Universitetet i Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study program/specialization: Automation and signalprocessing	Spring semester, 2016 Open / ~~Confidential~~
Author: Gieril Ánde E. Lindi	Gieril Á. Lindi ..... (signature author)
Instructor: Trygve Christian Eftestøhl Supervisor(s): Trygve Christian Eftestøhl	
Title of Master's Thesis: Development of face recognition system for use on the NAO robot Norwegian title: Utvikling av ansiktsgjenkjenningssystem for bruk på NAO robot	
ECTS: 30	
Subject headings: NAO, LBP, K-NN, GUI, face recognition, online learning	Pages: 56 + attachments/other: 15 Stavanger, 15.06/2016 Date/year

Abstract

The main objective of this thesis was to implement a demonstration behaviour for the NAO robot, with focus on face recognition. To achieve this, a complete framework for face recognition that is capable of real-time processing and learning had to be implemented. A pre-trained database is not needed, as the framework learns new faces on-the-fly.

For real time processing and recognition the computation lightness is important, as well as the precision. Therefore the LBP descriptor was chosen to be the main descriptor in the mentioned framework. The K- Nearest Neighbour classifier is used for matching, where the distance metric between the face representations is calculated using the χ^2 distance score.

To be able to classify an unknown face, a threshold is used when predicting. If the χ^2 distance score returned is above a set threshold the learning module is initialized, where only key frames are extracted from the face and stored in the database. These key frames represent the face in different poses and expressions, thus assuring robustness for the real-time face recognition system.

The NAO robot acts upon various "events" based on the classifications done by the system.

The performance of the system is evaluated by using available pre-existing face databases consisting of faces under varying conditions regarding illumination, facial expressions and pose. These tests were done by performing a K-fold cross validations. The validation results show high performance for both precision and speed. The face recognition system achieves 91.7% precision when evaluated on the yale face database A, and 99.8% precision for the AT&T database.

Contents

Contents **iii**

1 Introduction **1**
1.1 Motivation & background 2
1.2 Thesis outline 2

2 Theory **5**
2.1 Image acquisition 6
2.2 Image Preprocessing 6
2.2.1 Photometric normalisation 6
2.2.2 De-noising / Smoothing 7
2.2.3 Face detection 8
2.2.4 Resizing 9
2.3 Face description 10
2.3.1 Local Binary Patterns 10
2.3.2 Eigenfaces 13
2.3.3 Fisherfaces 13
2.4 Classification 14
2.4.1 K-nearest neighbour 14
2.4.2 Support Vector Machine 14
2.5 Learning 15
2.5.1 Off-line learning 15
2.5.2 On-line learning 16

3 System overview **17**

3.1 NAO - the humanoid robot . . . 17
3.2 Development tools . . . 24
3.2.1 Python . . . 24
3.2.2 Integrated Development Environment . . . 24

4 Implementation **27**
4.1 Choice of prediction model . . . 29
4.2 Graphical user interface . . . 30
4.2.1 Saving and loading a pre-trained model . . . 32
4.2.2 Connecting to the robot . . . 32
4.3 Image acquisition . . . 33
4.4 Face detection and extraction . . . 34
4.5 Image preprocessing . . . 36
4.6 Feature extraction and classifying . . . 37
4.7 Prediction and learning module . . . 37

5 Experiments & results **41**
5.1 Datasets . . . 42
5.1.1 AT&T face database . . . 42
5.1.2 Yale face database A . . . 42
5.2 Choice of K for K-NN classifier . . . 43
5.3 Experiment 1 - varying block size . . . 44
5.4 Experiment 2 - Distance metric . . . 45
5.5 Experiment 3 - Applying image processing . . . 46
5.6 Threshold value for unknown faces . . . 48

6 Conclusion & Future work **51**
6.1 Future work . . . 52

Bibliography **53**

A Implementation of software **57**
A.1 Installing the Python distribution . . . 57
A.2 Adding the NAO - Python SDK . . . 58
A.3 Adding the OpenCV library . . . 58
A.4 Installing Qt . . . 58
A.4.1 Minimalist GNU for Windows . . . 58
A.5 Adding the Dlib library . . . 59
A.6 Adding the Pyqtgraph . . . 59
A.7 Using the program . . . 59

B Implementations in Python **61**
B.1 Prediction model 62
B.2 Classifier class 63
B.2.1 K-NN classifier 64
B.3 Feature class 65
B.3.1 Spatially enhanced histogram 66
B.4 Local descriptor class 67
B.4.1 Uniform LBP class 67
B.4.2 Extended(Circular) LBP class 68

List of Tables **71**

Acronyms

AHE Adaptive Histogram Equalization. 6

BF Bilateral Filter. 7

CLAHE Contrast-Limited Adaptive Histogram Equalization. 7

GUI Graphical User Interface. 26

HE Histogram Equalization. 6

HRI Human-Robot Interaction. 10

KNN K-Nearest Neighbour. 14

LBP Local Binary Pattern. 10

PCA Principal Component Analysis. 13

SDK Software Development Kit. 24

SVM Support Vector Machines. 14

CHAPTER 1

Introduction

This thesis provides a study on face recognition for the humanoid robot NAO. The background of this thesis was the idea to eventually use the developed demonstration behaviour at stands, for example "open day" at the university, when students and employees shall inform about the available programs at the institute. The ambition with this is to promote the institute and hopefully increase the interest among future students at the University of Stavanger.

The tasks concerning both the face recognition and NAO robot implementations were heavily based on subjects educated at the institute for Information Technology - automation and signal processing.

The solution was to develop a complete framework capable of detecting faces in the retrieved frames and recognise these, in addition to be able to learn new faces real-time. Everything is programmed purely in Python, this includes the framework and the management of the NAO.

The robot reacts on various "events" based on the predictions done, for instance if a face is not recognised the robot will ask the user for his or her name, in which the learning module will start using the name provided as input.

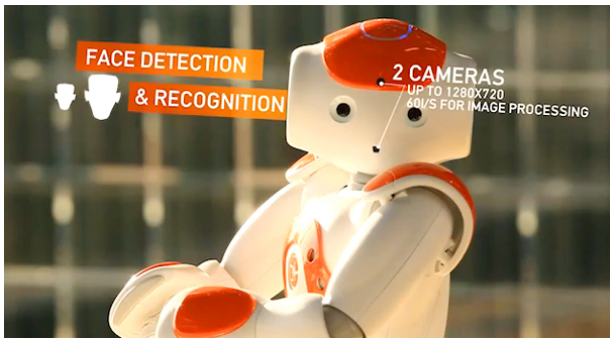


Figure 1.1: NAO robot - ©2012 Aldebaran Robotics. All rights reserved

1.1 Motivation & background

The main objective of this thesis was to develop a face recognition system for use on the NAO robot. This system can then be used and developed further for a complete demonstration behaviour on the NAO, where the robot acts upon visual recognition of faces. The key objectives of this thesis have been:

1. Develop a framework for face recognition, which can be expanded.
2. Make a simple GUI to make the program easy to use, and simplifies the steps needed to connect to the robot.
3. Develop a complete face recognition system capable of learning in real-time, without the need of a pre-trained database.

1.2 Thesis outline

This sections presents the overview of the thesis. The thesis is organized as listed below and gives a short summary of each chapter.

Introduction

The main objective of the thesis is introduced here, along with a brief introduction to face recognition technology and it's uses.

Theory

This chapter gives a brief introduction to various face recognition theories needed in order to grasp the key principles of face recognition.

System overview

Provides a system overview of the humanoid robot NAO and how it works, in addition to the various libraries used.

Implementation

This chapter presents the implementation of the system.

Experiments & results

This chapter presents the results achieved. The main focus was the performance of the face recognition system in terms of precision and processing speed.

Limitations, conclusion & Future work

Discussions regarding future work for the developed face recognition system and conclusion.

Appendix A - CD contents

This appendix includes the contents provided on the CD.

Appendix B - Software implementation requirements

The software and libraries needed to run the implementation is included in this appendix. A step-by-step guide regarding installation is also provided.

Appendix C - Overview of Python code

The overview of some the functions implemented in Python is found here.

CHAPTER 2

Theory

At it's core face recognition is a visual patter recognition problem, where a 3D object, subject to changes in Expressions, pose, illumination etc., is to be classified based on its two-dimensional image.

A face recognition system usually consists of four modules, shown in figure 2.1. Initial detection of face in image, alignment or image preprocessing, feature extraction and finally classifying the face.

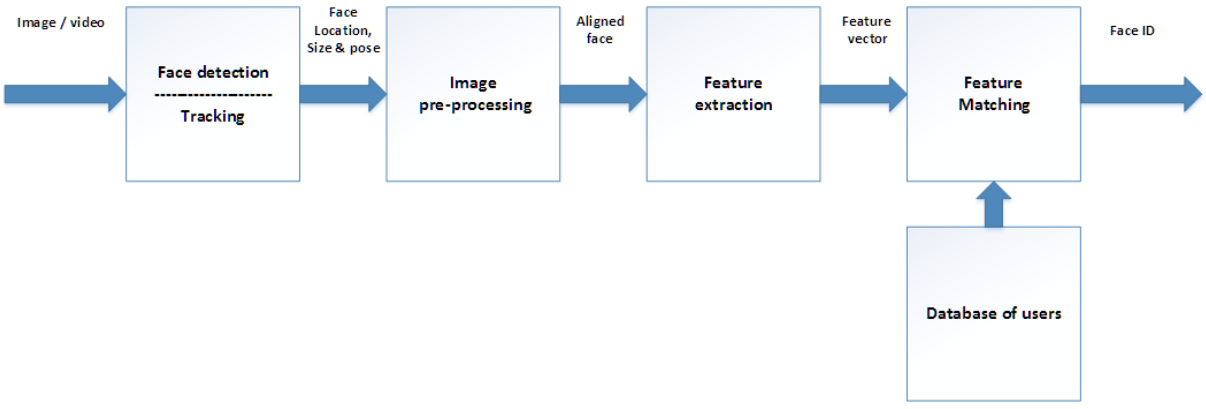


Figure 2.1: Face recognition process flow. Source: [14]

The theory behind the work done in this thesis will be presented in this chapter. First section briefly introduces the first steps needed for face recognition, image acquisition and

face detection followed up with pre-processing steps done on the detected face. Then the face recognition technology is presented, involving face descriptors and classifiers. In the end learning methods are discussed.

2.1 Image acquisition

Image acquisition is done by retrieving frames from the top camera on the robot by using the module **ALVideoDevice**. This module provides images from the robots camera with a resolution up to 1280x960 at 30 frames per seconds with native colorspace YUV422 [24], see section 3.1 in chapter 3 for details regarding the robots camera.

It is desired to have the best possible performance for the live image acquisition, therefore the FPS parameter in table 3.2 is set to it's maximum value (30). However, this does not mean that the video source is able to run at the set frame rate [21]. Furthermore, to be able to run this program on all types of networks and based on table 3.6, the camera resolution was set to *320x240*. This assures at least 11 FPS is possible via WiFi and does not compromise image quality considerably.

2.2 Image Preprocessing

For face recognition it is important that the captured face frames describing the face are consistent. Properties like illumination, pose, facial expressions and scale of the face play a vital role in the performance of a face recognition system. Hence image enhancement is applied to improve the performance. At first a few image enhancement methods will be described, followed up by the method used for face detection.

2.2.1 Photometric normalisation

Photometric normalisation is an important step for face recognition, as illumination variations is one of the most significant problems. [7]. For instance a directed light-source, like the sun, may partially over-saturate one half of the face, while casting a shadow on the other half, making it almost invisible. Photometric normalisation is a possible approach for this problem, hence it is important for face recognition.

A computer image processing technique called Adaptive Histogram Equalization (AHE), can be used to improve contrast in images. The difference from normal Histogram Equalization (HE) is that the image is divided into small regions called tiles, wherein a histogram

is computed locally. The histograms are then used to redistribute the intensity values of the image, which then improve the local contrast and enhances the edge definitions in each region.

