | 4084 |
| ! Attention.rso | 881 |
| ! Click.rso | 229 |
| Try-Color.rtm | 4346 |
| Try-Touch.rtm | 1238 |
| Try-Light.rtm | 684 |
| Try-Ultrasonic.rtm | 1208 |
| Try-Motor.rtm | 676 |
| Try-Sound.rtm | 638 |

6) When step 5 is done the Motor Control file should be in the list as shown in the next figure (marked file).



| Filename | Size |
|---|---|
| MotorControl22.rxe | 37530 |
| NVConfig.sys | 6 |
| DemoV2.rxe | 3996 |
| RPGReader.sys | 5908 |
| faceclosed.ric | 316 |
| faceopen.ric | 316 |
| Woops.rso | 2232 |
| ! Startup.rso | 4084 |
| ! Attention.rso | 881 |
| ! Click.rso | 229 |
| Try-Color.rtm | 4346 |
| Try-Touch.rtm | 1238 |
| Try-Light.rtm | 684 |
| Try-Ultrasonic.rtm | 1208 |
| Try-Motor.rtm | 676 |
| Try-Sound.rtm | 638 |

7) Check that the Motor Control program is transferred to the NXT by choosing *My Files* and then *Software Files* on the NXT. The program will be placed here if it's successfully transferred.

## D.5 Initialize Bluetooth Connection Between Mac OS X 10.6 (Snow Leopard) and NXT

Follow this setup to initiate the bluetooth communication.

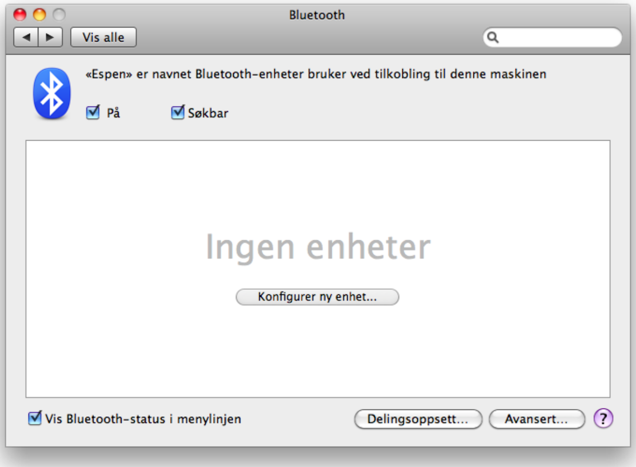
1) Turn on the NXT.

2) Enter the ***Bluetooth*** menu on the NXT by pushing the left arrow three times and then push the orange button.

3) Push the left arrow two times until the ***On/Off*** icon is displayed and turn the bluetooth on (a icon will appear in the upper left corner of the display).

4) In the ***Bluetooth*** menu, also check that the ***Visibility*** is turned on.

5) Turn on the bluetooth on the Mac by clicking on ***Settings*** and then ***Bluetooth***. A list will appear on the screen, like the one below.



«Espen» er navnet Bluetooth-enheter bruker ved tilkobling til denne maskinen

☑ På ☑ Søkbar

Ingen enheter

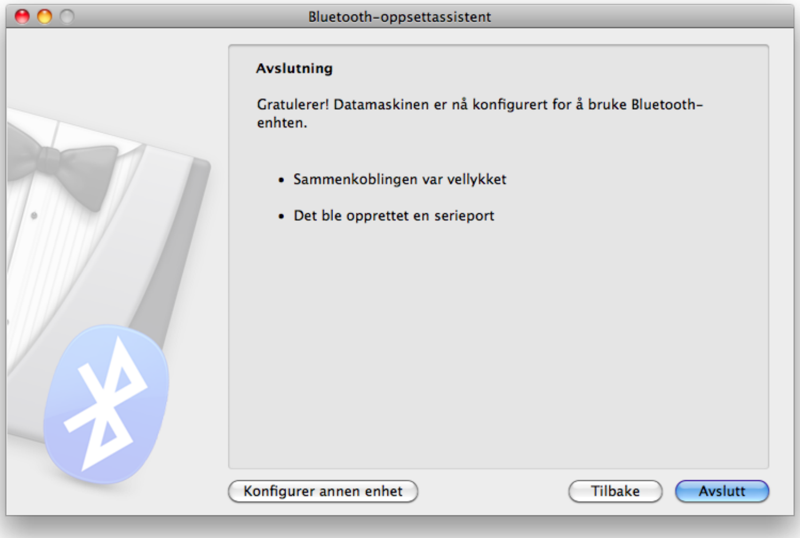
☑ Vis Bluetooth-status i menylinjen

6) Choose ***Konfigurer ny enhet*** and the Mac will search for bluetooth units.

7) In the list of bluetooth units that appears on the screen, double click on the right name (in this case; EXPLORER).

8) A password is needed to connect to the unit and it appears on the NXTs display. The user needs to enter the password on the NXT (usually the password is 0000) and finish by pressing ✓.

9) If the bluetooth connection is successful the window in the figure below will appear.



Bluetooth-oppsettassistent

Avslutning

Gratulerer! Datamaskinen er nå konfigurert for å bruke Bluetooth-enheten.

- Sammenkoblingen var vellykket
- Det ble opprettet en serieport

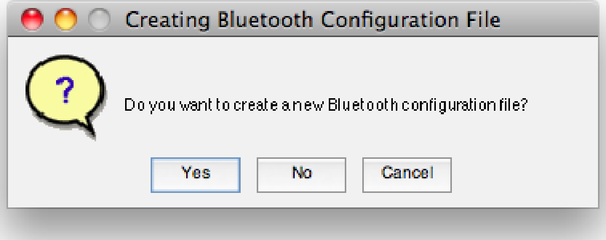
## D.6 Create a Bluetooth Configuration File

For MATLAB and NXT to communicate, one must use a so called SSP, *Serial Port Profile*. In the RWTH toolbox documentation the user is told to create a configuration file that can handle different bluetooth adaptors on different computers, and contain settings for MATLAB functions. This file is called *bluetooth.ini* and is created in MATLAB.
To create this file follow these steps.

1) Run MATLAB.

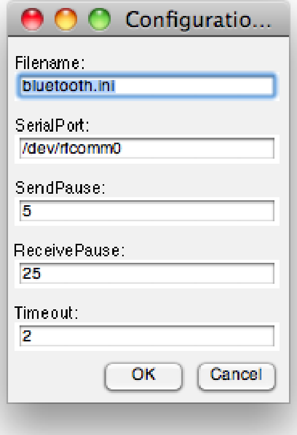
2) Enter `COM_MakeBTConfigFile` in the command window.

3) After a little while this window will appear on the screen. Click **Yes**.



Do you want to create a new Bluetooth configuration file?

4) MATLAB will now create the configuration file and the user have to choose were to save the file on the hard drive. Save it in RWTH toolbox folder. The window shown below will appear. Leave it be! It will be used in later steps.



Filename: bluetooth.ini

SerialPort: /dev/rfcomm0

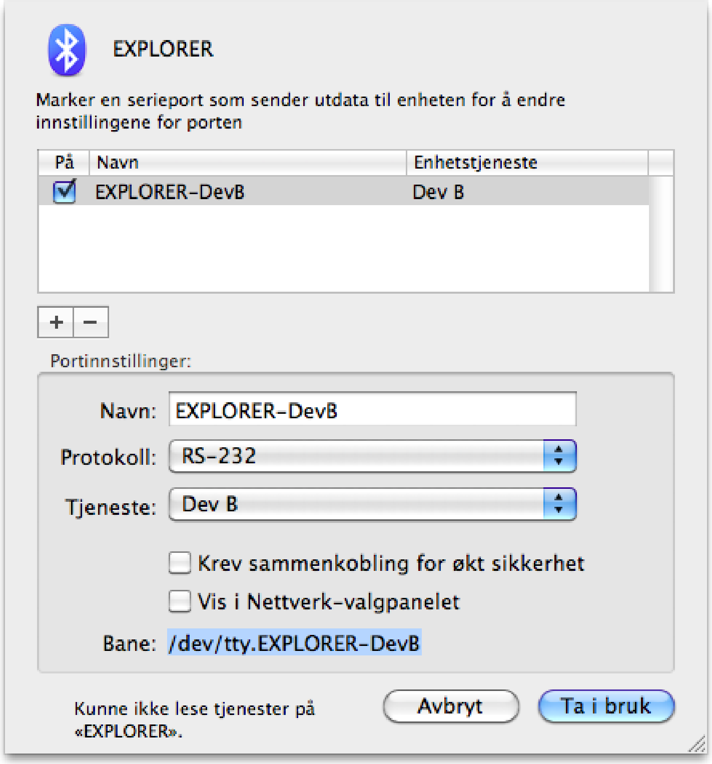
SendPause: 5

ReceivePause: 25

Timeout: 2

5) On the Mac, click on ***Settings*** and then ***Bluetooth***. Mark the name of the NXT and click on the gear icon. Click on ***Rediger serieporter...***

6) Copy the address marked on the figure below.



EXPLORER

Marker en serieport som sender utdata til enheten for å endre innstillingene for porten

| På | Navn | Enhetstjeneste |
|---|---|---|
| ☑ | EXPLORER-DevB | Dev B |

Portinnstillinger:

Navn: EXPLORER-DevB

Protokoll: RS-232

Tjeneste: Dev B

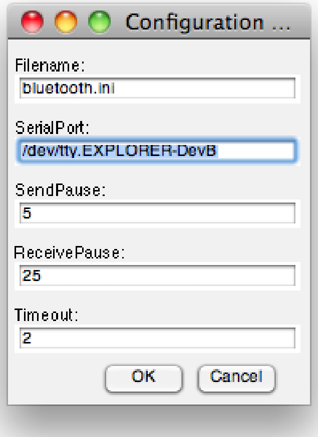
☐ Krev sammenkobling for økt sikkerhet

☐ Vis i Nettverk-valgpanelet

Bane: /dev/tty.EXPLORER-DevB

Kunne ikke lese tjenester på «EXPLORER».

7) Paste the address in the *Serial Port* text box in the window that appeared in step 4, as shown in the next figure and then click **OK**.



Filename:
bluetooth.ini

SerialPort:
/dev/tty.EXPLORER-DevB

SendPause:
5

ReceivePause:
25

Timeout:
2

Now the bluetooth connection should be successful. Follow the example in the next section to ensure that it works.

## D.7 Example: Initiate Communication Between MATLAB and NXT

**NOTE!** Ensure that the motor / servo is connected to PORT A on the NXT before performing the example.

Write the following in the MATLAB command window to initiate bluetooth communication between MATLAB and NXT:

```
HNXT = COM_OpenNXT('bluetooth.ini');
COM_SetDefaultNXT(HNXT);
```

As mentioned in section 3.2.4, `COM_OpenNXT` opens the communication with the help of the created configurations file *bluetooth.ini* and returns the handle `HNXT` to MATLAB. `COM_SetDefaultNXT` sets `HNXT` to a global handle. Now that the communication is ready, MATLAB can send and receive data to the NXT. The lines of code below show how to create a motor object on the NXTs port A and start / stop the motor.

```
% Constructs a motor object on port B with a power of 30.
motorA = NXTMotor('A', 'Power', 30);

% Sends the motor—settings to NXT and starts the motor.
motorA.SendToNXT();

% Pause for 3 seconds (MATLAB function).
pause(3);

% Stop motor.
motorA.Stop();
```

To end the bluetooth communication between MATLAB and NXT, simply enter `COM_CloseNXT('all')` in the command window.

# Appendix E

# Implementations in MATLAB

In this appendix all the algorithms that have been implemented in MATLAB are presented. The developed MATLAB functions that are used to run the system are also shown. These functions are based on and further developed from the algorithms.