However, if there's noise in the image, it will be amplified. Therefore an advanced local histogram equalisation, named Contrast-Limited Adaptive Histogram Equalization (CLAHE), attempts to prevent this by limiting the contrast. Meaning if a histogram bin is above a certain threshold the pixels are clipped and distributed uniformly to other bins before applying HE locally in the tiles. To avoid artefacts around the tile borders, bilinear interpolation is applied in the end. [17]

2.2.2 De-noising / Smoothing

Captured frames might be subjective to some sort of noise, due to errors and electronic noise in the capturing process. Images can be enhanced by applying de-noising or smoothing, for instance an Gaussian filter, amongst many other. However smoothing an image can have a negative effect on the face recognition performance as important facial features might be lost, like edges. This can be clearly seen in figure 2.2.

Figure 2.2: Effects of Gaussian blur. Original image to the left, right image shows the effects of too much blurring; important facial information is lost. Source: [1].

Bilateral Filter (BF) is an extended version of a Gaussian filter, where the variations of intensities to preserve edges is also considered. The key idea of BF is that two pixels are close to each other in both spatial location and photometric range similarity. The BF is defined as such:

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(I_p - I_q) I_q \tag{2.1}$$

where W_p is a normalization factor:

$$W_p = \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(I_p - I_q) \tag{2.2}$$

The amount of filtering done on image I is measured with the parameters σ_s and σ_r, Equation (2.1) is a normalized weighted average where:

1. G_{σ_s} - Spatial Gaussian that decreases the influence of distant pixels
2. G_{σ_r} - Range Gaussian that decreases the influence of pixels q with an intensity value different from I_p

The term *range* refers to the pixel values themselves, while *space* refers to the pixel location. [18] Figure 2.3 shows an example where BF is applied.

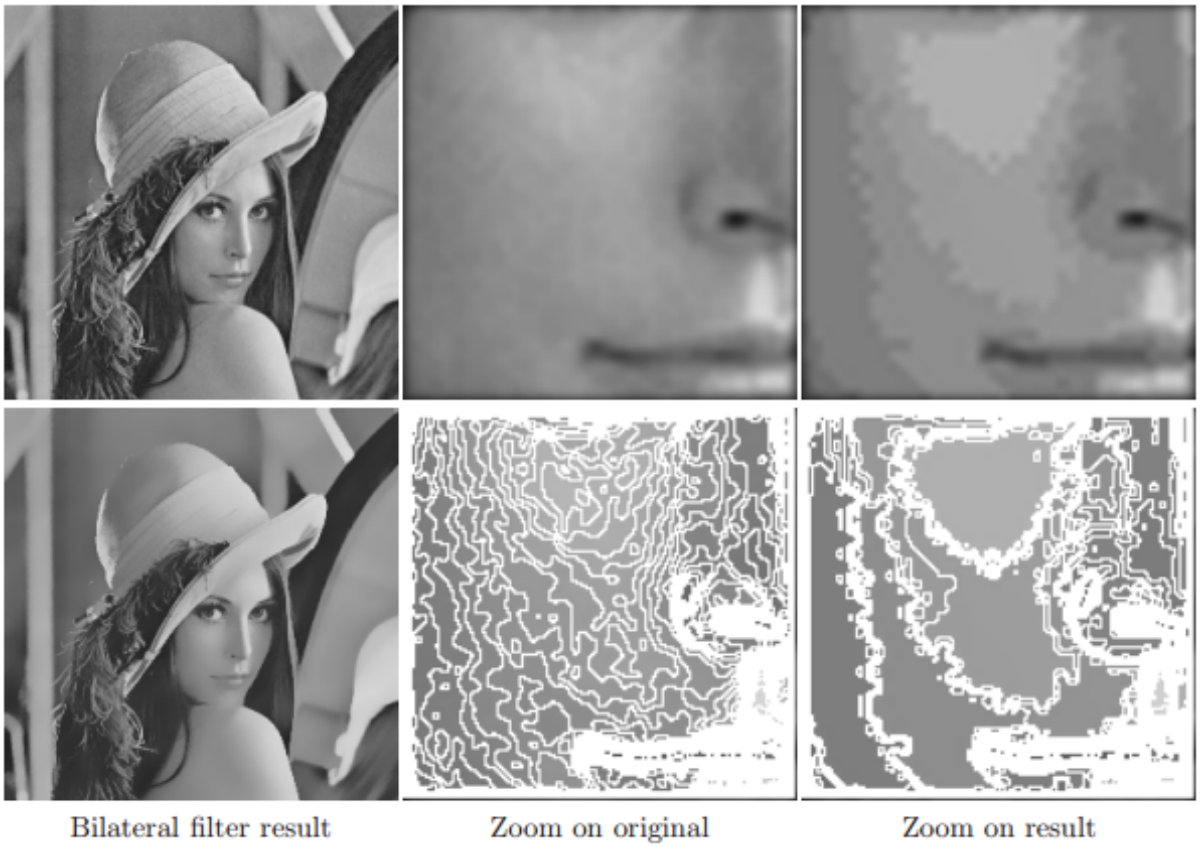


Figure 2.3: Example of results obtained with the bilateral filter. Source: [18]

2.2.3 Face detection

Before a face can be recognised it has to be detected in each frame. For this a generic framework for object detection is used, introduced by P. Viola and M. Jones in 2001.

The framework can be trained to detect a range of objects, but its main focus was face detection. [32]

The framework is a machine learning approach capable of real-time processing while still being robust with a very high detection rate (true-positives). It applies the Adaptive Boosting method, meaning a strong classifier is created by combining numerous weak classifiers for features found in specific positions in the face; e.g eyes, nose, mouth, see figure 2.4

The framework is also scale invariant, meaning it can detect both small and large faces in an image, depending on the distance from the camera. This is done by building an image-pyramid for the detector where the image retrieved is down-sampled based on the scaling factor provided.

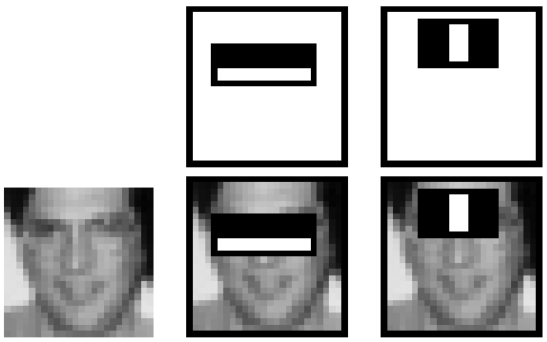


Figure 2.4: Example of some features found by matching local contrast differences. The eye regions are usually slightly darker than the cheek regions, as shown in the centre. The image pair to the right shows the intensity difference between eye regions and the nose bridge. Source: [32]

2.2.4 Resizing

With the chosen resolution for image acquisition discussed briefly in section 2.1, it was found that the extracted faces had resolutions varying from roughly 30x40 pixels to 90x100 pixels[1], depending on the distance from the user to the robot. Thus, a fixed resolution of 70x80[2] was chosen, on the assumption that users will be around 1 meter away from the robot. In addition the processing speed will be slightly better with smaller images that need to be processed..

[1]The distance varied from around 0.5 meters to 1.5 meters
[2](WxH)

2.3 Face description

Face recognition and it's applications has received a significant rise of attention the recent years, and is a very active topic in computer vision research. [33] The fundamental issue in face recognition is finding an efficient facial descriptor.

Most of the latest face recognition methods are based on deep learning and focus heavily on specific subproblems regarding recognition, thus as a result are often computationally intensive [30], [27], [28]. It means these are not easy to implement in a system where prediction and learning has to be done in real-time, and are likely not suitable for Human-Robot Interaction (HRI) applications where preferred interaction time is short.

2.3.1 Local Binary Patterns

The Local Binary Pattern (LBP) operator describes features based on local properties of the object rather than globally. The operator was originally designed for texture description [16], but it can be applied for face recognition problems as well.

Firstly, the image is converted to grey-scale, then the operator assigns a label to every pixel of an image by thresholding the 3-by-3 neighbourhood of each pixel with the centre pixel value. The result is then considered as a binary number, which is defined in equation (2.3).

$$LBP(x_c, y_c) = \sum_{n=0}^{7} s(I_n - I_c) \cdot 2^p \tag{2.3}$$

Where I_c and I_n correspond to the intensity values of the centre pixel and the surrounding 8 pixels respectively. The function $s(k)$ is defined as:

$$s(k) = \begin{cases} 1 & \text{if } k \geq 0 \\ 0 & \text{if } k < 0 \end{cases} \tag{2.4}$$

As an example for the equation above:

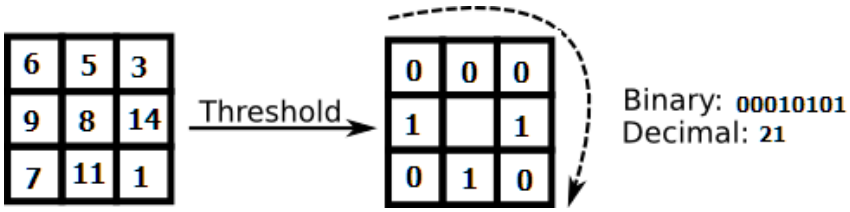


Figure 2.5: LBP thresholding

Circular Local Binary Patterns

The drawback of the original descriptor with a fixed neighbourhood size is that it can't capture details at varying scales. Thus an extension to LBP was made, called Circular Local Binary Patterns, allowing the use of variable neighbourhood sizes (P, R). Defined as P sampling points on a circle of radius R.

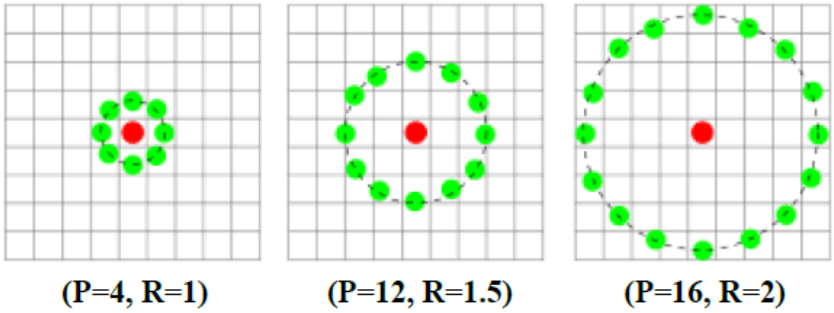


Figure 2.6: Varying Radius and Points for LBP descriptor. Source: [10]

Uniform patterns

It is also possible to use a subset of the 2^P LBPs to describe an image. These patterns are called uniform patterns and a LBP is considered to be uniform if it has at most two 0-1 or 1-0 transitions. For example, the LBP in figure 2.5 is **not** uniform, but 00111100 is.

Each unique pattern has its own bin in the LBP histogram, while the patterns that are not uniform are stored in a single "miscellaneous' bin. The number of uniform patterns depend on the sample points P, meaning a larger P results in a higher dimensionality of the histograms. If $P = 8$ there are 59 bins in a histogram for the uniform patterns, including the miscellaneous bin $(58 + 1)$[1].

Compared to a normal histogram for grey-scale images this is a 77% reduction in feature vectors. This is possible as the uniform patterns are enough to describe textures in an image. [16] The following notation is used for the uniform LBP descriptor: $LBP^{u2}_{P,R}$, where $u2$ stands for the use of uniform patterns.

The histogram of the LBP image is defined as:

$$H_i = \sum_{x,y} I\{f_l(x, y) = i\}, i = 0..., n - 1 \tag{2.5}$$

[1]For the normal LBP descriptor with $P = 8$, the bin size is: $2^P = 2^8 = 256$

where n is the different labels produced by the LBP operator and:

$$I\{A\} = \begin{cases} 1, & \text{A is true} \\ 0, & \text{A is false} \end{cases} \tag{2.6}$$

Face recognition based on LBP

The explained approach works well for texture classification, but applying the same approach for face images would result in severe loss of spatial information. Thus, to preserve spatial info, the image is divided into M blocks $R_0, R_1, ..., R_{M-1}$ and the spatially enhanced histogram of the image is defined as:

$$H_{i,j} = \sum_{x,y} I\{f_l(x,y) = i\} I\{(x,y) \in R_j\}, i = 0, ..., n-1, j = 0, ..., m-1 \tag{2.7}$$

See figure 2.7 for overview. In the literature [2], $7x7$ blocks of size $18x21$ pixels is recommended to achieve a good balance between recognition performance and feature vector length. Three following distance metrics are proposed for histogram comparisons, and are used to compute the distance between feature vectors[1] S and M.

1. Chi-Square Distance, recommended in the literature [2]:

$$\chi^2(S, M) = \sum_{i=1} \frac{(S_i - M_i)^2}{Si + M_i} \tag{2.8}$$

2. Euclidean distance:

$$D(S, M) = \sqrt{\sum_{i=1} (S_i - M_i)^2} \tag{2.9}$$

3. Dimensionality Invariant Similarity Measure presented newly by A. Hassanat. This metric is invariant to data scale, noise and outliers, and is referred to as the Hassan distance. [3]:

$$D(S, M) = \sum_{i=1} (1 - \frac{1 + min(S_i, M_i)}{1 + max(S_i, M_i)}) \tag{2.10}$$

[1]The spatially enhanced histograms of each image are the feature vectors.

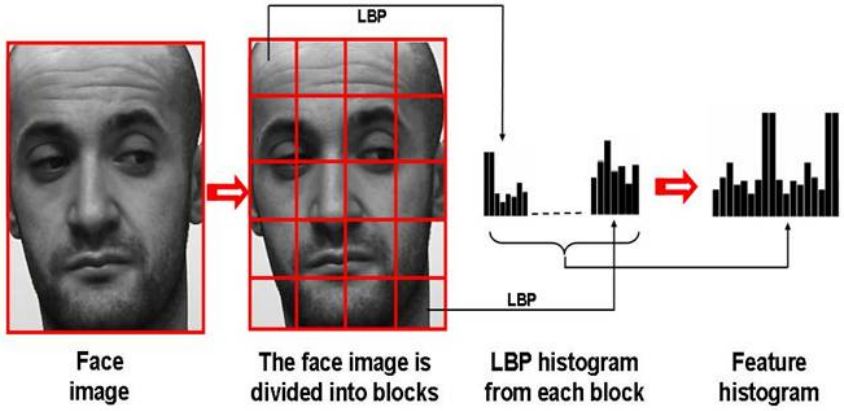


Figure 2.7: Face representation with Local Binary Patterns. Image source: [19]

2.3.2 Eigenfaces

One of the most thoroughly investigated approaches to face recognition is Eigenfaces [5]. The Eigenfaces approach is based on using Principal Component Analysis (PCA), and is named after the eigenvectors used to describe the faces. To find the Eigenspace of the training samples, which describes the difference between each one of these, PCA is applied on a single vector containing the image data of a face. This method achieves close to real-time performance as described in [31]. Regrettably this method requires a lot of training samples for each class and preferably in all kinds of various conditions regarding: illumination, pose and facial expression, due to being very weak against variations in pose and illumination.

2.3.3 Fisherfaces

For the Eigenfaces approach the difference between the training samples is maximized, but with multiple training samples for a single class the differences within a class are also increased.

To correct this a new method, called FisherFaces, is introduced [6]. Here the inter- and intra- class scatter matrices, S_b and S_w, are defined:

$$S_b = \sum_{j=1}^{c} (\mu_j - \mu)(\mu_j - \mu)^T \tag{2.11}$$

$$S_b = \sum_{j=1}^{c} \sum_{i=1}^{N_j} (x_i^j - \mu_j)(x_i^j - \mu_j)^T \tag{2.12}$$

where:

1. μ is the average of all training samples
2. μ_j is the average of all training samples of a class
3. c is the number of classes
4. N_j is the number of training samples in class j

To minimize the intra-class scatter, the method tried to find a linear projection space that maximises:

$$\frac{\det S_b}{\det S_w} \tag{2.13}$$

For each class from all training samples of a class FisherFaces creates a single dimension in the description space.

2.4 Classification

Previous section explained how to detect and describe a face in an image. The descriptors are then used as input for the classifier to determine whom the face belongs to. In this section two different classification methods are introduced, which are commonly used for object recognition.

2.4.1 K-nearest neighbour

K-Nearest Neighbour (KNN) is one of the simplest available classifiers, and is considered to be a lazy algorithm. This means that all of the training data is kept, and the decisions done by this classifier is based on the entire training data. The K-Nearest Neighbour classifier performs well on multi-class problems and is very fast, although as the description dimensionality and sample size increases the classifier will be slower. If the classifier is to be used with very large datasets consisting of a huge number of classes, then PCA can be used to reduce the dimensions size. [**?**]

2.4.2 Support Vector Machine

Support Vector Machines (SVM) is a classification method defined by a separating hyperplane. When the classifier is given labelled training data, the output from the SVM

is an optimal hyperplane that categorizes new examples. Consider the example given in figure 2.8, here the SVM algorithm has found an optimal hyperplane that returns the largest minimum distance to the training points, on both sides. This distance is called the maximum margin, which is what an SVM at its simplest tries to find.

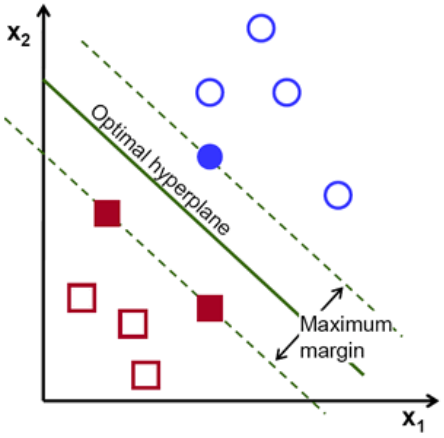


Figure 2.8: Two class problem.

SVM is a binary-class classifier, often used in object matching cause of its precision and speed, even with large sets of training samples. But SVMs can be extended to multi-class problems by combining numerous binary-class classifiers.

2.5 Learning

before a classifier can predict anything, it has to be trained using data that represents the faces correctly. This can be done by for example feeding the classifier with a set of training images with their respective class labels before starting the prediction. Here two types of learning will be described.

2.5.1 Off-line learning

Off-line learning is the most common way of learning a classifier and must be done before the classifier is set to do its task. This means that for example the robot is able to classify and recognise faces immediately when the application is started, and if the learned database consists of all possibilities for the specific recognition task then no new learning is needed.

Off-line learning is usually used in papers to test and validate various classification and descriptor methods by using a pre-existing set of images containing the object intended

to be classified. This is done by gathering a huge set of images of the object and then splitting the set into a test and training subset. The split-ratio is usually 20-80, where 20% of the total set is used for training and the rest for validation / testing.

The obvious advantage for this kind of learning is that there is no time limit for the learning process, and in addition testing can be done on an existing database, thus validating and guaranteeing some degree of performance for the classifier before it's set out to do it's intended task.

However, off-line learning has its disadvantage as well, in regards that it is not possible to learn an unknown object, if encountered, resulting in an robot unable to adapt itself to an unknown situation.

2.5.2 On-line learning

On-line learning is learning of objects on-the-fly, meaning while the robot is active and in use. If the robot encounters an unknown object with this type of learning, it is able to adapt and learn the unknown object in a few seconds at most, making the robot highly adaptable to its surroundings. In addition, relearning of objects is also possible.

CHAPTER 3

System overview

3.1 NAO - the humanoid robot

The physical system consists purely of the humanoid robot. The different modules regarding the robot will be presented here. The department is in possession of four NAO humanoid robots, which are developed by a French company, Aldebaran Robotics. Two of these robots are of a newer model, V5.

Hardware

One of the mentioned robots is called Randi. She is 57.4 cm tall and is equipped with a myriad of features that define her as a humanoid robot. Some of the senses that account for natural interaction with the robot:

- **Moving around:** The body itself has 25 degrees of freedom (DOF), controlled by actuators and motors, thus allowing the robot to do basic human behaviours.
- **Feeling:** Randi has numerous sensors in her head, hands and feet, as well as sonars, enabling her to perceive the environment as well as orientate in it.
- **Communicating:** With her 4 directional microphones and loudspeakers, Randi can interact with humans in a completely natural manner, by listening and speaking.

- **Seeing:** Randi is also equipped with 2 cameras, thus making it possible for Randi to see in high resolution, helping her recognise shapes, faces, objects among other things.
- **Connecting:** NAO is also able to access the internet autonomously by using a range of different connection modes; Wifi, Ethernet.

  Her brain itself is powered by an **ATOM Z530** 1.6GHz CPU, that runs a Linux kernel and communicates with Aldebaran's proprietary software called NAOqi, which is the main software that runs and controls the robot.

  Furthermore the robot is equipped with 48.6 Wh battery providing her 60 to 90 minutes of autonomy, depending on usage level. See table 3.1 for technical overview of the robot.

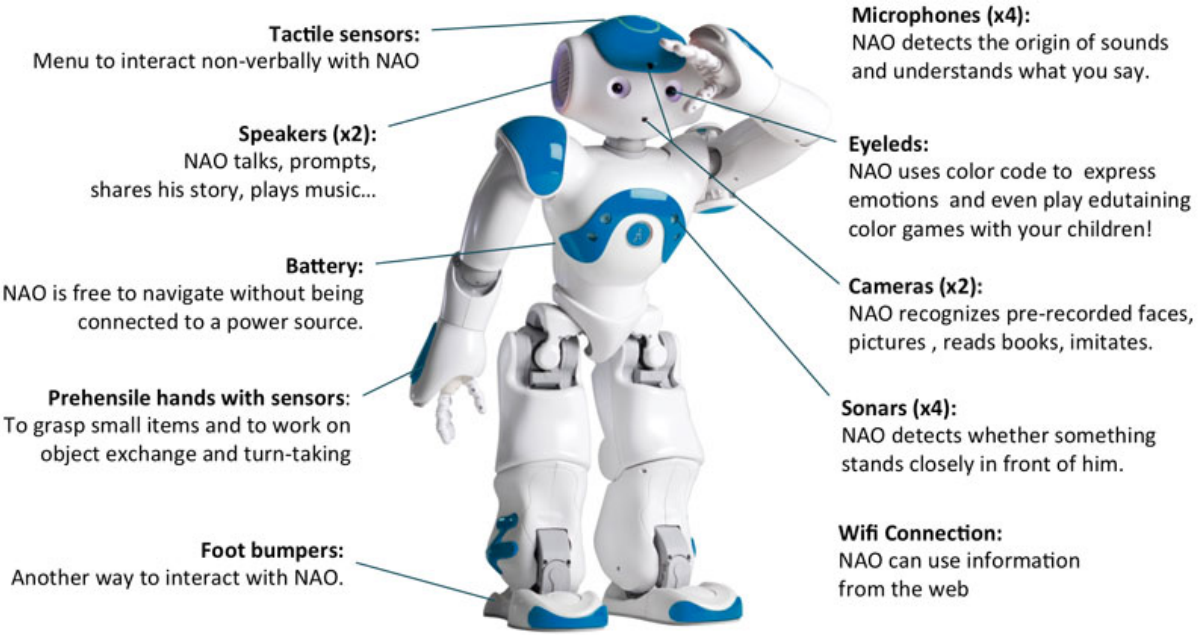


Figure 3.1: NAO features. ©Gigabotics 2016, all rights reserved.