## E.1 Main Program - main.m

```
% Declare variables.
global_var

% Connect to NXT.
HNXT = con2NXT();

% Variable used to store data in Class and Img.
i = 0;

% Used to restart or stop/clear program.
log = 1;

while(log == 1)
    % Start logging on camera
    start(Vid);
    pause(3);

    % Start motor.
    mA.SendToNXT();

    % Function to stop the infinite loop.
    % Creates a figure and uses the handle
    % to stop the loop.
    handle = figure('position', [0 0 eps eps],...
        'menubar', 'none');

    % Infinite loop. User can end the loop
    % by pressing 'q' in the command window.
    while(1)
        % Increments i each time a new object is
        % detected.
        i = i + 1;

        % Moving Object Detection.
        [imres] = MOD(Vid, UDT, alpha, Mrm);

        % Selects the best component for
        % extracting object out of imres.
        % Returns the best binary version,
        % imresBW.
        imresBW = rgbComp(imres);

        % Object Of Interest.
        [BC Odata per img] = ooi(imresBW, seO, seE);

        % Calculates the Signature.
        [Sign, Angle] = signature(BC,...
            Odata.Centroid(1),Odata.Centroid(2));

        % The Signature is smoothed before finding
        % the vertices/peaks.
        SmoothSign = smooth(Sign, 0.1, 'loess');

        % Find the mean value of the Signature.
        MSign = mean(Sign);

        % Find the standard deviation of
        % the Signature.
        StdSign = std2(Sign);

        % Finds the number of vertices/peaks
        % in the signature.
        NoP = peaks(SmoothSign, Angle);

        % Feature Vector. Saves each feature
        % from every object
        % in X.
        X = [Odata.Area MSign StdSign NoP]';
        X1{i} = X;
        %disp(['Object number: ' num2str(i)])

        % Normalize X. Uses M and Std calculated
        % from Data.mat.
        X = (X—repmat(M,1,1))./repmat(Std,1,1);

        % Classify the feature vector, X, with Parzen.
        [Cx] = kNN_classifier(data, C, kn, X);

        % If object belongs to class 1, move the
        % sorter arm across conveyor belt.
        % If Cx == 2, the object falls of the
        % conveyor belt at the end.
        if(Cx == 1)
            arm(mDown, mUp, deg);
        end

        % Collect data.
        % Class indicates which class the ith object
        % belongs to.
        % Img stores all OOI.
        Class(i) = Cx;
         imres(per) = 0;
         Img{i} = imres;

        % If user enters 'q' in the command window,
        % MATLAB interprets this as closing figure
        % 'handle'. This again triggers 'break',
        % which ends the loop.
        if strcmp(get(handle,'currentcharacter'),'q')
            close(handle)
            break;
        end
        figure(handle)
        drawnow
    end

    % Stop motor.
    mA.Stop();

    % Stop logging on camera.
    stop(Vid);

    % Shows the total number of objects that
    % are classified and the number of objects
    % classified to class 1 and 2, before user
    % quit the program.
    clc;
    disp(['-------------------------'...
        '-------------------------'])
    disp(['Total number of objects classified: '...
        num2str(i)])
    disp(['Number of objects classified to class 1: '...
        num2str(sum(Class == 1))])
    disp(['Number of objects classified to class 2: '...
        num2str(sum(Class == 2))])
    disp(['-------------------------'...
        '-------------------------'])
    disp(['Img contains every OOI detected.'...
        ' Object is highlighted with a border.'])

    % Enter 'Y' to continue or 'N' to end. If user doesn't
    % want to continue, but ends program, he/she may
    % save the current results to two .mat files
    % (optional).
    rep = input(['Do you want to continue without'...
        ' deleting the result? Y/N: '], 's');
    if(strcmp(rep, 'Y'))
        log = 1;
    elseif(strcmp(rep, 'N'))
        rep = input(['Save the result before'...
            ' ending program? Y/N: '], 's');
        if(strcmp(rep, 'Y'))
            c = clock;
            pause(1);
            file1 = ['Class' num2str(c(3))...
                num2str(c(2)) num2str(c(4))...
                num2str(c(5))];
            file2 = ['Img' num2str(c(3))...
                num2str(c(2)) num2str(c(4))...
                num2str(c(5))];
            save([file1 '.mat'], 'Class');
            save([file2 '.mat'], 'Img');
            COM_CloseNXT(HNXT);
        else
            COM_CloseNXT(HNXT);
        end
        log = 0;
    end
end
clear all;
clc;
```

## E.2 Frame Differencing / Background Subtraction & Background Modeling - MOD()

This is the final algorithm for BS / BM and are implemented in the system. Based on Algorithm 3.

```
function [imres] = MOD(Vid, UDT, alpha, mask)
% Compares (i)th image with (i)th BM by subtraction, and
% returns imres when object is in the camera Field Of
% View (FOV).
% INPUTS:
    % Vid — video object.
    % UDT — User—Defined Threshold. Used to detect
    %       object in FOV.
    % alpha — adapt rate used in BM.
    DET = 1; % DETecting mode ON.
    i = 1;

    while(DET) % While DET is true.
        % Get snapshot.
        im1 = getsnapshot(Vid);
        % Saves the each RGB frame.
        image{i} = im1;
        % Converts im1 to grayscale.
        im1 = rgb2gray(im1);
        % Adjusts the image intensity.
        % (this increases the contrast)
        im1 = imadjust(im1);
        % Saves the (i)th im1 in im{i}.
        im{i} = im1;

        % Generates a background model and computes
        % the threshold of every img.
        if(i <= 30)
            if(i == 1)
               B{i} = im{i};
            else
               B{i} = B{i—1} + alpha*(im{i} — B{i—1});
               img = im{i} — B{i};
               thr{i—1} = graythresh(img);
            end
            % Increment i.
            i = i + 1;
        else
            % The background model adapts to gradual
            % illumination changes (slow). Update BM.
            B{i} = B{i-1} + alpha*(im{i} - B{i-1});

            % Background subtraction/frame
            % differencing (object detection).
            img = im{i} - B{i};

            % Convert to binary image.
            % The function graythresh uses
            % Otsu's method to compute the
            % threshold.
            thr{i-1} = graythresh(img);
            img = im2bw(img, thr{i-1});

            % Clear objects on image border.
            img = imclearborder(img, 8);
            % Fill holes.
            img = imfill(img, 'holes');

            % Computes the average threshold for the last
            % 10 frames.
            avgthr{i-30} = 0;
            for j=0:9
               avgthr{i-30} = avgthr{i-30} + (thr{(i-1)-j}/10);
            end

            % r and c is empty if object does not overlap
            % mask.
            [r c] = find((img & mask));

            % Object detected?
            if(~isempty(r) && ~isempty(c) && avgthr{i-30} > UDT)
                % Returns imres.
                imres = image{i};
                % When an object is detected in FOV,
                % the DETection mode is turned OFF.
                DET = 0;
            else
                % Increment i.
                i = i + 1;
            end
        end
    end
    % Deletes variables in the function that no longer
    % are used when an object is detected. Frees up
    % memory.
    clearvars im1 im img B image thr avgthr r c;
end
```

## E.3 Segmenting Objects from a Background using Information from each RGB Component - rgbComp()

This function is based on algorithm 4

```
function [BWres] = rgbComp(iRGB)
% Finds the best component to extract an object
% from the background in a RGB image by
% comparing the threshold values (computed
% by the graythresh function).
% Components:
    % iR - iRGB(:, :, 1) (RED)
    % iG - iRGB(:, :, 2) (GREEN)
    % iB - iRGB(:, :, 3) (BLUE)

    % Creates a 1-by-3 vector.
    thr = zeros([1 3]);

    % Extract each component from the RGB
    % image and computes each threshold.
    for i=1:3
        img{i} = iRGB(:, :, i);
        thr(i) = graythresh(imadjust(img{i}));
    end

    % Finds the maximum threshold value index.
    [junk idx] = max(thr);

    % Returns the binary version of the image (iR, iG or iB)
    % that have the largest threshold value.
    BWres = im2bw(img{idx}, graythresh(img{idx}));

    % Clear variables/vectors/matrixes from memory.
    clearvars img thr junk idx;
end
```

## E.4 Find and Label Object of Interest - ooi()

This function is based on algorithm 5.

```
function [BC, ObjectData, perimeter, imres] = ooi2(imC, seO, seE)
% Finds and labels object of interest in the input
% image imC. Returns border coord., image data,
% object perimeter and image.

% INPUTS:
    % imC — Binary image reveived from rgbComp.
    % seO — Structural element used for opening.
    % seE — Structural element used for erosion.
% OUTPUTS:
    % BC — Coordinates for the object border.
    % ObjectData — A cell containing area,
    %              centroid and boundingbox of the
    %              object.
    % perimeter — Object perimeter as an binary image.
    % imres — Binary image containg only the object.
    %         Could also be a 0—matrix if MOD has
    %         performed a false positive detection.
% Remove noise.
imC = imopen(imC, seO);

% Smooth the outer boundary of the object.
imC = imerode(imC, seE);

% Fill holes in object.
imC = imfill(imC, 'holes');

% Object data.
[LM junk] = bwlabel(imC);

% Find and extract the object using bwselect
imres = bwselect(LM,180,160,8);
   % Does object overlap 180, 160?
   if(isequal(imres, zeros(370,705)))
       imres = bwselect(LM,220,160,8);
       % Does object overlap 220, 160?
       if(isequal(imres, zeros(360,705)))
           imres = zeros([370 705]);
           ObjectData = [];
           BC = [];
           perimeter = [];
       else % Object overlaps 220, 160.
```

```
           % Get object data.
           ObjectData = regionprops(imres == 1);
           % Find the outer boundary of the object.
           perimeter = bwperim(imres,8);
           % Find the coordinates of the object border.
           coord = bwboundaries(perimeter, 'noholes');
           BC(:, 1) = coord{1}(:, 2); % x/n coordinates.
           BC(:, 2) = coord{1}(:, 1); % y/m coordinates.
       end
   else % Object overlaps 180, 160.
       % Get object data.
       ObjectData = regionprops(imres == 1);
       % Find the outer boundary of the object.
       perimeter = bwperim(imres,8);
       % Find the coordinates of the object border.
       coord = bwboundaries(perimeter, 'noholes');
       BC(:, 1) = coord{1}(:, 2); % x/n coordinates.
       BC(:, 2) = coord{1}(:, 1); % y/m coordinates.
   end
   % Free up memory space.
   clearvars imC L junk coord
end
```

## E.5 Signature - signature.m

```
function [dist, angle] = signature(b, x0, y0)
% SIGNATURE Computes the signature of a boundary
%   [DIST, ANGLE, XC, YC] = SIGNATURE(B, X0, Y0) computes the
%   signature of a given boundary. A signature is defined as
%   the distance from (X0, Y0) to the boundary, as a function
%   of angle (ANGLE). B is an np—by—2 array (np > 2) containing
%   the (x, y) coordinates of the boundary ordered in a clock—
%   wise or counterclockwise direction. If (X0, Y0) is not inc—
%   luded in the input argument, the centroid of the boundary
%   is used default. The maximum size of arrays DIST anf ANGLE
%   is 360—by—1, indicating a maximum resolution of one degree.
%   The input must be a one—pixel—thick boundary obtained, for
%   example, by using function bwboundaries.
%
%   If (X0, Y0) or the default centroid is outside the boundary,
%   the signature is not defined and an error is issued.

% Check dimensions of b.
[np, nc] = size(b);
if (np < nc || nc ~= 2)
    error('b must be of size np—by—2.');
end

% Some boundary tracing programs, such as boundaries.m, result
% in a sequence in which the coordinates of the first and last
% points are the same. If this is the case, in b, eliminate the
% last point.
if isequal(b(1, :), b(np, :))
    b = b((1:np — 1), :);
    np = np — 1;
end

% Compute the origin of vector as the centroid, or use the two
% values specified. Use the same symbol (xc, yc) in case the
% user includes (xc, yc) in the output call.
if nargin == 1
    x0 = sum(b(:, 1))/np; % Coordinates of the centroid.
    y0 = sum(b(:, 2));
end

% Check to see that (xc, yc) is inside boundary.
IN = inpolygon(x0, y0, b(:, 1), b(:, 2));
if ~IN
    error('(x0, y0) or centroid is not inside the boundary.')
end
```

```

% Shift origin of coordinate system to (x0, y0).
b(:, 1) = b(:, 1) - x0;
b(:, 2) = b(:, 2) - y0;

% Convert the coordinates to polar. But first have to convert
% the given image coordinates, (x, y), to the coordinate system
% used by MATLAB for conversion between Cartesian and polar
% coordinates. Designate these coordinates by (xcart, ycart).
% The two coordinate systems are related as follows:
% xcart = y and ycart = -x.
xcart = b(:, 2);
ycart = -b(:, 1);
[theta rho] = cart2pol(xcart, ycart);

% Convert angles to degrees.
theta = theta.*(180/pi);

% Convert to all nonnegative angles.
j = theta == 0; % Store the indices of theta = 0 for use below.
theta = theta.*(0.5*abs(1 + sign(theta)))...
    - 0.5*(-1 + sign(theta)).*(360 + theta);
theta(j) = 0; % To preserve the 0 values.

% Round theta to 1 degree increments.
theta = round(theta);

% Keep theta and rho together for sorting purposes.
tr = [theta, rho];

% Delete duplicate angles. The unique operation also sorts the
% input in ascending order.
[w, u] = unique(tr(:, 1));
tr = tr(u, :); % u identifies the rows kept by unique.

% If the last angle equals 360 degrees plus the first angle,
% delete the last angle.
if tr(end, 1) == (tr(1) + 360)
    tr = tr(1:end - 1, :);
end

% Output the angle values:
angle = tr(:, 1);

% Output the length values.
dist = tr(:, 2);
end
```

## E.6 Detect Vertices / Peaks in Signature - peaks()

This function is based on algorithm 6.

```
function [NoP] = peaks(sign, angle)
% Finds the peaks in the signature by
% comparing sign(i) with sign(i-1) and
% sign(i+1). If sign(i-1) < sign(i) < sign(i+1),
% a peak is found. The variable sign is the
% signature and angle is the point for each
% signature.

    % Starts comparing points in the signa-
    % ture with the second point. Checks sign(1)
    % in the end.
    i = 2;

    % When a peak is found the variable peak
    % increments.
    peak = 0;

    % A vector that stores each point in the
    % angle where a peak is found.
    p = 0;

    % Goes through the signature from the
    % second point to the end.
    while(i < length(angle))
        sign(i);
        % Detect corner (peak).
        if((sign(i) > sign(i-1)) &&...
            (sign(i) > sign(i+1)));
            if(sign(i) > mean(sign))
                peak = peak + 1;
                p(peak) = angle(i);
            end
        end
        % Increment. Check next point in
        % the signature.
        i = i + 1;
    end

    % Checks the first point in the sign-
    % ture. If an object is turned so that
    % the first point in the signature starts
    % in a corner, it will detect all the other
    % corners except at the first point. So if
    % the value of sign(1) is higher than the
    % value of the first peak, sign(1) is cons—
    % idered a peak.
    if(sign(1) > sign(p(1)))
        peak = peak + 1;
    end

    % Number of Peaks that is returned
    % to the main program.
    NoP = peak;

    % Free up memory space.
    clearvars peak i p
end
```

## E.7 kNN Classifier - kNN_Classifier

```
function [Cx] = kNN_classifier(Data, C, kn, X)
    for i = 1:length(Data)
        delta = X - Data(:,i);
        R(1,i) = delta'*delta;
    end
    [junk idx] = sort(R);
    gx(1,1) = (sum(C(1,idx(1:kn)) == 1)./...
            length(find(R <= R(1,idx(kn)))));
    gx(1,2) = (sum(C(1,idx(1:kn)) == 2)./...
            length(find(R <= R(1,idx(kn)))));
    [tmp Cx] = max(gx);
end
```

## E.8 Move the Sorter Arm - arm()

This function is based on an example given in the documentation in [14], under NXTMotor.

```
function arm(mDown, mUp, deg)
% Prepare motor:
    % Stop prospective motor movement.
    mUp.Stop('off');
    % Reset motor position. Position set to 0.
    mUp.ResetPosition();

% Find the initial position of the sorter arm
% and save this position in pos:
    % Read data from motor.
    data = mUp.ReadFromNXT();
    % Find current position.
    pos  = data.Position;

% Position the arm across the conveyer belt:
    % Set TachoLimit to 90 (deg) + pos.
    mDown.TachoLimit = (deg + pos);
    % Send settings to NXT and run motor.
    mDown.SendToNXT();
    % MATLAB waits until previous command is executed.
    mDown.WaitFor();

% Find new position:
    % Read data from motor.
    data = mUp.ReadFromNXT();
    % Find current position.
    pos  = abs(data.Position);

% Holds the arm out for 7 seconds.
    pause(7);

% Position the arm in stationary position:
    % Set TachoLimit to pos
    % (equals total last movement of the motor).
    mUp.TachoLimit = pos;
    % Send settings to NXT and run motor.
    mUp.SendToNXT();
    % MATLAB waits until previous command is executed.
    mDown.WaitFor();
end
```

## E.9 Declare Global Variables - global_var.m

```
%% Creates a video input object.
Vid = videoinput('macvideo',1);
triggerconfig(Vid,'manual');
set(Vid, 'ReturnedColorSpace', 'rgb');
set(Vid, 'ROIPosition', [325 220 705 370]);

%% Parameters used for Moving Object Detection (MOD).
% User—Defined Threshold (used in fdo.m).
UDT = 0.18;

% Adapt rate for the background modeling.
alpha = 0.05;

% Mask with reference mask.
Mrm = zeros([370 705]);  % 0—matrix.
Mrm(66:264,180:200) = 1; % Reference mask.

%% Parameters used for Object of Interest (ooi.m).
nO = 0;
seO = strel('diamond',4);
seE = strel('diamond',1);

%% NXT Motor Control Setup.
% Motor A. Used to run the conveyor belt.
% Properties:
%   * 'A' — selects motor A.
%   * 'Power' — integer from −100 to 100 that sets
%               the power level and direction of
%               a motor.
%   * −10 — power level 10.
%   SmoothStart — lets the motor accelerate smoothly
%                 when it starts.

mA = NXTMotor('A', 'Power', −10);
mA.SmoothStart = true;

% Motor B. Used to control the sorter arm.
% Properties:
%   % 'ActionAtTachoLimit' — Decides how the motor
    %                        will react when it reaches
    %                        the TachoLimit.
    % 'Brake' — Smoothly slowing down the motor when it
    %           reaches the TachoLimit.
    % TachoLimit — An integer that specifies the angle
    %              in degrees the motor will try to reach.
% Setup for moving the arm up and down
    % moveUp — moves the arm from working to
    %          stationary position.
    % moveDown — moves the arm from stationary to
    %            working position.

mDown = NXTMotor('B', 'Power',  −20,...
                      'ActionAtTachoLimit', 'Brake');

mUp = NXTMotor('B', 'Power', 20,...
                      'ActionAtTachoLimit', 'Brake');

% The angle of motor B's movement. Used in conjunction
% with TachoLimit.
deg = 90;

%% Preparing the Parzen Window Classifier (parzen_classifier).
% Data contains 104 samples for class 1 and class 2
% (52 for each class).
% This data was collected 14.05.2012 while doing
% experiments in the system.
% The first data set is Data.mat and it contains
% 280 samples (140 for each class).
% The reduced data set, Data2.mat, contains 272
% samples (136 for each class).
% Each data set can be found on the CD that
% follows the report.
% Data3{1} — Class 1.
% Data3{2} — Class 2.
load Data3.mat

D = [Data3{1} Data3{2}];

% Normalize Data.
M = mean(D,2);
Std = (std(D'))';
for i=1:2
  Data3{i} = (Data3{i}—repmat(M,1,52))./repmat(Std,1,52);
end

data = [Data3{1} Data3{2}];

% Label vector C.
C = [ones([1 length(Data3{1})]) 2*ones([1 length(Data3{2})])];

% kn—nearest neighbor paramter.
kn = 6;
```

## E.10 Classifiers Used in Experiments

### E.10.1 Parzen Window Classifier - parzen_classifier()

```
function [Cx] = parzen_classifier(Data, x, hn, Pw)
% Data — Data for class 1 and 2.
% x — Feature vector.
% hn — Window width.
% Pw — A priori probabilities.

    for i=1:2
       pn = 0;
       for j=1:140
          [delta(i, :)] = norm_classifier(Data{i}(:, j)...
              , hn(i)^2.*eye(4), x);
          pn = pn + delta(i, :);
       end
       gx(i,:) = (Pw(i)*pn)/140;
    end
    [tmp, c] = max(gx);
    Cx = c;
end
```

### E.10.2 ML Classifier - norm_classifier()

```
function [p] = norm_classifier(my, Sgm, X)
% my — Expectation.
% Sgm — Covariance matrix.
% X — Feature vector.

[d, n] = size(X);
% d — Dimension of X.
% n — The amount of samples in X.

% Compute pdf—values for X.
for i=1:n
       x = X(:,i);
       p(1 ,i) = 1/((2*pi)^(d/2)*det(Sgm)^(1/2))...
           *exp(-1/2*((x-my)'*inv(Sgm)*(x-my)));
end
```

## E.11 Frame Differencing / Background Subtraction - Alg2()

This function is based on algorithm 2.

```
function [imres] = Alg2(Vid, UDT, Mrm)
    % Frame Differencing Operation/ Background subtration.
    % Compares (i)th image with (i?1)th by subtraction, and
    % returns imres when object is in the camera Field Of
    % View (FOV).
    % INPUTS:
        % Vid - Video object.
        % UDT - User-Defined Threshold.
        % Mrm - Matrix containing Reference Mask.
    % OUTPUT:
        % imres - image with object in FOV.

    i = 1;
    DET = 1; % DETecting mode ON.

    while(DET)
        % Get frame from camera.
        im1 = getsnapshot(Vid);
        % Save RGB image.
        imRGB{i} = im1;
        % Convert to grayscale.
        im1 = rgb2gray(im1);
        % Adjust image contrast.
        im1 = imadjust(im1);
        % Save im1 in im{i}.
        im{i} = im1;
        if(i <= 10)
           if(i == 1)
               % Increment i.
               i = i + 1;
           else
               % Do BS.
               img = im{i} - im{i-1};
               % Compute threshold.
               thr{i-1} = graythresh(img);
               % Increment i.
               i = i + 1;
           end
        else
            % Do BS.
            img = im{i} - im{i-1};
            % Compute threshold.
            thr{i-1} = graythresh(img);
            % Convert to binary.
            img = im2bw(img, thr{i-1});
            % Clear objects on image border.
            img = imclearborder(img, 8);
            % Fill holes.
            img = imfill(img, 'holes');
            % Compute average threshold.
            avgthr{i-10} = 0;
            for j=0:9
                avgthr{i-10} = avgthr{i-10} + (thr{(i-1)-j}/10);
            end
            % Multply img and Mrm.
            % Find r and c.
            [r c] = find((img & Mrm));
            if(~isempty(c) && ~isempty(r) && avgthr{i-10} > UDT)
                % Return last image.
                imres = imRGB{i};
                DET = 0; % DETecting mode OFF.
            else
                % Increment i.
                i = i + 1;
            end
        end
    end
    % Delete vectors/matrixes from memory.
    clearvars im1 imRGB im img thr avgthr r c
end
```

# Appendix F

# Data Collection from Cross Validation

## F.1 Maximum Likelihood Estimation

### 2D data (Std, NoP)

| **i** | **Cov. Matrix** | **NOWC in D1(i)** | **NOWC in D2(i)** | **NOWC in TS1(i)** | **NOWC in TS2(i)** | **ER for test sets** | **ER for tr. sets** |
|---|---|---|---|---|---|---|---|
| **1** | Gen. | 2 | 16 | 3 | 48 | 25.71 | 24.29 |
| **2** | Gen. | 2 | 17 | 2 | 45 | 27.14 | 22.38 |
| **3** | Gen. | 2 | 7 | 17 | 26 | 12.86 | 20.48 |
| **4** | Gen. | 0 | 20 | 4 | 51 | 28.57 | 26.19 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **23.57** | **23.34** |
| **1** | Diag. | 2 | 27 | 8 | 72 | 41.43 | 38.10 |
| **2** | Diag. | 2 | 27 | 7 | 79 | 41.43 | 40.95 |
| **3** | Diag. | 5 | 9 | 11 | 43 | 20.00 | 21.43 |
| **4** | Diag. | 3 | 23 | 5 | 83 | 37.14 | 41.90 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **35.00** | **35.60** |
| **1** | Unit | 8 | 15 | 33 | 44 | 32.86 | 36.67 |
| **2** | Unit | 11 | 14 | 30 | 45 | 35.71 | 35.71 |
| **3** | Unit | 9 | 13 | 31 | 42 | 31.43 | 34.76 |
| **4** | Unit | 12 | 17 | 29 | 42 | 41.43 | 33.81 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **35.36** | **35.24** |

Table F.1: Unnormalized data for ML estimation (2D).