NAO technical overview	
Construction:	
Height	574mm
Depth	311mm
Width	275mm
Weight	5.4kg
Battery:	
Energy	48.6Wh
Autonomy	60-90 min
Motherboard:	
ATOM Z530 CPU	1.6 GHz
Connectivity:	
WiFi	IEEE 802.11 a/b/g/n
Ethernet	RJ45 - 10/100/1000 base T
USB	
Video Camera:	
Two identical HD cameras	1280x960 resolution @ up to 30 FPS
Programming languages:	
C++, Python, Java, MATLAB, Urbi, C, Net	

Table 3.1: NAO technical overview. Source: [22]

NAO's camera

NAO has two cameras that act as its eyes. The following camera parameters can be modified:

Parameter	Min Value	Max Value	Default Value	Camera ID name
Brightness	0	255	55	kCameraBrightnessID
Contrast	16	64	32	kCameraContrastID
Saturation	0	255	128	kCameraSaturationID
Hue	-180	180	0	kCameraHueID
Gain	32	255	32	kCameraGainID
Horizontal Flip	0	1	0	kCameraHFlipID
Vertical Flip	0	1	0	kCameraVFlipID
Auto Exposition	0	1	1	kCameraAutoExpositionID
Auto White Balance	0	1	1	kCameraAutoWhiteBalanceID
Camera Resolution	kQVGA	k4VGA	kQVGA	kCameraResolutionID
Frames Per Second	1	30	5	kCameraFrameRateID
Exposure (time in ms = value / 10)	1	2500 (250ms)	NA	kCameraExposureID
Camera Select	0	1	0	kCameraSelectID
Reset camera registers	0	1	0	kCameraSetDefaultParamsID
Auto Exposure Algorithm	0	3	1	kCameraExposureAlgorithmID
Sharpness	-1	7	0	kCameraSharpnessID
White Balance (Kelvin)	2700	6500	NA	kCameraWhiteBalanceID
Back light compensation	0	4	1	kCameraBacklightCompensationID

Table 3.2: Supported parameters for NAO camera. [24]

Supported camera resolutions and colourspaces

The camera supports a multitude of resolutions:

Parameter ID Name	ID Value	Description
AL::kQQQQVGA	8	Image of 40*30px
AL::kQQQVGA	7	Image of 80*60px
AL::kQQVGA	0	Image of 160*120px
AL::kQVGA	1	Image of 320*240px
AL::kVGA	2	Image of 640*480px
AL::k4VGA	3	Image of 1280*960px

Table 3.3: Supported resolutions [24]

Additionally the camera has a range of supported colourspaces. The most commonly used are:

Parameter ID Name	ID Value	Number of layers	Number of channels
AL::kYUV422ColorSpace	9	2	2
AL::kYuvColorSpace	0	1	1
AL::kYUVColorSpace	10	3	3
AL::kRGBColorSpace	11	3	3
AL::kBGRColorSpace	13	3	3

Table 3.4: Supported colorspaces [24]

Camera performance and limitations

The supported frame rates at given resolutions are listed below.

Resolution	Supported Framerate
AL::kQQQQVGA	from 1 to 30 fps
AL::kQQQVGA	from 1 to 30 fps
AL::kQQVGA	from 1 to 30 fps
AL::kQVGA	from 1 to 30 fps
AL::kVGA	from 1 to 30 fps
AL::k4VGA	from 1 to 30 fps

Table 3.5: Supported frame rates [24]

The processing times are ranked as follow for the main colourspaces:

$YUV422 < Yuv < YUV < RGB/BGR < HSY.$

Using the native colourspace on NAO v4 table 3.6 shows the observed frame rates achieved with varying resolution and network type.

Resolution	local	Gb Ethernet	100Mb Ethernet	WiFi g
40x30 (QQQQVGA)	30fps	30fps	30fps	30fps
80x60 (QQQVGA)	30fps	30fps	30fps	30fps
160x120 (QQVGA)	30fps	30fps	30fps	30fps
320x240 (QVGA)	30fps	30fps	30fps	11fps
640x480 (VGA)	30fps	30fps	12fps	2.5fps
1280x960 (4VGA)	29fps	10fps	3fps	0.5fps

Table 3.6: Observed frame rates [20]

From the table above it can be seen that the maximum requested frame rate will be achieved locally, but if the robot is connected remotely, the frame rate is entirely dependent on the available bandwidth on the network. [20]

Software

NAOqi Framework The NAO robots main software is named NAOqi. The NAOqi framework is the programming framework used to program NAO. It answers to the robots needs, namely: parallelism, resources, synchronization, events.

The Framework allows homogeneous communication between different modules (motion, audio, video), homogeneous programming and information sharing.

The framework is also **cross language**, with an identical API for both C++ and Python programming. Meaning software can be developed in C++ and Python using the same programming methods [26].

The NAOqi process

The broker which runs on the robot is a NAOqi executable. Once it's started, it loads a preferences file called *autoload.ini* that defines which libraries should be loaded. Each library contains one or more modules that use the broker to advertise their methods [26].

This broker provides lookup services so that any module in the tree or across the network can find any method that has been advertised. see figure 3.2

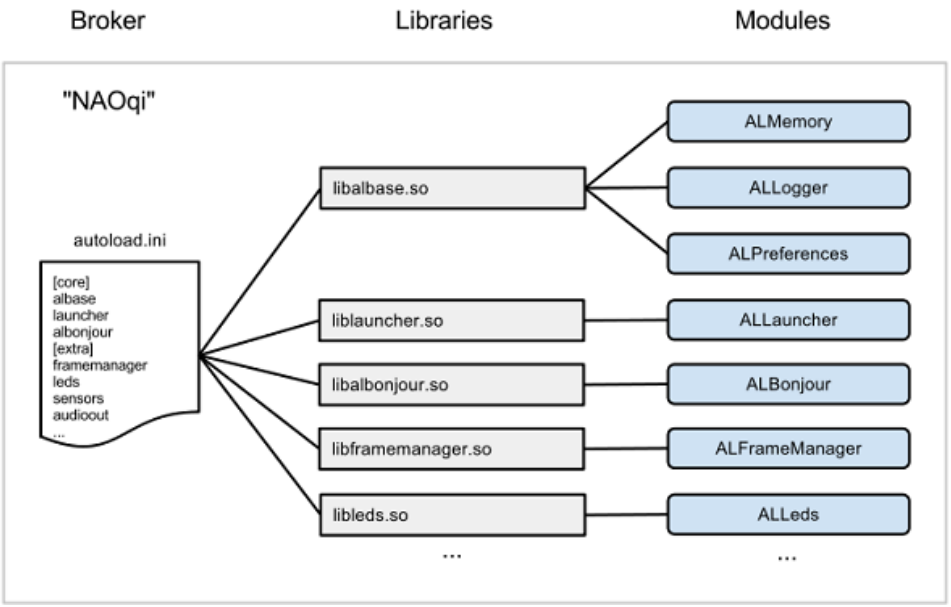


Figure 3.2: The NAOqi broker tree. Broker-Libraries-Modules.Source: [26] ©2016 Aldebaran Robotics. All rights reserved

Furthermore, loading these modules forms a tree of methods attached to the modules and modules attached to a broker. see figure 3.3.

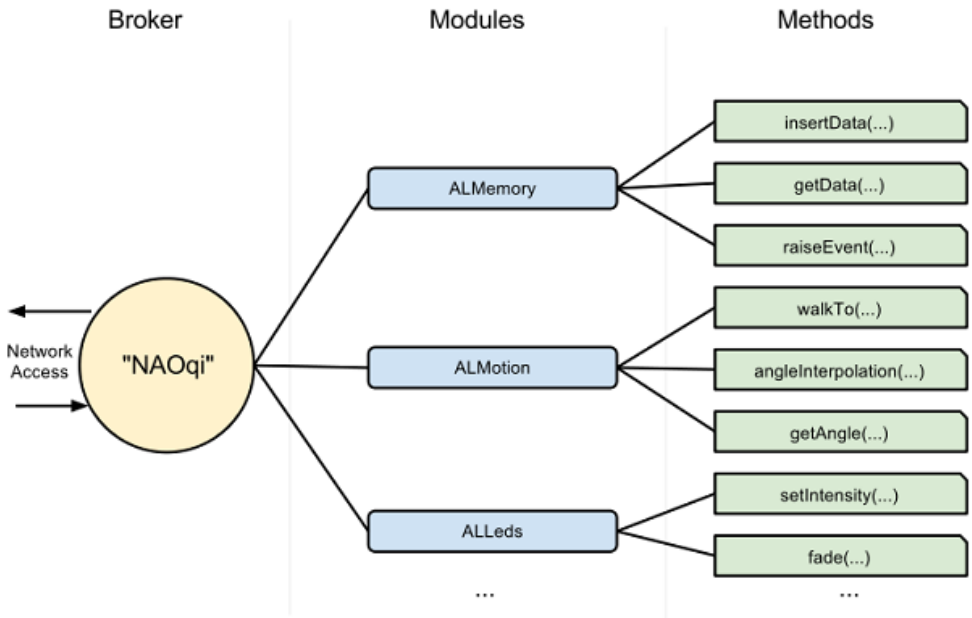


Figure 3.3: The NAOqi broker tree. Broker-Modules-Methods. Source: [26] ©2016 Aldebaran Robotics. All rights reserved

Broker
The broker is an object that allows the user to find modules and methods, and it provides network access; allowing the methods of attached modules to be called from outside the process. Brokers work transparently, allowing the person to write code both for calls to modules in the same process or to modules in another process / machine. [26]

Proxy
A proxy is an object that will behave as the module it represents. For example, if a proxy is created to the *ALTextToSpeech* module, an object containing all the *ALTextToSpeech* methods will be available [26]. See listing 1.

3.2 Development tools

The development tools used in the thesis will be shortly presented here. The main tool used is Python, where most of the programming has been done. For this the Python Software Development Kit (SDK) for NAOqi is required to be able to communicate with the robot.

3.2.1 Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed. [15]

3.2.2 Integrated Development Environment

The Integrated Development Environment (IDE) used for programming in Python in this thesis is called PyCharm. PyCharm features a smart code editor that understands the specifics of Python and provides various productivity boosters; automatic code formatting, code completion, name re-factoring, auto-import, etc.

These features in addition to advanced code analysis routines make PyCharm a useful tool for both advanced Python developers and beginners. [29]

The professional Edition of PyCharm is licensed and available through various subscription options, which feature the same software functionality, but differ in price and terms of use.

The professional edition is free for open source projects and educational uses. [12]

Anaconda

Anaconda is a high performance distribution of Python which includes over 100 of the most popular Python packages for data science. Additionally there's over 720 packages that can be easily installed with conda if necessary. Anaconda includes Conda as a package, dependency and environment manager. Amongst the included packages are *Scikit-learn*, *Scikit-image*, *Matplotlib* to name a few which are used in this thesis. Anaconda is BSD licensed. [4]

NAOqi Python SDK

The NAOqi Python API for the NAO robot allows the user to use all of the C++ API from a remote machine. The API also allows the user to create their own Python scripts, that can run remotely on a computer or on the robot itself. [25]

Using Python is one of the easiest ways to program with NAO.

Aldebaran Robotics has a software documentation web page where everything regarding the robot itself, software and programming is explained through technical details and examples. [23]

The SDK is very easy to use once both Python and the NAOqi SDK are installed, this can be shown with an example using ALProxy, which is an object that gives access to all the methods / modules on the robot that the user wants to connect to. See listing under.

```
from naoqi import ALProxy
tts = ALProxy("ALTextToSpeech", "<IP of your robot>", 9559)
tts.say("Hello, world!")
```

Listing 1: Basic example. [25]

Here the ALProxy object is imported from the NAOqi Python SDK and used to create the module that handles Text-To-Speech in the robot. The last line tells the robot to say "Hello, world!"

OpenCV

OpenCV is an open source computer vision and machine learning software library, released under a BSD license, thus it's free for both academic and commercial use. OpenCV has interfaces to C++, C, Python.

The library has a vast amount of algorithms mainly aimed at real time image processing, these include a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. [11]

Qt

Qt is an application framework. It allows the user to create cross-platform user-interfaces and can be used with several different programming languages, one of these being Python. Qt includes a tool called Qt Designer, allowing users to easily design and build Graphical User Interface (GUI) [8].

Dlib

Dlib is a C++ library containing machine learning algorithms and has an API for Python as well. Dlib is used in this thesis to track and retrieve coordinates for facial landmarks. [13]

CHAPTER 4

Implementation

This chapter describes the implementation for the face recognition system for the NAO robot, with simple interactive behaviours included. The system is able to detect faces and recognise them followed with various robot interactions with the user. If the face is unknown the user will be prompted by the robot to input his or hers name in an input box, after which the learning module will initialize.