## Normalized 2D data (Std, NoP)

| **i** | **Cov. Matrix** | **NOWC in D1(i)** | **NOWC in D2(i)** | **NOWC in TS1(i)** | **NOWC in TS2(i)** | **ER for test sets** | **ER for tr. sets** |
|---|---|---|---|---|---|---|---|
| **1** | Gen. | 1 | 17 | 3 | 54 | 25.71 | 27.14 |
| **2** | Gen. | 3 | 9 | 16 | 23 | 17.14 | 18.57 |
| **3** | Gen. | 1 | 13 | 3 | 47 | 20.00 | 23.81 |
| **4** | Gen. | 1 | 16 | 3 | 52 | 24.29 | 26.19 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **21.79** | **23.93** |
| **1** | Diag. | 3 | 27 | 6 | 81 | 42.86 | 41.43 |
| **2** | Diag. | 2 | 11 | 14 | 33 | 18.57 | 22.38 |
| **3** | Diag. | 3 | 27 | 6 | 76 | 42.86 | 39.05 |
| **4** | Diag. | 2 | 24 | 6 | 81 | 37.14 | 41.43 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **35.60** | **36.07** |
| **1** | Unit | 10 | 17 | 31 | 41 | 38.57 | 34.29 |
| **2** | Unit | 4 | 8 | 14 | 21 | 17.14 | 16.67 |
| **3** | Unit | 9 | 15 | 32 | 43 | 34.29 | 35.71 |
| **4** | Unit | 11 | 11 | 30 | 47 | 31.43 | 36.67 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **30.36** | **30.83** |

Table F.2: Normalized data for ML estimation (2D).

## 3D data (Mean, Std, NoP)

| **i** | **Cov. Matrix** | **NOWC in D1(i)** | **NOWC in D2(i)** | **NOWC in TS1(i)** | **NOWC in TS2(i)** | **ER for test sets** | **ER for tr. sets** |
|---|---|---|---|---|---|---|---|
| **1** | Gen. | 3 | 7 | 4 | 18 | 14.29 | 10.48 |
| **2** | Gen. | 2 | 4 | 4 | 20 | 8.57 | 11.43 |
| **3** | Gen. | 3 | 2 | 11 | 8 | 7.14 | 9.05 |
| **4** | Gen. | 1 | 4 | 3 | 13 | 7.14 | 7.62 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **9.29** | **9.65** |
| **1** | Diag. | 2 | 23 | 5 | 52 | 35.71 | 27.14 |
| **2** | Diag. | 2 | 14 | 5 | 64 | 22.86 | 32.86 |
| **3** | Diag. | 5 | 2 | 12 | 10 | 10.00 | 10.48 |
| **4** | Diag. | 1 | 22 | 6 | 56 | 32.86 | 29.52 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **25.36** | **25.00** |
| **1** | Unit | 8 | 5 | 14 | 7 | 18.57 | 10.00 |
| **2** | Unit | 4 | 2 | 21 | 7 | 8.57 | 13.33 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **3** | Unit | 5 | 4 | 12 | 13 | 12.86 | 11.91 |
| **4** | Unit | 5 | 2 | 19 | 9 | 10.00 | 13.33 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **12.50** | **12.14** |

Table F.3: Unnormalized data for ML estimation (3D).

### Normalized 3D data (Std, NoP)

| **i** | **Cov. Matrix** | **NOWC in D1(i)** | **NOWC in D2(i)** | **NOWC in TS1(i)** | **NOWC in TS2(i)** | **ER for test sets** | **ER for tr. sets** |
|---|---|---|---|---|---|---|---|
| **1** | Gen. | 0 | 6 | 5 | 19 | 8.57 | 11.43 |
| **2** | Gen. | 1 | 8 | 2 | 17 | 12.86 | 9.05 |
| **3** | Gen. | 2 | 4 | 5 | 20 | 8.57 | 11.90 |
| **4** | Gen. | 8 | 3 | 8 | 8 | 15.71 | 7.62 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **11.43** | **10.00** |
| **1** | Diag. | 0 | 17 | 7 | 75 | 24.29 | 39.05 |
| **2** | Diag. | 2 | 24 | 5 | 61 | 37.14 | 31.43 |
| **3** | Diag. | 1 | 27 | 6 | 60 | 40.00 | 31.43 |
| **4** | Diag. | 10 | 3 | 10 | 8 | 18.57 | 8.57 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **30.00** | **27.62** |
| **1** | Unit | 7 | 10 | 24 | 27 | 24.29 | 24.29 |
| **2** | Unit | 6 | 7 | 29 | 30 | 18.57 | 28.10 |
| **3** | Unit | 9 | 15 | 23 | 31 | 34.29 | 25.71 |
| **4** | Unit | 6 | 3 | 8 | 7 | 12.86 | 7.14 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **22.50** | **21.31** |

Table F.4: Normalized data for ML estimation (3D).

### 4D data (Area, Mean, Std, NoP)

| **i** | **Cov. Matrix** | **NOWC in D1(i)** | **NOWC in D2(i)** | **NOWC in TS1(i)** | **NOWC in TS2(i)** | **ER for test sets** | **ER for tr. sets** |
|---|---|---|---|---|---|---|---|
| **1** | Gen. | 2 | 4 | 8 | 5 | 8.57 | 6.19 |
| **2** | Gen. | 4 | 6 | 3 | 24 | 14.29 | 12.86 |
| **3** | Gen. | 1 | 8 | 4 | 36 | 12.86 | 19.05 |
| **4** | Gen. | 0 | 13 | 4 | 31 | 18.57 | 16.67 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Mean Error Rate (test sets/training sets):** | | | | | | **13.57** | **13.69** |
| **1** | Diag. | 5 | 4 | 10 | 8 | 12.86 | 8.57 |
| **2** | Diag. | 5 | 13 | 4 | 46 | 25.71 | 23.81 |
| **3** | Diag. | 1 | 6 | 7 | 57 | 24.29 | 30.48 |
| **4** | Diag. | 0 | 21 | 7 | 60 | 30.00 | 31.91 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **23.22** | **23.69** |
| **1** | Unit | 0 | 35 | 0 | 103 | 50.00 | 49.05 |
| **2** | Unit | 0 | 35 | 0 | 105 | 50.00 | 50.00 |
| **3** | Unit | 0 | 34 | 0 | 103 | 48.57 | 49.05 |
| **4** | Unit | 0 | 35 | 0 | 102 | 50.00 | 48.57 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **49.64** | **49.17** |

Table F.5: Unnormalized data for ML estimation (4D).

## Normalized 4D data (Std, NoP)

| **i** | **Cov. Matrix** | **NOWC in D1(i)** | **NOWC in D2(i)** | **NOWC in TS1(i)** | **NOWC in TS2(i)** | **ER for test sets** | **ER for tr. sets** |
|---|---|---|---|---|---|---|---|
| **1** | Gen. | 2 | 12 | 5 | 29 | 20.00 | 16.19 |
| **2** | Gen. | 0 | 8 | 4 | 33 | 11.43 | 17.62 |
| **3** | Gen. | 2 | 8 | 3 | 36 | 14.29 | 18.57 |
| **4** | Gen. | 4 | 3 | 7 | 8 | 10.00 | 7.14 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **13.93** | **14.88** |
| **1** | Diag. | 3 | 16 | 5 | 51 | 27.14 | 26.67 |
| **2** | Diag. | 1 | 20 | 7 | 55 | 30.00 | 29.52 |
| **3** | Diag. | 1 | 17 | 6 | 63 | 25.71 | 32.86 |
| **4** | Diag. | 5 | 3 | 10 | 12 | 11.43 | 10.48 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **23.57** | **24.88** |
| **1** | Unit | 8 | 9 | 18 | 23 | 24.29 | 19.52 |
| **2** | Unit | 5 | 3 | 23 | 28 | 11.43 | 24.29 |
| **3** | Unit | 7 | 7 | 20 | 24 | 20.00 | 20.95 |
| **4** | Unit | 3 | 4 | 8 | 9 | 10.00 | 8.10 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **16.43** | **18.21** |

Table F.6: Normalized data for ML estimation (4D).

## F.2 Parzen Window Technique

### 2D data (Std, NoP)

| i | WW (h1) | NOWC in D1(i) | NOWC in D2(i) | NOWC in TS1(i) | NOWC in TS2(i) | ER for test sets | ER for tr. sets |
|---|---|---|---|---|---|---|---|
| **1** | 0.01 | 3 | 21 | 0 | 0 | 34.29 | 0.00 |
| **2** | 0.01 | 5 | 18 | 0 | 0 | 32.86 | 0.00 |
| **3** | 0.01 | 5 | 16 | 0 | 0 | 30.00 | 0.00 |
| **4** | 0.01 | 2 | 15 | 0 | 0 | 24.29 | 0.00 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **30.36** | **0.00** |
| **1** | 0.1 | 3 | 9 | 1 | 1 | 17.14 | 0.95 |
| **2** | 0.1 | 5 | 3 | 1 | 2 | 11.43 | 1.43 |
| **3** | 0.1 | 6 | 3 | 1 | 2 | 12.86 | 1.43 |
| **4** | 0.1 | 3 | 8 | 0 | 1 | 15.71 | 0.48 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **14.29** | **1.07** |
| **1** | 1 | 1 | 5 | 5 | 7 | 8.57 | 5.71 |
| **2** | 1 | 4 | 3 | 4 | 11 | 10.00 | 7.14 |
| **3** | 1 | 5 | 4 | 6 | 11 | 12.86 | 8.10 |
| **4** | 1 | 4 | 8 | 5 | 7 | 17.14 | 5.71 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **12.14** | **6.67** |
| **1** | 5 | 2 | 5 | 9 | 14 | 10.00 | 10.95 |
| **2** | 5 | 4 | 8 | 9 | 21 | 17.14 | 14.29 |
| **3** | 5 | 5 | 6 | 7 | 19 | 15.71 | 12.38 |
| **4** | 5 | 1 | 9 | 9 | 15 | 14.29 | 11.43 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **14.29** | **12.26** |
| **1** | 6 | 2 | 6 | 10 | 17 | 11.43 | 12.86 |
| **2** | 6 | 4 | 10 | 10 | 19 | 20.00 | 13.81 |
| **3** | 6 | 5 | 6 | 7 | 22 | 15.71 | 13.81 |
| **4** | 6 | 2 | 9 | 10 | 20 | 15.71 | 14.29 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **15.71** | **13.69** |
| **1** | 7 | 2 | 7 | 12 | 16 | 12.86 | 13.33 |
| **2** | 7 | 4 | 10 | 13 | 21 | 20.00 | 16.19 |
| **3** | 7 | 5 | 5 | 9 | 25 | 14.29 | 16.19 |
| **4** | 7 | 2 | 10 | 12 | 22 | 17.14 | 16.19 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **16.07** | **15.48** |
| **1** | 8 | 3 | 6 | 14 | 19 | 12.86 | 15.71 |
| **2** | 8 | 4 | 11 | 14 | 21 | 21.43 | 16.67 |
| **3** | 8 | 7 | 6 | 10 | 26 | 18.57 | 17.14 |
| **4** | 8 | 2 | 11 | 13 | 24 | 18.57 | 17.62 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **17.86** | **16.79** |

Table F.7: Unnormalized data (2D) for Parzen window estimation.

### Normalized 2D data (Std, NoP)

| i | WW (h1) | NOWC in D1(i) | NOWC in D2(i) | NOWC in TS1(i) | NOWC in TS2(i) | ER for test sets | ER for tr. sets |
|---|---|---|---|---|---|---|---|
| **1** | 0.01 | 1 | 24 | 0 | 0 | 35.71 | 0.00 |
| **2** | 0.01 | 5 | 4 | 0 | 1 | 12.86 | 0.48 |
| **3** | 0.01 | 6 | 10 | 1 | 2 | 22.86 | 1.43 |
| **4** | 0.01 | 6 | 4 | 0 | 1 | 14.29 | 0.48 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **21.43** | **0.60** |
| **1** | 0.1 | 3 | 8 | 1 | 1 | 15.71 | 0.95 |
| **2** | 0.1 | 6 | 3 | 4 | 4 | 12.86 | 3.81 |
| **3** | 0.1 | 6 | 7 | 4 | 5 | 18.57 | 4.29 |
| **4** | 0.1 | 6 | 2 | 4 | 8 | 11.43 | 5.71 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **14.64** | **3.69** |
| **1** | 1 | 4 | 6 | 7 | 8 | 14.29 | 7.14 |
| **2** | 1 | 4 | 4 | 9 | 18 | 11.43 | 12.86 |
| **3** | 1 | 2 | 12 | 9 | 17 | 20.00 | 12.38 |
| **4** | 1 | 5 | 7 | 9 | 18 | 17.14 | 12.86 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **15.71** | **11.31** |
| **1** | 2 | 1 | 6 | 7 | 12 | 10.00 | 9.05 |
| **2** | 2 | 6 | 4 | 15 | 24 | 14.29 | 18.57 |
| **3** | 2 | 7 | 15 | 17 | 30 | 31.43 | 22.38 |
| **4** | 2 | 9 | 10 | 15 | 25 | 27.14 | 19.05 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **20.71** | **17.26** |
| **1** | 3 | 0 | 5 | 7 | 14 | 7.14 | 10.00 |
| **2** | 3 | 10 | 5 | 27 | 27 | 21.43 | 25.71 |
| **3** | 3 | 11 | 16 | 23 | 33 | 38.57 | 26.67 |
| **4** | 3 | 9 | 8 | 20 | 26 | 24.29 | 21.90 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **22.86** | **21.07** |
| **1** | 4 | 0 | 6 | 7 | 15 | 8.57 | 10.48 |
| **2** | 4 | 14 | 5 | 41 | 28 | 27.14 | 32.86 |
| **3** | 4 | 12 | 16 | 26 | 36 | 40.00 | 29.52 |
| **4** | 4 | 12 | 8 | 34 | 25 | 28.57 | 28.10 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **26.07** | **25.24** |
| **1** | 5 | 0 | 8 | 7 | 16 | 11.43 | 10.95 |
| **2** | 5 | 18 | 5 | 48 | 28 | 32.86 | 36.19 |
| **3** | 5 | 13 | 16 | 28 | 39 | 41.43 | 31.90 |
| **4** | 5 | 17 | 8 | 38 | 25 | 35.71 | 30.00 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **30.36** | **27.26** |

Table F.8: Normalized data (2D) for Parzen window estimation.