The program implementation is purely done in Python, allowing for easy use of the NAOqi Python SDK for further development. The image preprocessing is done with OpenCV, see section 3.2.2. The following flowchart shows a simplified model of how the program runs. The structure of this chapter follows the flow of the program, roughly similar to what is shown in figure 4.1

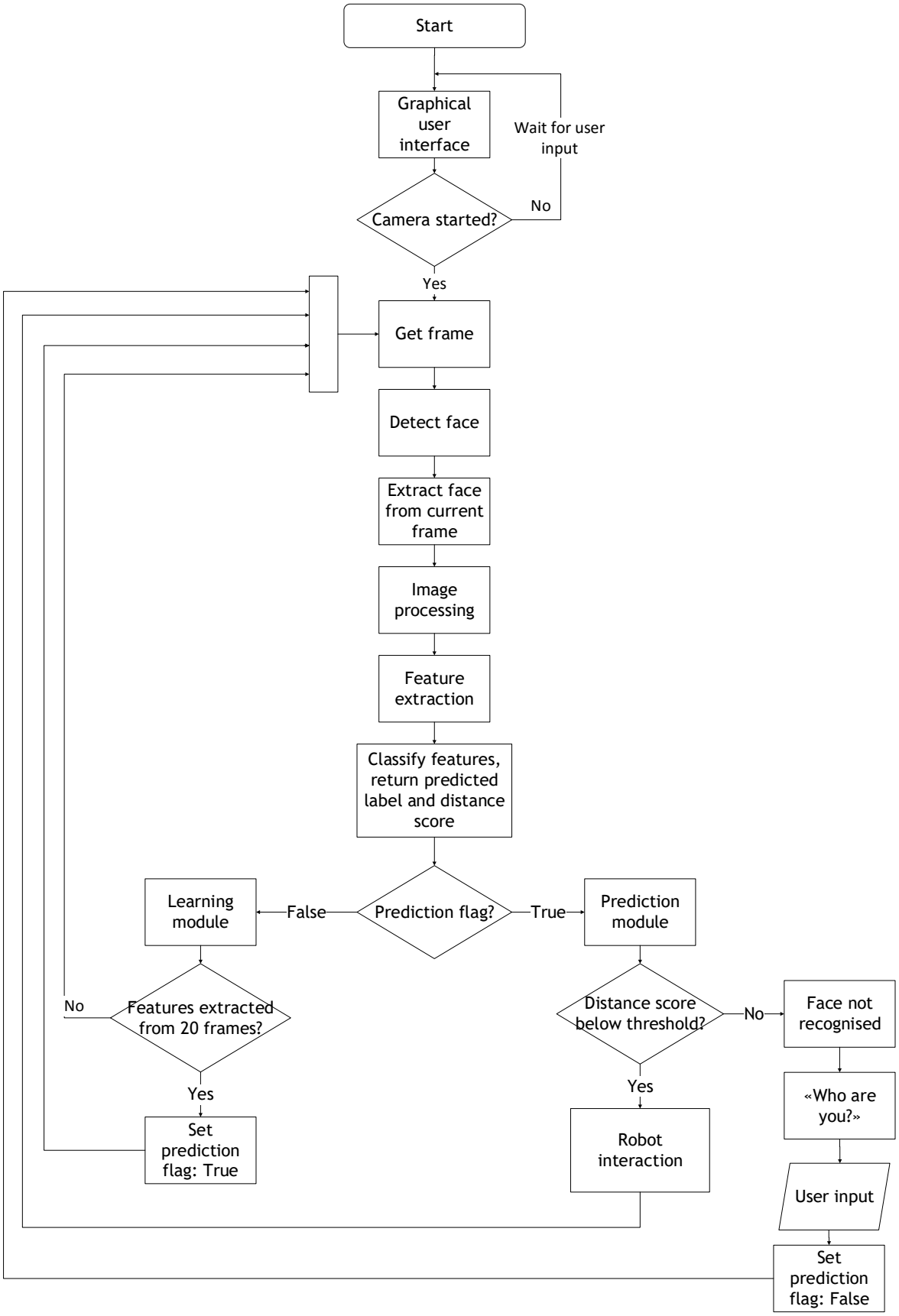


Figure 4.1: Simplified flowchart of the system.

4.1 Choice of prediction model

Before the Graphical User Interface (GUI) is started, which runs the application, the prediction model is loaded. The prediction model is a combination of feature extraction method and a classifier, see sections 2.3 and 2.4.

```
1    model = hent_modell()
```

Listing 2: Python code to load the desired prediction model.

The model is loaded by calling the *hent_modell* function as shown above in listing 2. This function is defined in listing 4. Here The LBP descriptor in combination with the K-Nearest neighbour classifier is chosen, which is also the default combination for this thesis. This is because the mentioned combination is capable of on-line learning (incremental learning), meaning there is no need to pre-train the model on a subset of images before starting up. Hence the *hent_modell* function is called with no additional inputs.

The Eigenfaces and Fisherfaces features are also available along with an SVM classifier, however these do not support on-line learning. If one wishes to try either of these, the model has to be pre-trained on an existing database consisting of images. The *hent_modell* function looks like what is shown in listing 3

```
1    face_resize = (80, 100)
2    [bilder, label, names] = read_images1(dataset_path, sz=face_resize, na=False)
3    model = hent_modell(im_sz=face_resize, person_navn=names)
4    model.kalkuler(bilder, label)
```

Listing 3: Python code to load and pre-train the desired prediction model.

The images with their respective labels are loaded into two arrays; ”bilder” and ”label”, and *face_resize* resizes all the images in the array to a fixed size, to assure all the images in the dataset are of the same size.

```
    def hent_modell(im_sz = (80, 100), person_navn  = None):
        """
        Denne metoden returnerer valgt model, som er brukt
        til læring og klassifisering.
        """
        # Only the LBP features support online learning. R = 1, P = 8
        lbp = ExtendedLBP(1, 8)
        # Block size chosen is  5x5
        feature = SpatialHistogram(lbp, sz=(5, 5))
        # Classifier chosen is the K-nearest neighbor with distance metric Chi-Square
        klassifiserer = NearestNeighbor(k=1, dist_metric=ChiSquareDistance())
        # Name of the classifier.
        navn = "NearestNeighbor"

        # Examples of other combinations are:

        # feature = Identity()
        # klassifiserer = KonvolverendeNeuralNettverk()

        # feature = fisherfaces()
        # klassifiserer = SVM()

        # Returnerer kombinasjon av klassifiserer samt feature extraction
        return UtvidetPrediksjonsmodell(navn = navn,
                                        lbp = lbp,
                                        feature=feature,
                                        klassifiserer=klassifiserer,
                                        image_size=image_size,
                                        subject_names=subject_names)
```

Listing 4: Python code to retrieve desired prediction model.

4.2 Graphical user interface

Although the GUI offers various features, the main purpose of the GUI is to connect to the robot. Figure 4.2 shows a screen shot taken of the GUI while the learning module is running. Here the chosen camera is set to web camera, meaning the robot is not connected in this instance.

The frame on the right shows the camera output with the face successfully detected and framed within a rectangle, with additional information about the learning phase; 2 out of 15 feature frames have been captured.

The graph on the bottom left shows the χ^2-distance score, see section 2.3.1, for each frame. Here the distance score steadily drops over time while learning the unknown face.

Another screen shot of the GUI shown in figure 4.3 displays the prediction module. Here the user is successfully recognised with in total 19 persons stored in memory[1]. Additionally the user is making a face in which the features seem to have not been extracted, judging from the small spike regarding the distance score in the graph at the end.

[1]The counting starts from zero, hence why the GUI shows my ID as nr. 18.

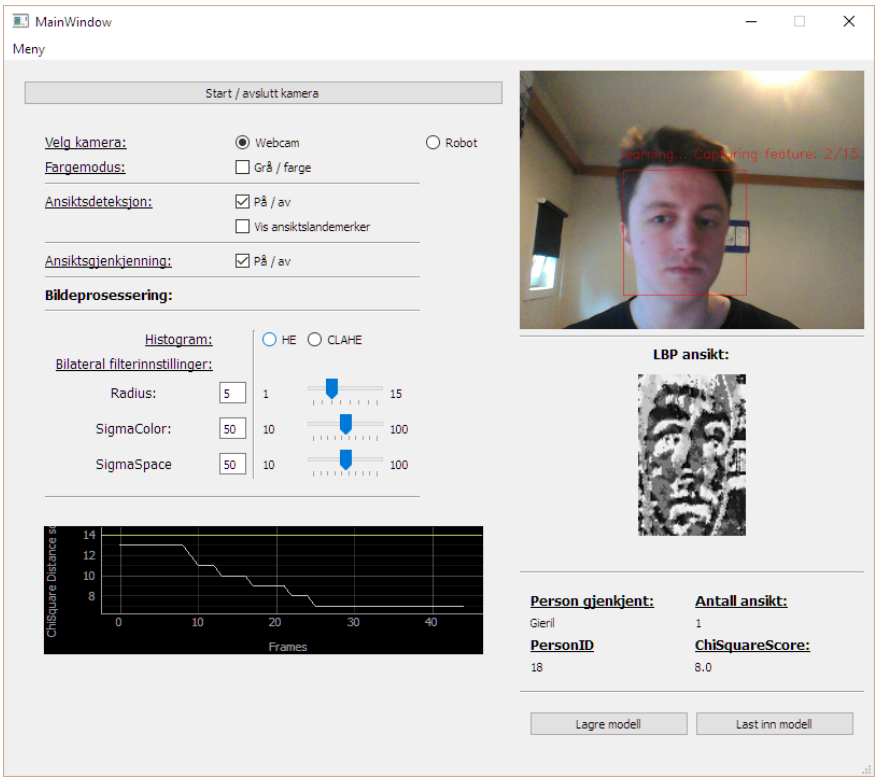


Figure 4.2: Graphical user interface showing the learning phase of the face recognition algorithm

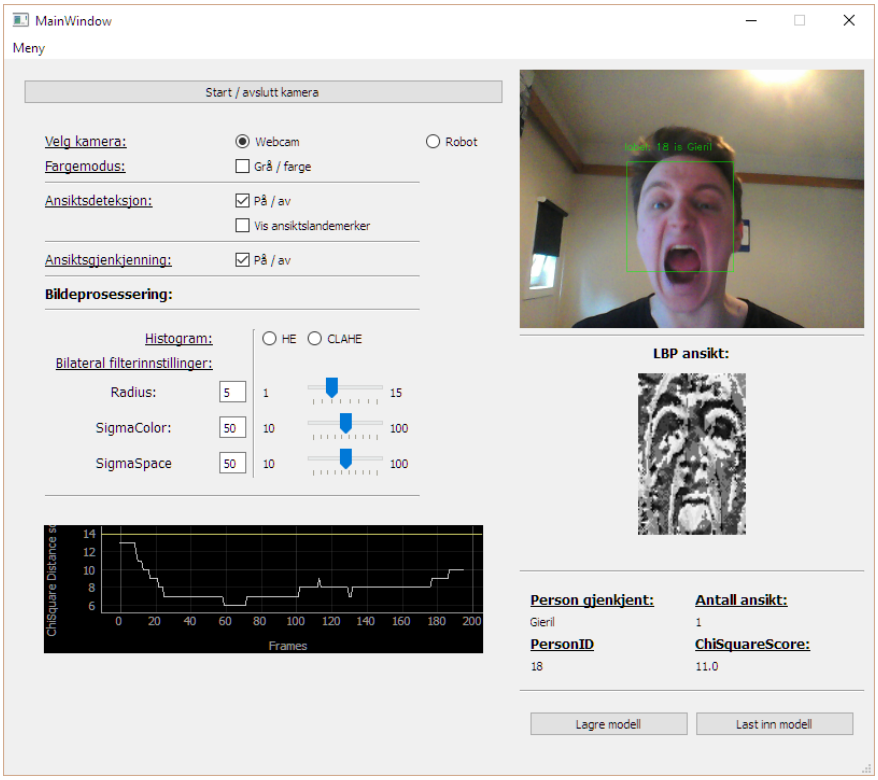


Figure 4.3: Graphical user interface showing a successfully recognised face.

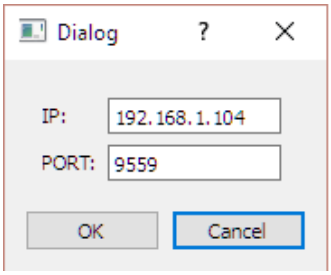
4.2.1 Saving and loading a pre-trained model

If the user has trained numerous unique faces in a session and does not want to re-learn the faces again after restarting the program, it is possible to save the current trained model. This can be done by simply clicking the "Lagre modell" button in the bottom right corner. This will save the prediction model to a file named *model.pkl*. To load the saved model, click the "last inn modell' button, if there are no saved models yet, an error message will appear.