## 3D data (Mean, Std, NoP)

| **i** | **WW (h1)** | **NOWC in D1(i)** | **NOWC in D2(i)** | **NOWC in TS1(i)** | **NOWC in TS2(i)** | **ER for test sets** | **ER for tr. sets** |
|---|---|---|---|---|---|---|---|
| **1** | 1 | 3 | 3 | 0 | 0 | 8.57 | 0.00 |
| **2** | 1 | 2 | 5 | 0 | 0 | 10.00 | 0.00 |
| **3** | 1 | 3 | 2 | 0 | 0 | 7.14 | 0.00 |
| **4** | 1 | 3 | 3 | 0 | 0 | 8.57 | 0.00 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.57** | **0.00** |
| **1** | 5 | 2 | 3 | 3 | 3 | 7.14 | 2.86 |
| **2** | 5 | 2 | 5 | 2 | 3 | 10.00 | 2.38 |
| **3** | 5 | 5 | 1 | 3 | 4 | 8.57 | 3.33 |
| **4** | 5 | 2 | 2 | 3 | 2 | 5.71 | 2.38 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **7.86** | **2.74** |
| **1** | 10 | 4 | 2 | 7 | 6 | 8.57 | 6.19 |
| **2** | 10 | 2 | 6 | 8 | 4 | 11.43 | 5.71 |
| **3** | 10 | 4 | 1 | 5 | 8 | 7.14 | 6.19 |
| **4** | 10 | 3 | 2 | 8 | 6 | 7.14 | 6.67 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.57** | **6.19** |
| **1** | 11 | 4 | 2 | 7 | 5 | 8.57 | 5.71 |
| **2** | 11 | 2 | 6 | 9 | 5 | 11.43 | 6.67 |
| **3** | 11 | 4 | 1 | 6 | 8 | 7.14 | 6.67 |
| **4** | 11 | 3 | 2 | 9 | 6 | 7.14 | 7.14 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.57** | **6.55** |
| **1** | 12 | 4 | 2 | 9 | 5 | 8.57 | 6.67 |
| **2** | 12 | 2 | 6 | 10 | 5 | 11.43 | 7.14 |
| **3** | 12 | 4 | 2 | 8 | 9 | 8.57 | 8.10 |
| **4** | 12 | 3 | 2 | 9 | 10 | 7.14 | 9.05 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.93** | **7.74** |
| **1** | 13 | 4 | 2 | 9 | 7 | 8.57 | 7.62 |
| **2** | 13 | 2 | 7 | 11 | 5 | 12.86 | 7.62 |
| **3** | 13 | 6 | 2 | 9 | 11 | 11.43 | 9.52 |
| **4** | 13 | 3 | 2 | 9 | 10 | 7.14 | 9.05 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **10.00** | **8.45** |
| **1** | 14 | 5 | 2 | 9 | 7 | 10.00 | 7.62 |
| **2** | 14 | 2 | 8 | 13 | 5 | 14.29 | 8.57 |
| **3** | 14 | 6 | 2 | 9 | 11 | 11.43 | 9.52 |
| **4** | 14 | 3 | 2 | 9 | 10 | 7.14 | 9.05 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **10.71** | **8.69** |
| **1** | 15 | 5 | 2 | 11 | 8 | 10.00 | 9.05 |
| **2** | 15 | 3 | 8 | 13 | 5 | 15.71 | 8.57 |
| **3** | 15 | 6 | 2 | 10 | 11 | 11.43 | 10.00 |
| **4** | 15 | 3 | 2 | 12 | 10 | 7.14 | 10.48 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Mean Error Rate (test sets/training sets):** | | | | | | **11.07** | **9.52** |
| **1** | 16 | 5 | 2 | 12 | 9 | 10.00 | 10.00 |
| **2** | 16 | 3 | 8 | 14 | 5 | 15.71 | 9.05 |
| **3** | 16 | 6 | 2 | 11 | 11 | 11.43 | 10.48 |
| **4** | 16 | 3 | 2 | 12 | 10 | 7.14 | 10.48 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **11.07** | **10.00** |
| **1** | 17 | 5 | 2 | 13 | 9 | 10.00 | 10.48 |
| **2** | 17 | 4 | 8 | 14 | 5 | 17.14 | 9.05 |
| **3** | 17 | 7 | 2 | 12 | 11 | 12.86 | 10.95 |
| **4** | 17 | 3 | 2 | 12 | 11 | 7.14 | 10.95 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **11.79** | **10.36** |
| **1** | 18 | 5 | 2 | 13 | 9 | 10.00 | 10.48 |
| **2** | 18 | 4 | 8 | 14 | 5 | 17.14 | 9.05 |
| **3** | 18 | 7 | 2 | 12 | 11 | 12.86 | 10.95 |
| **4** | 18 | 3 | 2 | 12 | 12 | 7.14 | 11.43 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **11.79** | **10.48** |
| **1** | 19 | 5 | 2 | 13 | 9 | 10.00 | 10.48 |
| **2** | 19 | 4 | 8 | 15 | 5 | 17.14 | 9.52 |
| **3** | 19 | 7 | 2 | 12 | 11 | 12.86 | 10.95 |
| **4** | 19 | 3 | 2 | 13 | 12 | 7.14 | 11.90 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **11.79** | **10.71** |
| **1** | 20 | 5 | 2 | 14 | 9 | 10.00 | 10.95 |
| **2** | 20 | 4 | 8 | 15 | 5 | 17.14 | 9.52 |
| **3** | 20 | 7 | 2 | 12 | 11 | 12.86 | 10.95 |
| **4** | 20 | 3 | 2 | 14 | 12 | 7.14 | 12.38 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **11.79** | **10.95** |
| **1** | 21 | 5 | 2 | 14 | 9 | 10.00 | 10.95 |
| **2** | 21 | 4 | 8 | 15 | 5 | 17.14 | 9.52 |
| **3** | 21 | 7 | 2 | 12 | 11 | 12.86 | 10.95 |
| **4** | 21 | 3 | 2 | 14 | 12 | 7.14 | 12.38 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **11.79** | **10.95** |
| **1** | 22 | 5 | 2 | 14 | 9 | 10.00 | 10.95 |
| **2** | 22 | 4 | 8 | 15 | 5 | 17.14 | 9.52 |
| **3** | 22 | 7 | 2 | 12 | 11 | 12.86 | 10.95 |
| **4** | 22 | 3 | 2 | 14 | 12 | 7.14 | 12.38 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **11.79** | **10.95** |
| **1** | 24 | 5 | 2 | 14 | 9 | 10.00 | 10.95 |
| **2** | 24 | 4 | 8 | 15 | 6 | 17.14 | 10.00 |
| **3** | 24 | 7 | 2 | 12 | 12 | 12.86 | 11.43 |
| **4** | 24 | 3 | 2 | 14 | 12 | 7.14 | 12.38 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **11.79** | **11.19** |
| **1** | 26 | 5 | 2 | 14 | 9 | 10.00 | 10.95 |
| **2** | 26 | 4 | 8 | 15 | 6 | 17.14 | 10.00 |
| **3** | 26 | 7 | 2 | 11 | 12 | 12.86 | 10.95 |
| **4** | 26 | 3 | 2 | 14 | 12 | 7.14 | 12.38 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **11.79** | **11.07** |

Table F.9: Unnormalized data (3D) for Parzen window estimation.

### Normalized 3D data (Mean, Std, NoP)

| **i** | **WW (h1)** | **NOWC in D1(i)** | **NOWC in D2(i)** | **NOWC in TS1(i)** | **NOWC in TS2(i)** | **ER for test sets** | **ER for tr. sets** |
|---|---|---|---|---|---|---|---|
| **1** | 0.1 | 0 | 15 | 0 | 0 | 21.43 | 0.00 |
| **2** | 0.1 | 4 | 4 | 0 | 0 | 11.43 | 0.00 |
| **3** | 0.1 | 4 | 2 | 0 | 0 | 8.57 | 0.00 |
| **4** | 0.1 | 4 | 4 | 0 | 0 | 11.43 | 0.00 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **13.21** | **0.00** |
| **1** | 1 | 1 | 6 | 0 | 1 | 10.00 | 0.48 |
| **2** | 1 | 4 | 3 | 1 | 3 | 10.00 | 1.90 |
| **3** | 1 | 3 | 0 | 4 | 4 | 4.29 | 3.81 |
| **4** | 1 | 4 | 3 | 2 | 2 | 10.00 | 1.9 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.57** | **2.02** |
| **1** | 2 | 1 | 5 | 0 | 1 | 8.57 | 0.48 |
| **2** | 2 | 5 | 3 | 7 | 5 | 11.43 | 5.71 |
| **3** | 2 | 4 | 2 | 9 | 4 | 8.57 | 6.19 |
| **4** | 2 | 2 | 3 | 7 | 2 | 7.14 | 4.29 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.93** | **4.17** |
| **1** | 3 | 1 | 5 | 2 | 2 | 8.57 | 1.90 |
| **2** | 3 | 7 | 3 | 11 | 6 | 14.29 | 8.10 |
| **3** | 3 | 3 | 1 | 10 | 5 | 5.71 | 7.14 |
| **4** | 3 | 2 | 3 | 11 | 4 | 7.14 | 7.14 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.93** | **6.07** |
| **1** | 4 | 1 | 5 | 3 | 2 | 8.57 | 2.38 |
| **2** | 4 | 7 | 2 | 14 | 6 | 12.86 | 9.52 |
| **3** | 4 | 4 | 1 | 15 | 6 | 7.14 | 10.00 |
| **4** | 4 | 4 | 3 | 15 | 5 | 10.00 | 9.52 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **9.64** | **7.86** |
| **1** | 4.5 | 1 | 5 | 3 | 2 | 8.57 | 2.38 |
| **2** | 4.5 | 8 | 2 | 15 | 6 | 14.29 | 10.00 |
| **3** | 4.5 | 4 | 1 | 17 | 5 | 7.14 | 10.48 |
| **4** | 4.5 | 5 | 3 | 16 | 4 | 11.43 | 9.52 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **10.36** | **8.10** |
| **1** | 5 | 0 | 5 | 3 | 2 | 7.14 | 2.38 |
| **2** | 5 | 10 | 4 | 15 | 7 | 20.00 | 10.48 |
| **3** | 5 | 4 | 1 | 18 | 5 | 7.14 | 10.95 |
| **4** | 5 | 5 | 3 | 17 | 4 | 11.43 | 10.00 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Mean Error Rate (test sets/training sets):** | | | | | | **11.43** | **8.45** |

Table F.10: Normalized data (3D) for Parzen window estimation.

### 4D data (Area, Mean, Std, NoP)

| **i** | **WW (h1)** | **NOWC in D1(i)** | **NOWC in D2(i)** | **NOWC in TS1(i)** | **NOWC in TS2(i)** | **ER for test sets** | **ER for tr. sets** |
|---|---|---|---|---|---|---|---|
| **1** | 1 | 1 | 29 | 0 | 0 | 42.86 | 0.00 |
| **2** | 1 | 0 | 32 | 0 | 0 | 45.71 | 0.00 |
| **3** | 1 | 1 | 30 | 0 | 0 | 44.29 | 0.00 |
| **4** | 1 | 1 | 29 | 0 | 0 | 42.86 | 0.00 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **43.93** | **0.00** |
| **1** | 10 | 9 | 10 | 0 | 0 | 27.14 | 0.00 |
| **2** | 10 | 6 | 11 | 0 | 0 | 24.29 | 0.00 |
| **3** | 10 | 7 | 11 | 0 | 0 | 25.71 | 0.00 |
| **4** | 10 | 3 | 14 | 0 | 0 | 24.29 | 0.00 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **25.36** | **0.00** |
| **1** | 50 | 9 | 6 | 0 | 0 | 21.43 | 0.00 |
| **2** | 50 | 6 | 8 | 2 | 1 | 20.00 | 1.43 |
| **3** | 50 | 8 | 5 | 1 | 0 | 18.57 | 0.48 |
| **4** | 50 | 3 | 6 | 0 | 2 | 12.86 | 0.95 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **18.21** | **0.71** |
| **1** | 100 | 9 | 6 | 2 | 0 | 21.43 | 0.95 |
| **2** | 100 | 6 | 7 | 3 | 3 | 18.57 | 2.86 |
| **3** | 100 | 8 | 4 | 2 | 1 | 17.14 | 1.43 |
| **4** | 100 | 3 | 5 | 1 | 3 | 11.43 | 1.90 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **17.14** | **1.79** |
| **1** | 200 | 9 | 6 | 4 | 4 | 21.43 | 3.81 |
| **2** | 200 | 6 | 6 | 7 | 6 | 17.14 | 6.19 |
| **3** | 200 | 7 | 5 | 4 | 11 | 17.14 | 7.14 |
| **4** | 200 | 2 | 5 | 7 | 8 | 10.00 | 7.14 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **16.43** | **6.07** |
| **1** | 400 | 7 | 6 | 7 | 10 | 18.57 | 8.10 |
| **2** | 400 | 6 | 5 | 12 | 10 | 15.71 | 10.48 |
| **3** | 400 | 6 | 4 | 8 | 12 | 14.29 | 9.52 |
| **4** | 400 | 1 | 5 | 8 | 14 | 8.57 | 10.48 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **14.29** | **9.64** |
| **1** | 600 | 7 | 6 | 6 | 10 | 18.57 | 7.62 |
| **2** | 600 | 7 | 6 | 13 | 9 | 18.57 | 10.48 |
| **3** | 600 | 6 | 3 | 11 | 12 | 12.86 | 10.95 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **4** | 600 | 1 | 5 | 11 | 12 | 8.57 | 10.95 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **14.64** | **10.00** |
| **1** | 800 | 7 | 7 | 7 | 12 | 20.00 | 9.05 |
| **2** | 800 | 7 | 5 | 13 | 12 | 17.14 | 11.90 |
| **3** | 800 | 6 | 3 | 11 | 13 | 12.86 | 11.43 |
| **4** | 800 | 1 | 5 | 14 | 12 | 8.57 | 12.38 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **14.64** | **11.19** |
| **1** | 1000 | 7 | 7 | 7 | 12 | 20.00 | 9.05 |
| **2** | 1000 | 6 | 4 | 13 | 11 | 14.29 | 11.43 |
| **3** | 1000 | 6 | 3 | 10 | 15 | 12.86 | 11.90 |
| **4** | 1000 | 1 | 5 | 15 | 11 | 8.57 | 12.38 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **13.93** | **11.19** |
| **1** | 1200 | 7 | 7 | 7 | 12 | 20.00 | 9.05 |
| **2** | 1200 | 4 | 4 | 12 | 11 | 11.43 | 10.95 |
| **3** | 1200 | 4 | 3 | 10 | 15 | 10.00 | 11.90 |
| **4** | 1200 | 1 | 5 | 14 | 11 | 8.57 | 11.90 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **12.50** | **10.95** |
| **1** | 1400 | 7 | 7 | 7 | 12 | 20.00 | 9.05 |
| **2** | 1400 | 3 | 4 | 11 | 14 | 10.00 | 11.90 |
| **3** | 1400 | 4 | 3 | 10 | 15 | 10.00 | 11.90 |
| **4** | 1400 | 0 | 6 | 13 | 14 | 8.57 | 12.86 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **12.14** | **11.43** |
| **1** | 1600 | 7 | 7 | 7 | 12 | 20.00 | 9.05 |
| **2** | 1600 | 3 | 4 | 11 | 14 | 10.00 | 11.90 |
| **3** | 1600 | 4 | 3 | 10 | 15 | 10.00 | 11.90 |
| **4** | 1600 | 0 | 6 | 13 | 14 | 8.57 | 12.86 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **12.14** | **11.43** |
| **1** | 1800 | 7 | 7 | 7 | 12 | 20.00 | 9.05 |
| **2** | 1800 | 3 | 4 | 11 | 14 | 10.00 | 11.90 |
| **3** | 1800 | 4 | 3 | 10 | 15 | 10.00 | 11.90 |
| **4** | 1800 | 0 | 6 | 13 | 14 | 8.57 | 12.86 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **12.14** | **11.43** |
| **1** | 2000 | 7 | 7 | 7 | 12 | 20.00 | 9.05 |
| **2** | 2000 | 3 | 4 | 11 | 14 | 10.00 | 11.90 |
| **3** | 2000 | 4 | 3 | 10 | 15 | 10.00 | 11.90 |
| **4** | 2000 | 0 | 6 | 13 | 15 | 8.57 | 13.33 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **12.14** | **11.55** |
| **1** | 2200 | 7 | 7 | 7 | 12 | 20.00 | 9.05 |
| **2** | 2200 | 3 | 4 | 11 | 14 | 10.00 | 11.90 |
| **3** | 2200 | 4 | 3 | 10 | 15 | 10.00 | 11.90 |
| **4** | 2200 | 0 | 6 | 13 | 15 | 8.57 | 13.33 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **12.14** | **11.55** |

Table F.11: Unnormalized data (4D) for Parzen window estimation.

### Normalized 4D data (Area, Mean, Std, NoP)

| i | WW (h1) | NOWC in D1(i) | NOWC in D2(i) | NOWC in TS1(i) | NOWC in TS2(i) | ER for test sets | ER for tr. sets |
|---|---|---|---|---|---|---|---|
| **1** | 1 | 4 | 6 | 1 | 2 | 14.29 | 1.43 |
| **2** | 1 | 2 | 4 | 0 | 1 | 8.57 | 0.48 |
| **3** | 1 | 1 | 4 | 2 | 2 | 7.14 | 1.90 |
| **4** | 1 | 4 | 1 | 3 | 4 | 7.14 | 3.33 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **9.29** | **1.79** |
| **1** | 2 | 4 | 4 | 8 | 2 | 11.43 | 4.76 |
| **2** | 2 | 2 | 3 | 0 | 2 | 7.14 | 0.95 |
| **3** | 2 | 2 | 3 | 7 | 5 | 7.14 | 5.71 |
| **4** | 2 | 5 | 2 | 8 | 4 | 10.00 | 5.71 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.93** | **4.29** |
| **1** | 3 | 4 | 4 | 10 | 4 | 11.43 | 6.67 |
| **2** | 3 | 1 | 3 | 2 | 2 | 5.71 | 1.90 |
| **3** | 3 | 2 | 3 | 11 | 4 | 7.14 | 7.14 |
| **4** | 3 | 5 | 2 | 10 | 4 | 10.00 | 6.67 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.57** | **5.60** |
| **1** | 4 | 6 | 3 | 12 | 4 | 12.86 | 7.62 |
| **2** | 4 | 1 | 3 | 2 | 5 | 5.71 | 3.33 |
| **3** | 4 | 3 | 2 | 16 | 4 | 7.14 | 9.52 |
| **4** | 4 | 6 | 2 | 14 | 4 | 11.43 | 8.57 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **9.29** | **7.26** |
| **1** | 5 | 7 | 2 | 14 | 4 | 12.86 | 8.57 |
| **2** | 5 | 1 | 2 | 3 | 6 | 4.29 | 4.29 |
| **3** | 5 | 4 | 2 | 18 | 5 | 8.57 | 10.95 |
| **4** | 5 | 6 | 1 | 16 | 4 | 10.00 | 9.52 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.93** | **8.33** |
| **1** | 5.5 | 7 | 2 | 15 | 5 | 12.86 | 9.52 |
| **2** | 5.5 | 1 | 2 | 4 | 7 | 4.29 | 5.24 |
| **3** | 5.5 | 4 | 3 | 18 | 4 | 10.00 | 10.48 |
| **4** | 5.5 | 7 | 1 | 16 | 5 | 11.43 | 10.00 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **9.64** | **8.81** |
| **1** | 6 | 7 | 3 | 15 | 5 | 14.29 | 9.52 |
| **2** | 6 | 1 | 2 | 4 | 8 | 4.29 | 5.71 |
| **3** | 6 | 4 | 3 | 19 | 4 | 10.00 | 10.95 |
| **4** | 6 | 7 | 1 | 16 | 5 | 11.43 | 10.00 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **10.00** | **9.05** |
| **1** | 6.5 | 6 | 3 | 16 | 5 | 12.86 | 10.00 |
| **2** | 6.5 | 1 | 2 | 5 | 8 | 4.29 | 6.19 |
| **3** | 6.5 | 4 | 3 | 19 | 4 | 10.00 | 10.95 |
| **4** | 6.5 | 7 | 1 | 16 | 5 | 11.43 | 10.00 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Mean Error Rate (test sets/training sets):** | | | | | | **9.64** | **9.29** |
| **1** | 7 | 6 | 3 | 16 | 5 | 12.86 | 10.00 |
| **2** | 7 | 1 | 2 | 5 | 8 | 4.29 | 6.19 |
| **3** | 7 | 4 | 3 | 19 | 4 | 10.00 | 10.95 |
| **4** | 7 | 7 | 1 | 16 | 6 | 11.43 | 10.48 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **9.64** | **9.40** |
| **1** | 7.5 | 6 | 3 | 16 | 5 | 12.86 | 10.00 |
| **2** | 7.5 | 2 | 2 | 6 | 8 | 5.71 | 6.67 |
| **3** | 7.5 | 4 | 3 | 21 | 3 | 10.00 | 11.43 |
| **4** | 7.5 | 8 | 1 | 17 | 6 | 12.86 | 10.95 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **10.36** | **9.76** |
| **1** | 8 | 6 | 3 | 16 | 5 | 12.86 | 10.00 |
| **2** | 8 | 2 | 2 | 5 | 8 | 5.71 | 6.19 |
| **3** | 8 | 4 | 3 | 20 | 3 | 10.00 | 10.95 |
| **4** | 8 | 8 | 1 | 17 | 8 | 12.86 | 11.90 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **10.36** | **9.76** |
| **1** | 8.5 | 6 | 3 | 16 | 6 | 12.86 | 10.48 |
| **2** | 8.5 | 2 | 2 | 5 | 8 | 5.71 | 6.19 |
| **3** | 8.5 | 4 | 3 | 20 | 3 | 10.00 | 10.95 |
| **4** | 8.5 | 8 | 1 | 16 | 8 | 12.86 | 11.43 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **10.36** | **9.76** |
| **1** | 9 | 6 | 3 | 17 | 6 | 12.86 | 10.95 |
| **2** | 9 | 2 | 2 | 5 | 8 | 5.71 | 6.19 |
| **3** | 9 | 4 | 3 | 20 | 3 | 10.00 | 10.95 |
| **4** | 9 | 8 | 1 | 16 | 8 | 12.86 | 11.43 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **10.36** | **9.88** |
| **1** | 9.5 | 7 | 4 | 17 | 8 | 15.71 | 11.90 |
| **2** | 9.5 | 2 | 2 | 6 | 8 | 5.71 | 6.67 |
| **3** | 9.5 | 4 | 3 | 19 | 3 | 10.00 | 10.48 |
| **4** | 9.5 | 8 | 1 | 16 | 10 | 12.86 | 12.38 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **11.07** | **10.36** |
| **1** | 10 | 7 | 4 | 17 | 9 | 15.71 | 12.38 |
| **2** | 10 | 2 | 2 | 8 | 8 | 5.71 | 7.62 |
| **3** | 10 | 4 | 3 | 19 | 3 | 10.00 | 10.48 |
| **4** | 10 | 8 | 1 | 16 | 11 | 12.86 | 12.86 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **11.07** | **10.83** |

Table F.12: Normalized data (4D) for Parzen window estimation.