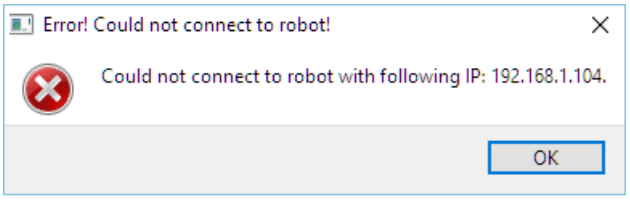
For best performance regarding frame rate it is recommended to activate the colormode grey. This will slightly improve the processing speed and will not affect the performance of the face recognition system itself, as it only changes the colormode of the frame shown in the GUI.

4.2.2 Connecting to the robot

To connect to a robot, access the menu at the top left corner and click "koble til robot". An input box prompting for the IP and PORT for the NAO robot will pop up. The default PORT value is already filled in, and can be left untouched. If the connection is successful the robot will say "I am connected", now the robot camera can be chosen and started.



(a) Input box used to connect to the robot.



(b) If connection fails.

Figure 4.4: Connecting to the robot.

4.3 Image acquisition

Image acquisition is done by either capturing frames from the web camera or from the robot camera. The acquisition and performance from the web camera will not be discussed. Once the robot camera is started in the GUI the following code is ran:

```
1     def registerRobotClient(self):
2             try:
3                     # Sets the resolution used for the camera
4                     resolution = vision_definitions.kQVGA  # 320 * 240
5
6                     # Sets the desired colorspace for the camera
7                     colorSpace = vision_definitions.kBGRColorSpace
8                     FPS = 30
9                     # Subscribes to the video module
10                    self.imgClient = self.videoProxy.subscribe("client",
11                     resolution,
12                     colorSpace,
13                     FPS)
14                    # Select camera.
15                    print("Setting camera parameters")
16                    self.videoProxy.setParam(vision_definitions.kCameraSelectID,
17                                                                    self.kameraID)
18            except IOError as e:
19                    print "I/O error({0}): {1}".format(e.errno, e.strerror)
20            except:
21                    print "Unexpected error:", sys.exc_info()[0]
22                    raise
```

Listing 5: Python code to register to robot proxies

Here the application subscribes to the video module on the robot and camera parameter's are set according to the theory, see section 2.1.

The colourspace is set to BGR, reason for this is because the pre-processing steps, explained in section 2.2, are done by using the OpenCV library in which the default colorspace is BGR.

The retrieved container from the robot camera is an array as seen in listing 6. This array is useless as it is and has to be reshaped using info about the width, height, number of layers and the binary array containing the image data, this is done as shown in line 23 in the mentioned listing.

```
        def reshapeImageFromRobot(self, robotimg):
            """
            he container retrieved is an array as follows:
            [0]: width.
            [1]: height.
            [2]: number of layers.
            [3]: ColorSpace.
            [4]: time stamp (seconds).
            [5]: time stamp (micro-seconds).
            [6]: binary array of size height * width * nblayers containing image data.
            [7]: camera ID (kTop=0, kBottom=1).
            [8]: left angle (radian).
            [9]: topAngle (radian).
            [10]: rightAngle (radian).
            [11]: bottomAngle (radian).

            To make the image usable and possible to process it has to be reshaped.
            :return: returns the image in a usable format
            """
            try:
                if (robotimg is not None):

                    img = np.reshape(np.frombuffer(robotimg[6], dtype='%iuint8' % robotimg[2]),
                                     (robotimg[1],
                                      robotimg[0],
                                      robotimg[2]))
                    img = cv2.cvtColor(img, self.color)

                    return img

            except BaseException, err:
                print("ERR: reshapeImageFromRobot: catching error: %s!" % err)
                return None
            except TypeError as e:
                print e
                raise
```

Listing 6: Python code to retrieve the next frame from chosen camera

4.4 Face detection and extraction

To detect a face in the acquired frame, a pre-trained Cascade-Classifier that comes with OpenCV is used, see section 2.2.3. The detector works fine with default parameter values for the scale factor and minimum neighbours, but after some testing these parameters were set to 1.2 and 3 respectively.

The scale factor sub samples the retrieved image by the set factoring number, meaning a scale factor of 2 would scale the image to half of it's current size. Having a smaller scale factor would assure faces further away can be detected, but is more computationally expensive. The minimum neighbour parameter makes sure the detected face is indeed a face by requiring at least, in this instance 3, detections in the same neighbourhood to return a positive match.

The detector returns a set of coordinates that represents a rectangle around the detected face, these coordinates are used to extract the face from the image, as seen in listing 7

below. On line 13 the face is extracted from the current frame, and is then used as input for either the learning or prediction module, depending on the prediction flag[1].

```
1         def ansikt_frame(self, currentFrame):
2                     #Retrieve coordinates for detected face
3                 faces = self.face_cascade.detectMultiScale(currentFrame,self.scaleFactor,
4                                                                 self.minNeighbors)
5                  #If no face is detected, return the retrieved frame
6                 if len(faces) == 0:
7                     return currentFrame
8                 faces[:, 2:] += faces[:, :2]
9                 for i, r in enumerate(faces):
10                        # Rectangle coordinates around detected face is retrieved
11                    self.x0, self.y0, self.x1, self.y1 = r
12                    #Face image extracted from current frame
13                    face_image = currentFrame[self.y0 + 2:self.y1 - 2, self.x0 + 5:self.x1 - 5]
14                    # If recognition is turned on
15                    if self.recognise:
16                            #If prediction flag is True run prediction
17                        if self.run_prediction:
18                                    # Prediction module
19                            self.gjenkjenning(face_image)
20                    # Else run learning
21                        else:
22                            self.learning(face_image)
23                    else:
24                            pass
```

Listing 7: Python code to detect and extract faces in current frame.

[1]This differs from the flowchart presented in the beginning of the chapter, in the sense that the prediction flag is checked at this point instead of after feature extraction and classifying as shown on the chart. This is due to the steps for both learning and prediction module are exactly the same up to that point.

4.5 Image preprocessing

The extracted face image is then resized according to section 2.2.4, histogram equalized and finally filtered, as seen in listing 8. Example of a face after pre-processing is shown in figure 4.5. Here it can be seen that the local contrast in the image is enhanced, as well as that smoothing is applied by the bilateral filter.

```
def facepreprocessing(self, X):
    """
    Retrieves the current face image, and processes it.
    :param X:
    :return: returns a processed face image.
    """
    try:
            #Resize image
        ansikt = cv2.resize(X, self.model.image_size, interpolation=cv2.INTER_CUBIC)
        #Apply CLAHE
        ansikt = self.clahe.apply(ansikt)
        #Filter face image
        ansikt = cv2.bilateralFilter(ansikt, self.d, self.sigmaColor, self.sigmaSpace)
        return ansikt
    except IOError as e:
        print "I/O error({0}): {1}".format(e.errno, e.strerror)
    except:
        print "Unexpected error:", sys.exc_info()[0]
        raise
```

Listing 8: Python code to process current face image

(a) Face before processing (b) Face after processing

Figure 4.5: Image processing routine

4.6 Feature extraction and classifying

The feature extraction and classifying is done by calling the function *self.model.prediksjon* with the processed face image as input, as seen in listing 9. This function is defined as seen in listing 10, and this function returns the predicted label and the related distance score. For a detailed overview for the classifier class and feature extraction class, see Appendix B.

```
1     # Get predicted label and the distance score from prediction done on retrieved face image.
2     self.predicted_label, self.distanse = self.model.prediksjon(self.ansikt)
```

Listing 9: Python code to extract features

```
1  def prediksjon(self, X):
2      # Extract features from query image
3      q = self.feature.extract(X)
4      # Return extracted features to classifier
5      return self.klassifiserer.prediksjon(q)
```

Listing 10: Python code to extract features, classify and return a predicted label along with the distance score.

4.7 Prediction and learning module

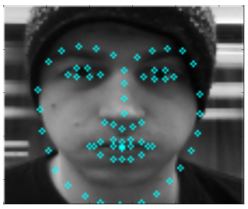
If a person is not known the learning module will be initialised. At first the function *NewPerson* will be ran, as shown in listing 11. Here the user will be prompted to type in his or her name in an input box, The name will then be appended to the models name list, where the name will be given an ID. This ID is used for both learning and prediction, at the end of the function the prediction flag will be set to False and the learning will start.

```
def newPerson(self):
    """
    This function handles if the detected
    face is not recognised. User will be asked to input their first name,
    and the learning module will start.
    :return:
    """
    # Get name from messagebox in GUI
    self.newName = ex.newperson()
    self.newName = self.newName.title()
    # Append given name to list.
    self.names.append(self.newName)
    self.soonflag = 1
            # If no users currently trained do this:
    if self.predicted_label == -1:
            #Add name of user to the prediction models name list
        self.model.subject_names.append(self.newName)
        # Retrieve the ID for the new name
        self.newlabel = self.model.subject_names.index(self.newName)
        #Update model with the current face frame, in conjuncting with the provided ID
        self.model.oppdater(self.ansikt, self.newlabel)
    if self.newName:
        # If user already exists in database, learn some more features
        if self.newName in self.model.subject_names:
            print "Navn finnes"
            self.newlabel = self.model.subject_names.index(self.newName)
            #Start learning module again, extract more features from face
            self.run_prediction = False
            #If robot is connected say the following:
            if self.kamera == 1:
                self.tts.say("Oh, it's you! Sorry for forgetting you,
                 lets just make me remember again!")
        else:
            # If user does not exist, add users name to database.
            #Set prediction flag to false --> run learning module
            self.run_prediction = False
            self.model.subject_names.append(self.newName)
            print(self.model.subject_names)
            self.newlabel = self.model.subject_names.index(self.newName)
            if self.kamera == 1:
                self.tts.say("All right," + str(self.newName) + ". Let me research your face.")
            else:
                pass
    else:
        self.distances = []
        self.run_prediction = True
```

Listing 11: Python code for adding a new person to the database

While the learning module is active, facial landmark coordinates are extracted from the face image. Example of facial landmarks is shown in figure 4.6a. These are used in the learning module to detect changes in pose and facial expressions.

This is important when key frames from a face are selected and used to update the model. These key frames should represent the face in varying poses and expressions, which leads to better performance and makes the system more robust. The facial landmark coordinates from the last 6 frames are stored, where the sixth and the newest coordinates are compared, if the change is big enough the face image will be used to update the model. Example of a set of key frames extracted is shown in figure 4.6b. Code showing how only key frames are used to update the model is shown in listing 12.



(a) Facial landmarks

(b) Keyframes extracted

```
    #If landmarks have been retrieved from the last 6 frames:
    if len(self.stored_landmarks) > 6:
        #Retrieve The absolute squared difference for
        #facial landmarks coordinates from current frame and 6th frame
        X_change = (np.mean((cv2.absdiff(self.stored_landmarks[6][:, 0],
                                         self.stored_landmarks[0][:, 0])))) ** 2
        Y_change = (np.mean((cv2.absdiff(self.stored_landmarks[6][:, 1],
                                         self.stored_landmarks[0][:, 1])))) ** 2

        # If the difference is above 15 for either x or y and the chi-square
        # distance is above the mean average for successfull classifications
        # then update the model with the current extracted face:
        if(X_change > 15 or Y_change > 15) and self.distanse[0] >=6 :
            self.model.oppdater(self.ansikt, self.newlabel)
            self.featureframes += 1
            print("Current key frame: {}".format(self.featureframes))

        # Or if the distance is above 9:
        elif self.distanse[0] > 9:
            # then update the model with the current extracted face:
            self.model.oppdater(self.ansikt, self.newlabel)
            self.featureframes += 1
            print("Current key frame: {}".format(self.featureframes))
        del self.stored_landmarks[0]
```

Listing 12: Python code to update model with key frames, based on facial landmark coordinates and the distance score.

Once in total 20 key frames are extracted and used to update the model, the learning module will end and normal prediction will resume. The prediction module will be ran

as long as the users in front of the robot are known, if an unknown person enters the frame, the distance score will most likely spike above the set threshold for unknown faces in which the learning module will be initialized again.