## F.3 $k_n$-Nearest-Neighbor Estimation

### 2D data (Std, NoP)

| **i** | **$k_n$** | **NOWC in D1(i)** | **NOWC in D2(i)** | **NOWC in TS1(i)** | **NOWC in TS2(i)** | **ER for test sets** | **ER for tr. sets** |
|---|---|---|---|---|---|---|---|
| **1** | 1 | 6 | 5 | 0 | 0 | 15.71 | 0.00 |
| **2** | 1 | 5 | 3 | 0 | 0 | 11.43 | 0.00 |
| **3** | 1 | 3 | 6 | 0 | 0 | 12.86 | 0.00 |
| **4** | 1 | 2 | 6 | 0 | 0 | 11.43 | 0.00 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **12.86** | **0.00** |
| **1** | 2 | 1 | 7 | 0 | 18 | 11.43 | 8.57 |
| **2** | 2 | 4 | 3 | 0 | 15 | 10.00 | 7.14 |
| **3** | 2 | 2 | 8 | 0 | 14 | 14.29 | 6.67 |
| **4** | 2 | 1 | 10 | 0 | 15 | 15.71 | 7.14 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **12.86** | **7.38** |
| **1** | 3 | 2 | 5 | 5 | 9 | 10.00 | 6.67 |
| **2** | 3 | 6 | 3 | 5 | 10 | 12.86 | 7.14 |
| **3** | 3 | 2 | 5 | 4 | 11 | 10.00 | 7.14 |
| **4** | 3 | 3 | 6 | 7 | 9 | 12.86 | 7.62 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **11.43** | **7.14** |
| **1** | 4 | 0 | 8 | 4 | 15 | 11.43 | 9.05 |
| **2** | 4 | 5 | 3 | 5 | 11 | 11.43 | 7.62 |
| **3** | 4 | 2 | 8 | 3 | 14 | 14.29 | 8.10 |
| **4** | 4 | 2 | 12 | 5 | 13 | 20.00 | 8.57 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **14.29** | **8.33** |
| **1** | 6 | 1 | 7 | 5 | 16 | 11.43 | 10.00 |
| **2** | 6 | 5 | 3 | 4 | 15 | 11.43 | 9.05 |
| **3** | 6 | 1 | 7 | 5 | 14 | 11.43 | 9.05 |
| **4** | 6 | 2 | 11 | 4 | 13 | 18.57 | 8.10 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **12.50** | **9.05** |
| **1** | 8 | 1 | 4 | 7 | 15 | 7.14 | 10.48 |
| **2** | 8 | 5 | 3 | 5 | 19 | 11.43 | 11.43 |
| **3** | 8 | 1 | 7 | 7 | 17 | 11.43 | 11.43 |
| **4** | 8 | 2 | 11 | 5 | 13 | 18.57 | 8.57 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **12.14** | **10.48** |
| **1** | 10 | 2 | 6 | 6 | 20 | 11.43 | 12.38 |
| **2** | 10 | 5 | 3 | 7 | 19 | 11.43 | 12.38 |
| **3** | 10 | 2 | 5 | 5 | 15 | 10.00 | 9.52 |
| **4** | 10 | 2 | 11 | 6 | 13 | 18.57 | 9.05 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **12.86** | **10.83** |
| **1** | 12 | 2 | 5 | 8 | 18 | 10.00 | 12.38 |
| **2** | 12 | 5 | 3 | 8 | 19 | 11.43 | 12.86 |
| **3** | 12 | 3 | 6 | 5 | 20 | 12.86 | 11.90 |
| **4** | 12 | 1 | 12 | 7 | 13 | 18.57 | 9.52 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Mean Error Rate (test sets/training sets):** | | | | | | **13.21** | **11.67** |
| **1** | 14 | 3 | 3 | 9 | 20 | 8.57 | 13.81 |
| **2** | 14 | 4 | 3 | 7 | 19 | 10.00 | 12.38 |
| **3** | 14 | 3 | 6 | 5 | 18 | 12.86 | 10.95 |
| **4** | 14 | 2 | 13 | 8 | 13 | 21.43 | 10.00 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **13.21** | **11.79** |
| **1** | 16 | 5 | 3 | 10 | 19 | 11.43 | 13.81 |
| **2** | 16 | 4 | 4 | 6 | 19 | 11.43 | 11.90 |
| **3** | 16 | 3 | 6 | 7 | 21 | 12.86 | 13.33 |
| **4** | 16 | 3 | 12 | 9 | 13 | 21.43 | 10.48 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **14.29** | **12.38** |
| **1** | 18 | 5 | 3 | 10 | 19 | 11.43 | 13.81 |
| **2** | 18 | 4 | 4 | 6 | 18 | 11.43 | 11.43 |
| **3** | 18 | 2 | 7 | 8 | 19 | 12.86 | 12.86 |
| **4** | 18 | 2 | 13 | 8 | 12 | 21.43 | 9.52 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **14.29** | **11.90** |

Table F.13: Unnormalized data (2D) for $k_n$-Nearest-Neighbor estimation.

## Normalized 2D data (Std, NoP)

| **i** | $\mathbf{k_n}$ | **NOWC in D1(i)** | **NOWC in D2(i)** | **NOWC in TS1(i)** | **NOWC in TS2(i)** | **ER for test sets** | **ER for tr. sets** |
|---|---|---|---|---|---|---|---|
| **1** | 1 | 4 | 5 | 0 | 0 | 12.86 | 0.00 |
| **2** | 1 | 6 | 4 | 0 | 0 | 14.29 | 0.00 |
| **3** | 1 | 7 | 4 | 0 | 0 | 15.71 | 0.00 |
| **4** | 1 | 3 | 4 | 0 | 0 | 10.00 | 0.00 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **13.21** | **0.00** |
| **1** | 2 | 2 | 7 | 0 | 15 | 12.86 | 7.14 |
| **2** | 2 | 5 | 5 | 0 | 12 | 14.29 | 5.71 |
| **3** | 2 | 2 | 5 | 0 | 16 | 10.00 | 7.62 |
| **4** | 2 | 2 | 6 | 0 | 14 | 11.43 | 6.67 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **12.14** | **6.79** |
| **1** | 3 | 2 | 5 | 4 | 10 | 10.00 | 6.67 |
| **2** | 3 | 8 | 4 | 5 | 6 | 17.14 | 5.24 |
| **3** | 3 | 4 | 3 | 8 | 10 | 10.00 | 8.57 |
| **4** | 3 | 2 | 3 | 5 | 9 | 7.14 | 6.67 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **11.07** | **6.79** |
| **1** | 4 | 2 | 5 | 3 | 14 | 10.00 | 8.10 |
| **2** | 4 | 4 | 4 | 5 | 10 | 11.43 | 7.14 |
| **3** | 4 | 1 | 6 | 5 | 16 | 10.00 | 10.00 |
| **4** | 4 | 2 | 3 | 4 | 16 | 7.14 | 9.52 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **9.64** | **8.69** |
| **1** | 5 | 3 | 5 | 6 | 12 | 11.43 | 8.57 |
| **2** | 5 | 7 | 3 | 6 | 9 | 14.29 | 7.14 |
| **3** | 5 | 1 | 5 | 6 | 13 | 8.57 | 9.04 |
| **4** | 5 | 3 | 3 | 7 | 12 | 8.57 | 9.04 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **10.71** | **8.45** |
| **1** | 6 | 2 | 7 | 5 | 17 | 12.86 | 10.48 |
| **2** | 6 | 6 | 4 | 5 | 11 | 14.29 | 7.62 |
| **3** | 6 | 0 | 6 | 6 | 15 | 8.57 | 10.00 |
| **4** | 6 | 3 | 4 | 3 | 16 | 10.00 | 9.05 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **11.43** | **9.29** |
| **1** | 7 | 5 | 6 | 7 | 13 | 15.71 | 9.52 |
| **2** | 7 | 7 | 3 | 7 | 8 | 14.29 | 7.14 |
| **3** | 7 | 1 | 6 | 8 | 13 | 10.00 | 10.00 |
| **4** | 7 | 3 | 4 | 7 | 11 | 10.00 | 8.57 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **12.50** | **8.81** |

Table F.14: Normalized data (2D) for $k_n$-Nearest-Neighbor estimation.

## 3D data (Mean, Std, NoP)

| **i** | **$k_n$** | **NOWC in D1(i)** | **NOWC in D2(i)** | **NOWC in TS1(i)** | **NOWC in TS2(i)** | **ER for test sets** | **ER for tr. sets** |
|---|---|---|---|---|---|---|---|
| **1** | 1 | 3 | 1 | 0 | 0 | 5.71 | 0.00 |
| **2** | 1 | 2 | 3 | 0 | 0 | 7.14 | 0.00 |
| **3** | 1 | 2 | 6 | 0 | 0 | 11.43 | 0.00 |
| **4** | 1 | 3 | 5 | 0 | 0 | 11.43 | 0.00 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.93** | **0.00** |
| **1** | 2 | 2 | 3 | 0 | 11 | 7.14 | 5.24 |
| **2** | 2 | 1 | 3 | 0 | 11 | 5.71 | 5.24 |
| **3** | 2 | 1 | 6 | 0 | 7 | 10.00 | 3.33 |
| **4** | 2 | 2 | 5 | 0 | 8 | 10.00 | 3.81 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.21** | **4.40** |
| **1** | 4 | 4 | 2 | 2 | 7 | 8.57 | 4.29 |
| **2** | 4 | 1 | 3 | 2 | 9 | 5.71 | 5.24 |
| **3** | 4 | 0 | 6 | 1 | 8 | 8.57 | 4.29 |
| **4** | 4 | 2 | 6 | 2 | 7 | 11.43 | 4.29 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.57** | **4.52** |
| **1** | 8 | 4 | 2 | 5 | 7 | 8.57 | 5.71 |
| **2** | 8 | 2 | 2 | 7 | 11 | 5.71 | 8.57 |
| **3** | 8 | 1 | 8 | 6 | 11 | 12.86 | 8.10 |
| **4** | 8 | 1 | 5 | 8 | 10 | 8.57 | 8.57 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.93** | **7.74** |
| **1** | 9 | 7 | 1 | 7 | 7 | 11.43 | 6.67 |
| **2** | 9 | 3 | 2 | 8 | 10 | 7.14 | 8.57 |
| **3** | 9 | 1 | 8 | 8 | 10 | 12.86 | 8.57 |
| **4** | 9 | 4 | 5 | 9 | 8 | 12.86 | 8.10 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **11.07** | **7.98** |
| **1** | 10 | 5 | 1 | 7 | 9 | 8.57 | 7.62 |
| **2** | 10 | 2 | 3 | 6 | 13 | 7.14 | 9.05 |
| **3** | 10 | 1 | 8 | 7 | 11 | 12.86 | 8.57 |
| **4** | 10 | 2 | 6 | 8 | 9 | 11.43 | 8.10 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **10.00** | **8.33** |
| **1** | 12 | 4 | 1 | 7 | 8 | 7.14 | 7.14 |
| **2** | 12 | 3 | 2 | 7 | 11 | 7.14 | 8.57 |
| **3** | 12 | 1 | 8 | 8 | 12 | 12.86 | 9.52 |
| **4** | 12 | 3 | 6 | 8 | 9 | 12.86 | 8.10 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **10.00** | **8.33** |
| **1** | 14 | 8 | 2 | 7 | 8 | 14.29 | 7.14 |
| **2** | 14 | 3 | 2 | 8 | 10 | 7.14 | 8.57 |
| **3** | 14 | 2 | 7 | 8 | 11 | 12.86 | 9.05 |
| **4** | 14 | 3 | 6 | 8 | 9 | 12.86 | 8.10 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **11.79** | **8.21** |

Table F.15: Unnormalized data (3D) for $k_n$-Nearest-Neighbor estimation.

## Normalized 3D data (Mean, Std, NoP)

| **i** | **$k_n$** | **NOWC in D1(i)** | **NOWC in D2(i)** | **NOWC in TS1(i)** | **NOWC in TS2(i)** | **ER for test sets** | **ER for tr. sets** |
|---|---|---|---|---|---|---|---|
| **1** | 1 | 2 | 3 | 0 | 0 | 7.14 | 0.00 |
| **2** | 1 | 3 | 4 | 0 | 0 | 10.00 | 0.00 |
| **3** | 1 | 4 | 3 | 0 | 0 | 10.00 | 0.00 |
| **4** | 1 | 5 | 0 | 0 | 0 | 7.14 | 0.00 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.57** | **0.00** |
| **1** | 2 | 0 | 6 | 0 | 7 | 8.57 | 3.33 |
| **2** | 2 | 2 | 4 | 0 | 8 | 8.57 | 3.81 |
| **3** | 2 | 3 | 5 | 0 | 8 | 11.43 | 3.81 |
| **4** | 2 | 2 | 2 | 0 | 8 | 5.71 | 3.81 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.57** | **3.69** |
| **1** | 3 | 0 | 5 | 1 | 6 | 7.14 | 3.33 |
| **2** | 3 | 3 | 4 | 4 | 6 | 10.00 | 4.76 |
| **3** | 3 | 5 | 3 | 7 | 4 | 11.43 | 5.24 |
| **4** | 3 | 4 | 1 | 8 | 6 | 7.14 | 6.67 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.93** | **5.00** |
| **1** | 4 | 0 | 5 | 1 | 7 | 7.14 | 3.81 |
| **2** | 4 | 3 | 4 | 3 | 7 | 10.00 | 4.76 |
| **3** | 4 | 5 | 3 | 4 | 8 | 11.43 | 5.71 |
| **4** | 4 | 2 | 3 | 4 | 8 | 7.14 | 5.71 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.93** | **5.00** |
| **1** | 5 | 1 | 5 | 3 | 6 | 8.57 | 4.29 |
| **2** | 5 | 4 | 3 | 5 | 4 | 10.00 | 4.29 |
| **3** | 5 | 5 | 3 | 7 | 4 | 11.43 | 5.24 |
| **4** | 5 | 3 | 1 | 6 | 5 | 5.71 | 5.24 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.93** | **4.76** |
| **1** | 6 | 1 | 5 | 2 | 6 | 8.57 | 3.81 |
| **2** | 6 | 3 | 5 | 4 | 7 | 11.43 | 5.24 |
| **3** | 6 | 3 | 3 | 5 | 6 | 8.57 | 5.24 |
| **4** | 6 | 3 | 3 | 5 | 10 | 8.57 | 7.14 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **9.29** | **5.36** |
| **1** | 7 | 1 | 5 | 2 | 6 | 8.57 | 3.81 |
| **2** | 7 | 3 | 2 | 10 | 6 | 7.14 | 7.62 |
| **3** | 7 | 6 | 3 | 8 | 6 | 12.86 | 6.67 |
| **4** | 7 | 3 | 2 | 8 | 9 | 7.14 | 8.10 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.93** | **6.55** |
| **1** | 8 | 1 | 5 | 2 | 6 | 8.57 | 3.81 |
| **2** | 8 | 3 | 4 | 9 | 9 | 10.00 | 8.57 |
| **3** | 8 | 5 | 3 | 6 | 7 | 11.43 | 6.19 |
| **4** | 8 | 2 | 3 | 8 | 10 | 7.14 | 8.57 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Mean Error Rate (test sets/training sets):** | | | | | | **9.29** | **6.79** |
| **1** | 9 | 1 | 5 | 2 | 5 | 8.57 | 3.33 |
| **2** | 9 | 4 | 2 | 10 | 5 | 8.57 | 7.14 |
| **3** | 9 | 6 | 3 | 8 | 6 | 12.86 | 6.67 |
| **4** | 9 | 2 | 1 | 9 | 10 | 4.29 | 9.05 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.57** | **6.55** |
| **1** | 10 | 1 | 5 | 2 | 6 | 8.57 | 3.81 |
| **2** | 10 | 3 | 2 | 10 | 5 | 7.14 | 7.14 |
| **3** | 10 | 4 | 3 | 7 | 7 | 10.00 | 6.67 |
| **4** | 10 | 2 | 2 | 9 | 10 | 5.71 | 9.05 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **7.86** | **6.67** |
| **1** | 11 | 1 | 5 | 2 | 5 | 8.57 | 3.33 |
| **2** | 11 | 3 | 2 | 10 | 5 | 7.14 | 7.14 |
| **3** | 11 | 5 | 3 | 8 | 6 | 11.43 | 6.67 |
| **4** | 11 | 3 | 1 | 9 | 10 | 5.71 | 9.05 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.21** | **6.55** |
| **1** | 12 | 1 | 5 | 2 | 5 | 8.57 | 3.33 |
| **2** | 12 | 3 | 2 | 9 | 7 | 7.14 | 7.62 |
| **3** | 12 | 4 | 3 | 8 | 8 | 10.00 | 7.62 |
| **4** | 12 | 3 | 2 | 8 | 12 | 7.14 | 9.52 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.21** | **7.02** |
| **1** | 13 | 1 | 5 | 2 | 5 | 8.57 | 3.33 |
| **2** | 13 | 5 | 2 | 10 | 4 | 10.00 | 6.67 |
| **3** | 13 | 4 | 3 | 9 | 7 | 10.00 | 7.62 |
| **4** | 13 | 3 | 2 | 9 | 10 | 7.14 | 9.05 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.93** | **6.67** |

Table F.16: Normalized data (3D) for $k_n$-Nearest-Neighbor estimation.

### 4D data (Area, Mean, Std, NoP)

| **i** | **$k_n$** | **NOWC in D1(i)** | **NOWC in D2(i)** | **NOWC in TS1(i)** | **NOWC in TS2(i)** | **ER for test sets** | **ER for tr. sets** |
|---|---|---|---|---|---|---|---|
| **1** | 1 | 5 | 5 | 0 | 0 | 14.29 | 0.00 |
| **2** | 1 | 7 | 3 | 0 | 0 | 14.29 | 0.00 |
| **3** | 1 | 7 | 7 | 0 | 0 | 20.00 | 0.00 |
| **4** | 1 | 8 | 6 | 0 | 0 | 20.00 | 0.00 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **17.14** | **0.00** |
| **1** | 2 | 3 | 9 | 0 | 21 | 17.14 | 10.00 |
| **2** | 2 | 6 | 5 | 0 | 17 | 15.71 | 8.10 |
| **3** | 2 | 2 | 11 | 0 | 18 | 18.57 | 8.57 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **4** | 2 | 3 | 12 | 0 | 19 | 21.43 | 9.05 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **18.21** | **8.93** |
| **1** | 4 | 3 | 5 | 7 | 14 | 11.43 | 10.00 |
| **2** | 4 | 4 | 5 | 7 | 13 | 12.86 | 9.52 |
| **3** | 4 | 2 | 10 | 6 | 14 | 17.14 | 9.52 |
| **4** | 4 | 3 | 5 | 7 | 14 | 11.43 | 10.00 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **13.21** | **9.76** |
| **1** | 8 | 5 | 5 | 11 | 13 | 14.29 | 11.43 |
| **2** | 8 | 8 | 2 | 14 | 13 | 14.29 | 12.86 |
| **3** | 8 | 3 | 7 | 11 | 12 | 14.29 | 10.95 |
| **4** | 8 | 4 | 4 | 10 | 15 | 11.43 | 11.90 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **13.57** | **11.79** |
| **1** | 12 | 3 | 5 | 11 | 14 | 11.43 | 11.90 |
| **2** | 12 | 7 | 3 | 14 | 13 | 14.29 | 12.86 |
| **3** | 12 | 3 | 7 | 11 | 12 | 14.29 | 10.95 |
| **4** | 12 | 4 | 4 | 10 | 15 | 11.43 | 11.90 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **12.86** | **11.90** |
| **1** | 16 | 3 | 5 | 11 | 14 | 11.43 | 11.90 |
| **2** | 16 | 4 | 3 | 11 | 14 | 10.00 | 11.90 |
| **3** | 16 | 3 | 7 | 11 | 12 | 14.29 | 10.95 |
| **4** | 16 | 3 | 5 | 7 | 17 | 11.43 | 11.43 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **11.79** | **11.55** |
| **1** | 20 | 3 | 5 | 10 | 15 | 11.43 | 11.90 |
| **2** | 20 | 6 | 3 | 13 | 14 | 12.86 | 12.86 |
| **3** | 20 | 2 | 8 | 8 | 14 | 14.29 | 10.48 |
| **4** | 20 | 4 | 5 | 9 | 17 | 12.86 | 12.38 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **12.86** | **11.90** |
| **1** | 24 | 3 | 5 | 11 | 15 | 11.43 | 12.38 |
| **2** | 24 | 4 | 3 | 12 | 16 | 10.00 | 13.33 |
| **3** | 24 | 3 | 7 | 11 | 12 | 14.29 | 10.95 |
| **4** | 24 | 4 | 4 | 10 | 16 | 11.43 | 12.38 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **11.79** | **12.26** |

Table F.17: Unnormalized data (4D) for $k_n$-Nearest-Neighbor estimation.

## Normalized 4D data (Area, Mean, Std, NoP)

| **i** | **$k_n$** | **NOWC in D1(i)** | **NOWC in D2(i)** | **NOWC in TS1(i)** | **NOWC in TS2(i)** | **ER for test sets** | **ER for tr. sets** |
|---|---|---|---|---|---|---|---|
| **1** | 1 | 2 | 2 | 0 | 0 | 5.71 | 0.00 |
| **2** | 1 | 4 | 4 | 0 | 0 | 11.43 | 0.00 |
| **3** | 1 | 2 | 3 | 0 | 0 | 7.14 | 0.00 |
| **4** | 1 | 4 | 1 | 0 | 0 | 7.14 | 0.00 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **7.86** | **0.00** |
| **1** | 2 | 2 | 4 | 0 | 11 | 8.57 | 5.24 |
| **2** | 2 | 1 | 4 | 0 | 9 | 7.14 | 4.29 |
| **3** | 2 | 0 | 4 | 0 | 5 | 5.71 | 2.38 |
| **4** | 2 | 4 | 2 | 0 | 11 | 8.57 | 5.24 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **7.50** | **4.29** |
| **1** | 3 | 2 | 2 | 7 | 6 | 5.71 | 6.19 |
| **2** | 3 | 2 | 3 | 8 | 4 | 7.14 | 5.71 |
| **3** | 3 | 2 | 2 | 2 | 4 | 5.71 | 2.86 |
| **4** | 3 | 5 | 0 | 8 | 7 | 7.14 | 7.14 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **6.43** | **5.48** |
| **1** | 4 | 2 | 3 | 4 | 9 | 7.14 | 6.19 |
| **2** | 4 | 2 | 4 | 4 | 7 | 8.57 | 5.24 |
| **3** | 4 | 2 | 4 | 1 | 8 | 8.57 | 4.29 |
| **4** | 4 | 5 | 1 | 4 | 8 | 8.57 | 5.71 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.21** | **5.36** |
| **1** | 5 | 2 | 2 | 8 | 8 | 5.71 | 7.62 |
| **2** | 5 | 2 | 3 | 9 | 5 | 7.14 | 6.67 |
| **3** | 5 | 2 | 4 | 1 | 7 | 8.57 | 3.81 |
| **4** | 5 | 7 | 1 | 7 | 8 | 11.43 | 7.14 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.21** | **6.31** |
| **1** | 6 | 2 | 2 | 4 | 11 | 5.71 | 7.14 |
| **2** | 6 | 2 | 5 | 6 | 7 | 10.00 | 6.19 |
| **3** | 6 | 1 | 5 | 1 | 9 | 8.57 | 4.76 |
| **4** | 6 | 6 | 1 | 5 | 10 | 10.00 | 7.14 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.57** | **6.31** |
| **1** | 7 | 2 | 1 | 9 | 8 | 4.29 | 8.10 |
| **2** | 7 | 2 | 5 | 11 | 7 | 10.00 | 8.57 |
| **3** | 7 | 1 | 3 | 1 | 8 | 5.71 | 4.29 |
| **4** | 7 | 8 | 1 | 9 | 9 | 12.86 | 8.57 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.21** | **7.38** |
| **1** | 8 | 2 | 1 | 8 | 10 | 4.29 | 8.57 |
| **2** | 8 | 2 | 5 | 11 | 10 | 10.00 | 10.00 |
| **3** | 8 | 1 | 3 | 1 | 9 | 5.71 | 4.76 |
| **4** | 8 | 6 | 1 | 7 | 9 | 10.00 | 7.62 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **7.50** | **7.74** |
| **1** | 9 | 3 | 1 | 10 | 9 | 5.71 | 9.05 |
| **2** | 9 | 2 | 4 | 11 | 8 | 8.57 | 9.05 |
| **3** | 9 | 1 | 3 | 4 | 6 | 5.71 | 4.76 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **4** | 9 | 8 | 1 | 9 | 8 | 12.86 | 8.10 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **8.21** | **7.74** |
| **1** | 10 | 3 | 2 | 9 | 12 | 7.14 | 10.00 |
| **2** | 10 | 2 | 4 | 11 | 9 | 8.57 | 9.52 |
| **3** | 10 | 1 | 3 | 2 | 9 | 5.71 | 5.24 |
| **4** | 10 | 5 | 1 | 8 | 10 | 8.57 | 8.57 |
| **Mean Error Rate (test sets/training sets):** | | | | | | **7.50** | **8.33** |

Table F.18: Normalized data (4D) for $k_n$-Nearest-Neighbor estimation.