CHAPTER 5

Experiments & results

To evaluate the introduced methods, several experiments have been performed using two sets of face databases, these are briefly described in section 5.1. The images in the datasets were also resized to simulate the resolution of the face images extracted during real-time recognition. See section 2.2.4

With the parameters chosen in implementation regarding face detection, the time for face detection for each frame was found to be $0.013s$.

The performance is evaluated based on precision and speed. The Eigenfaces method was not evaluated as Fisherfaces has been shown to outperform this. The performance for Fisherfaces is shown in table 5.1, these results will be compared in the end.

Method	**Precision**
Fisherfaces	93.12% (±4.84)
Fisherfaces + HE	91.58% (±5.16)
Fisherfaces + HE + BF	91.30% (±5.24)
Fisherfaces + CLAHE	**94.20**% (±4.19)
Fisherfaces + CLAHE + BF	91.08% (±5.02)

Table 5.1: Performance for state of the art face descriptor with various image processing techniques. Evaluation for this method was done on the Yale face database

5.1 Datasets

Experiments were done on the two face databases mentioned in the following subsections.

5.1.1 AT&T face database

Also referred to as the ORL database of faces. It contains in total 40 subjects with then different images each. These images were taken at different times with varying light, facial expressions, poses and occlusions[1] The size of the images is 92x112 pixels, with 256 grey levels per pixels [9].

Figure 5.1: Sub sample of images from one subject in the AT&T face database.

5.1.2 Yale face database A

This database is slightly harder compared to the previous one, but also contains less subjects, and might be more suitable for experiments. This database consists of 15 subjects with 11 images each. In these there are drastic changes in light conditions, facial expressions and occlusions. The size of the images is 320x243 pixels. A sub sample of images in this database is shown in figure 5.2 below.

[1] With and without glasses on.

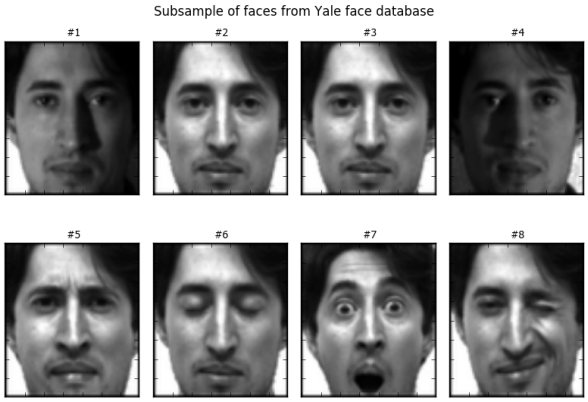


Figure 5.2: Sub sample of images from one subject in the yale face database.

5.2 Choice of K for K-NN classifier

Before the experiments are done, the K value for the K-NN classifier should be set. Maximising correct classifications is wanted, in other words the error rate for classifications should be minimized. Therefore the choice of K for the K-NN classifier should be where the mean error rate is at it's minimum.

For this an experiment was done with varying values of K. The results are shown in figure 5.3 below. Based on these results, it can be seen that $K = 1$ is the best choice.

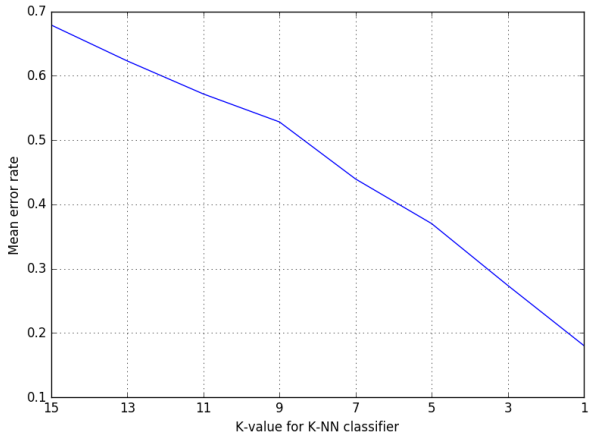


Figure 5.3: Mean error rate as a function of K - value

5.3 Experiment 1 - varying block size

The K-NN classifier with $K = 1$ is used, in combination with the χ^2 distance metric. This experiment focuses on varying the amount of blocks the image is divided into, no image processing except resizing is done. As mentioned in the theory chapter, section 2.3.1, $7x7$ blocks of window size $18x21$pixels[1] is recommended to achieve a good balance between recognition performance and feature vector length. The blocks tested range from $4x4$ to $9x9$. This experiment was done on the Yale face database.

Method	Blocks	Precision [%]	% Increase	Processing time [s]	% Increase
$LBP_{8,1}$	4x4	81.8 (±4.8)	-	0.0098	-
	5x5	83.5 (±5.2)	2.1^2	0.0112	14.3
	6x6	83.7 (±4.6)	2.3	0.0130	32.7
	7x7	85.3 (±5.7)	4.3	0.0151	54.1
	8x8	86.5 (±5.0)	5.7	0.0179	82.7
	9x9	86.7 (±4.8)	6.0	0.0218	122.5
$LBP^{u2}_{8,1}$	4x4	82.1 (±5.1)	-	0.0092	-
	5x5	83.1 (±5.1)	1.2	0.0100	8.7
	6x6	84.6 (±5.7)	3.0	0.0110	19.6
	7x7	85.5 (±4.9)	4.1	0.0120	30.4
	8x8	86.1 (±5.7)	4.9	0.0133	44.6
	9x9	86.2 (±4.9)	5.0	0.0146	58.7
$LBP^{u2}_{8,2}$	4x4	82.7 (±5.1)	-	0.0069	-
	5x5	83.2 (±5.1)	0.6	0.0084	21.7
	6x6	84.1 (±4.7)	1.7	0.0097	40.6
	7x7	86.0 (±4.8)	4.0	0.0109	60.0
	8x8	86.4 (±4.4)	4.5	0.0122	76.8
	9x9	**86.8 (±5.3)**	4.8	0.0134	94.2

Table 5.2: Performance for normal and uniform LBP descriptors with varying blocks.

The results from this experiment show that the LBP descriptor is quite robust with respect to differing block sizes. The precision increases slightly with larger block sizes for all types of the LBP descriptor. This is as expected, as more spatial info is preserved.

Based on the precision performance, $LBP^{u2}_{8,2}$ is the best choice as it outperforms the other two marginally. The standard deviation does not change much and remains roughly the same.

In regards of processing time, there's a massive difference. For the basic LBP descriptor,

[1]This would result in the LBP face images having a resolution of $7 \cdot (18x21)pixel = 126x147pixels$, which is not suitable.

the processing time increases by 122% from $4x4$ to $9x9$ blocks, compared to $LBP^{u2}_{8,1}$ with only an 58.7% increase. This is due to the length of the feature vectors created. For $LBP^{u2}_{8,1}$ the feature vector length for $9x9$ blocks is: 4779[1], while for the basic descriptor the length is 20736.

Considering that the application must be capable of running in real-time with best possible performance, the choice of method is simple. Even with $9x9$ blocks and adding the processing time for face detection, $LBP^{u2}_{8,2}$ is still capable of $\frac{1s}{0.0134s+0.013s} = 38FPS$, which is well above the desired value.

Although, seeing as the image processing steps have not been added yet, and the fact that reducing the block size from 9x9 to 7x7 only yields a loss of 0.8% precision for 22% better performance in respect of processing speed. Thus the choice is to use $LBP^{u2}_{8,2}$ with block size $7x7$ for further experiments.

5.4 Experiment 2 - Distance metric

This experiment is done to determine which distance metric is the best choice regarding both precision and speed. In the previous section the χ^2 distance metric was used for experiments and the results for this will be included here. This experiment was done on the Yale face database.

Distance metric	**Precision**[%]	**Processing speed** $[s]$	**FPS comparison**
χ^2	86.0 ($\pm$5.4)	**0.0109**	**42**
Euclidean	86.3 ($\pm$4.8)	0.0269	25
Hassan	**87.3** (**$\pm$4.7**)	0.0351	21

Table 5.3: Performance for the distance metrics

Both Hassan and Euclidean distance outperform χ^2 in regards of precision, butχ^2 distance is 2-3 times faster than the other metrics. Based on these results the χ^2 distance metric is chosen.

[1] $9x9$ blocks $x\frac{59bins}{block} = 4779$, for normal LBP the bin size is 256.

5.5 Experiment 3 - Applying image processing

The previous experiments were done on the Yale face database, which as mentioned is a slightly harder dataset, in this experiment both of the datasets will be used separately.

Now the proposed image processing techniques are applied to the face images. The results from this experiment will determine the final choices for the best possible performance, regarding both precision and speed, for the face recognition system.

Table 5.4 shows the results done on both of the datasets. Here, a difference is clearly seen regarding precision for the proposed histogram equalization methods.

Face database	**Method(χ^2 distance)**	**Precision**[%]
Yale face database A	$LBP^{u2}_{8,2}$	86.0 ($\pm$4.8)
	$LBP^{u2}_{8,2} + HE$	86.2 ($\pm$5.0)
	$LBP^{u2}_{8,2} + HE + BF$	82.6 ($\pm$5.1)
	$LBP^{u2}_{8,2} + CLAHE$	91.6 ($\pm$4.2)
	$LBP^{u2}_{8,2} + CLAHE + BF$	**91.7** (**$\pm$3.9**)
ORL database	$LBP^{u2}_{8,2}$	98.0 ($\pm$1.3)
	$LBP^{u2}_{8,2} + HE$	98.3 ($\pm$1.4)
	$LBP^{u2}_{8,2} + HE + BF$	**98.8** (**$\pm$1.6**)
	$LBP^{u2}_{8,2} + CLAHE$	96.5 ($\pm$1.0)
	$LBP^{u2}_{8,2} + CLAHE + BF$	98.0 ($\pm$1.3)

Table 5.4: Performance for various image processing techniques.

For the Yale face database, CLAHE with and without filtering outperforms normal HE by a large margin. Best precision is achieved with the $LBP^{u2}_{8,2} + CLAHE + BF$ combination with a precision of 91.7% against HE with 86.2%.

For the ORL database, its completely opposite. Now the normal histogram method outperforms CLAHE by a small margin with 98.8% precision against CLAHE with 98%, which is the same result as without any preprocessing done.

Based on these results it can be seen that the choice of image processing method , especially for histogram equalization, relies heavily on illumination conditions. If there are heavy changes regarding illumination, as shown in section 5.1.2, CLAHE works better as the local contrast is improved, in addition to edges being enhanced.

However, if the room is well lit where the light source is evenly distributed for the whole image, then normal histogram equalization is preferred, as this will increase the global contrast. In the GUI, it is possible to choose histogram equalization method accordingly.

The processing time to apply these techniques were in the $0.0002-0.0003s$ range, meaning these were negligible, thus after image processing the resulting FPS is 37.

Based on the experiments done the choices can be summarized as such:

1. Uniform LBP descriptor with $P = 8$ and $R = 2$, denoted as $LBP^{u2}_{8,2}$, was shown to perform best for both speed and precision.
2. The Hassan distance metric had the best performance in regards of precision, but was very slow. Thus, the χ^2 was chosen, because of the speed it provided.
3. The image processing techniques depend on the illumination. CLAHE works best under challenging circumstances, while normal HE is good where light is distributed evenly.

These decisions result in a performance highly satisfactory for both precision and speed. With precision ranging from 91.7% to 98.8% at 37 FPS, this is only limited by the image acquisition which varies from 11 to 30 FPS, depending on the network type. The precision performance was evaluated by using K-fold cross validation strategy, with $K = 5$.

5.6 Threshold value for unknown faces

As mentioned in the previous experiments, the $LBP^{u2}_{8,2}$ descriptor with the χ^2 distance metric was the best suitable combination for the face recognition framework with respect to precision and speed. Once a prediction is done, a predicted label along with the χ^2 score is returned. To be able to classify not recognized faces as unknown, a threshold is needed. For this an experiment was done combining the images from both datasets to estimate the mean score for correctly classified faces, and the same for incorrect classifications. Both datasets were used to ensure as many variances as possible regarding pose, illumination and facial expressions were included.

Classification	**Mean score**
Correct	7.14 ($\pm$1.65)
Incorrect	9.82 ($\pm$0.85)

Table 5.5: Mean score and std. dev. for correct and incorrect classifications

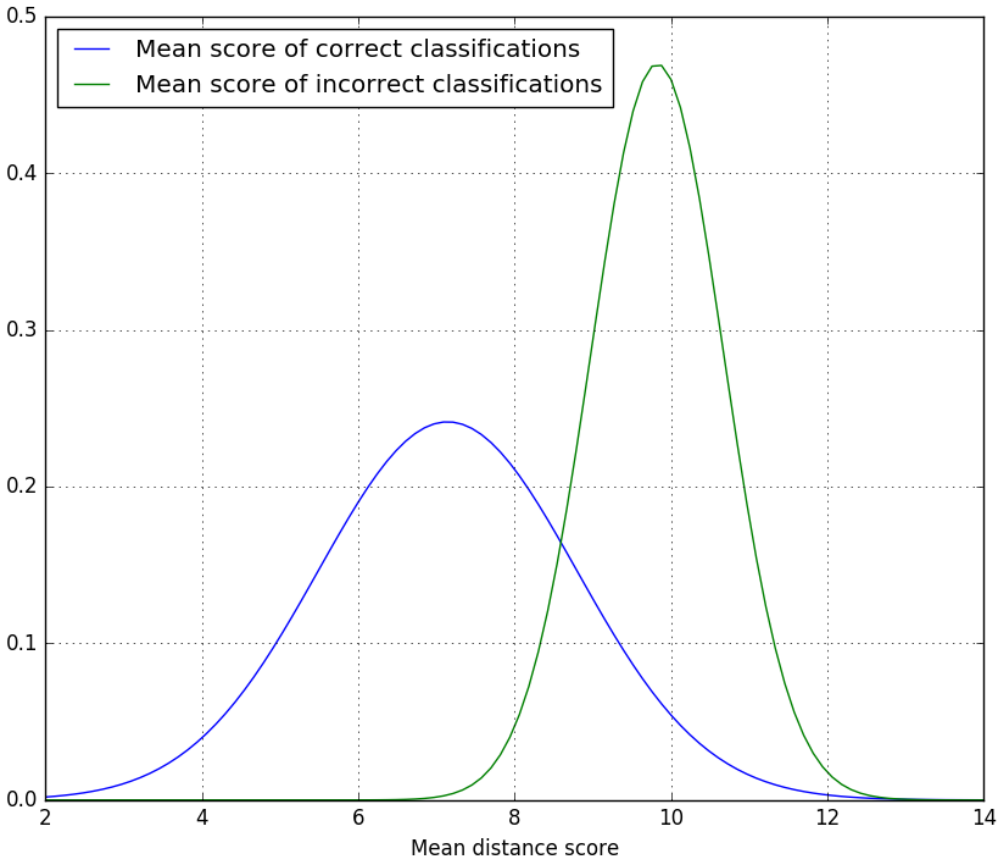


Figure 5.4: Normal distribution of the mean score for correct and incorrect classifications.

Table 5.5 along with figure 5.4 shows the results of these tests. As the facial expressions and pose can vary extensively in real-time recognition, and features might not have been extracted from all possible variations, the distance score might spike well above the mean for correct classifications. Therefore the threshold was set so that:

$P_{correct}(Z \leq x) = 0.95$

From Z-table for normal distribution this gives a threshold value at $7.14 + 1.64 \cdot \sigma_{correct} = 9.8$, which is roughly the same as the mean score for the incorrect classifications. This value was then further tested in real-time and provided satisfactory results.

CHAPTER 6

Conclusion & Future work

For a good HRI with focus on face recognition, the robot has to be able to adapt to new, unknown faces. Therefore the facial appearances along with the users name has to be learned. In this thesis, a face recognition framework capable of real-time face recognition and learning, for use on the NAO robot was proposed.

The introduced framework contains all the necessary steps ranging from detection to recognition of faces, all done in real-time. By tracking and using facial landmarks in the learning phase, only unique key frames representing the face in different poses and expressions are selected, which are then used to update the model on-the-fly. This adds an extra degree of robustness and performance. Having a pre-trained database of faces is not required for the developed system, but can be used if wanted. It is also possible to expand the prediction model with more classifiers and feaure extractors.

A graphical user interface was also made which adds a few features, in which saving and loading a pre-trained model might be the most important one.

Based on the experiments and results it was shown that the implemented face recognition system provides high performance in regards of both precision and speed.

6.1 Future work

The current implementation of the face recognition system for the NAO robot provides all the steps necessary for recognition of faces, but the interactive behaviour is severely lacking at the moment. This part of the system can be greatly improved by adding logic that handles interactions with recognised people and those who are not recognised.

Additionally, the current system performs poorly if there are several faces detected in one frame. This can be improved by adding logic that keeps track of the faces, as it is now the faces are not tracked but only detected in each frame.

Another feature that might be interesting is to keep track of the distance to users. The code for this is already implemented, but is not in use at the moment. The function is called *distance_to_camera* in the main.py file. What it does is calculate the distance from the robots camera to the detected face, this can be used to for instance tell users to come closer to the robot, adding a new level of interactive behaviour.

Detection of facial expressions and reacting accordingly is also possible. Facial landmark coordinates are already being extracted and tracked in the current implementation, but only to detect changes in pose and facial expression. To detect facial expressions like happiness, sadness, surprise etc., these coordinates can be used as input to train a Multi-Layer Perceptron for instance. Then if a recognised person is detected as being sad, the robot can react upon this, addressing the user by name and asking why he or she is sad.

Bibliography

[1] Edward H Adelson, Charles H Anderson, James R Bergen, Peter J Burt, and Joan M Ogden. Pyramid methods in image processing. *RCA engineer*, 29(6):33–41, 1984.

[2] Timo Ahonen, Abdenour Hadid, and Matti Pietikainen. Face description with local binary patterns: Application to face recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(12):2037–2041, 2006.

[3] Mouhammd Alkasassbeh, Ghada A Altarawneh, and Ahmad Hassanat. On enhancing the performance of nearest neighbour classifiers using hassanat distance metric. *arXiv preprint arXiv:1501.00687*, 2015.

[4] Continuum Analytics. Download anaconda now.

[5] Elham Bagherian and Rahmita Wirza OK Rahmat. Facial feature extraction for face recognition: a review. In *Information Technology, 2008. ITSim 2008. International Symposium on*, volume 2, pages 1–9. IEEE, 2008.

[6] Peter N Belhumeur, João P Hespanha, and David J Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):711–720, 1997.

[7] Chi-Ho Chan, Josef Kittler, and Kieron Messer. *Multi-scale local binary pattern histograms for face recognition*. Springer, 2007.

[8] The Qt Company. About qt.

[9] A.S. Georghiades, P.N. Belhumeur, and D.J. Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Trans. Pattern Anal. Mach. Intelligence 2*, pages 643–660, 2001.

[10] M. C. Amirani H. R. Eghtesad Doost. Texture classification with local binary pattern based on continues wavelet transformation. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, page 4651.

[11] Itseez. About opencv.

[12] JetBrains. Jetbrains toolbox subscription.

[13] Davis E. King. Dlib - python.

[14] Stan Z Li and Anil K Jain. Handbook of face recognition. page 3.

[15] University of Southampton. What is python? executive summary.

[16] Timo Ojala, Matti Pietikäinen, and Topi Mäenpää. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):971–987, 2002.

[17] OpenCV. Clahe (contrast limited adaptive histogram equalization).

[18] Sylvain Paris, Pierre Kornprobst, Jack Tumblin, and Frédo Durand. A gentle introduction to bilateral filtering and its applications. In *ACM SIGGRAPH 2007 courses*, page 1. ACM, 2007.

[19] Matti Pietikäinen. Local binary patterns.

[20] Aldebaran Robotics. Alvideodevice.

[21] Aldebaran Robotics. Alvideodeviceproxy mono stream management.

[22] Aldebaran Robotics. Nao - technical overview.

[23] Aldebaran Robotics. Nao documentation.

[24] Aldebaran Robotics. Nao video camera.

[25] Aldebaran Robotics. Python sdk.

[26] Aldebaran Robotics. What is naoqi framework.

[27] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015.

[28] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deeply learned face representations are sparse, selective, and robust. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2892–2900, 2015.

[29] Darryl K. Taft. Jetbrains strikes python developers with pycharm 1.0 ide.

[30] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2014.

[31] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86, 1991.

[32] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[33] Wenyi Zhao, Rama Chellappa, P Jonathon Phillips, and Azriel Rosenfeld. Face recognition: A literature survey. *ACM computing surveys (CSUR)*, 35(4):399–458, 2003.

APPENDIX A

Implementation of software

This appendix presents the setup needed to run the application. The application runs fine on Windows 10, other operative system's have not been tried. The guidelines shown should be followed in order.

Using the command prompt is needed in some of these steps, to open / start a command prompt see: http://www.digitalcitizen.life/7-ways-launch-command-prompt-windows-7-windows-8

A.1 Installing the Python distribution

The Python distribution used in this thesis is Anaconda, which includes various libraries used in this thesis. It can be downloaded directly from following link: http://repo.continuum.io/archive/Anaconda2-4.0.0-Windows-x86.exe.

Alternatively at their download page: https://www.continuum.io/downloads, be sure to choose the Python 2.7 installer for windows 32-bit.

A.2 Adding the NAO - Python SDK

The NAOqi Python SDK is needed to communicate with the robot remotely with Python. It can be downloaded from The Aldebaran Robotics homepage at: users.aldebaran-robotics.com. A username and password is required, these are provided by the institute. Once logged in click resources and then software, scroll down and locate the file *Python 2.7 SDK 2.1.4 Win 32 Binaries*.

A.3 Adding the OpenCV library

OpenCV is needed for many of the image processing steps. This library can be downloaded at their website: http://opencv.org/downloads.html. Make sure to download version 3.1 released on 21.12.2015. The installer should automatically locate your installed Python distribution.

A.4 Installing Qt

This can be done in two ways. If the Anaconda distribution is installed the easiest solution is to start CMD and type in the following: *conda install -c asmeurer qt=4.8.6*

This should install the required libraries for Qt. Alternatively if this does not work, go to the following website: http://download.qt.io/archive/qt/4.8/4.8.6/, and download *qt-opensource-windows-x86-mingw482-4.8.6-1.exe*, but do NOT install before installing the GNU compiler in section A.4.1

A.4.1 Minimalist GNU for Windows

A GNU compiler is needed to install Qt if the first option didn't work. This is done by installing MinGW, which can be retrieved from: https://sourceforge.net/projects/mingw/files/MinGW/, download the latest version and install the 32-bit version.

A.5 Adding the Dlib library

This library is used to track and detect facial landmarks, used in the learning module. To add this library open CMD and type in: *conda install -c menpo dlib=18.18* and hit enter.

In addition a pre-trained facial shape predictor must be downloaded from the following link:
http://sourceforge.net/projects/dclib/files/dlib/v18.10/shape_predictor_68_face_landmarks.dat.bz2

Add this file in the same directory as the *main.py* file.

A.6 Adding the Pyqtgraph

To show the graph on the ui Pyqtgraph is needed. Open CMD and type in: *conda install -c ufechner pyqtgraph=0.9.10* and hit enter.

A.7 Using the program

After the previous steps are done the application should be usable. To run the application, open the *main.py* file and simply run it. No extra commands are needed.

APPENDIX B

Implementations in Python

This appendix presents various parts of the face recognition code implemented.

B.1 Prediction model

The class that handles the prediction, and updating the classifier is shown here.

```
1       class Prediksjonsmodell(object):
2           def __init__(self, navn, feature, klassifiserer):
3               if not isinstance(feature, GenerellFeatures):
4                   raise TypeError("feature must be of type AbstractFeature!")
5               if not isinstance(klassifiserer, GenerellKlassifiserer):
6                   raise TypeError("classifier must be of type AbstractClassifier!")
7               self.navn = navn
8               self.feature = feature
9               self.klassifiserer = klassifiserer
10
11          def kalkuler(self, X, y):
12              features = self.feature.kalkuler(X, y)
13              self.klassifiserer.kalkuler(features, y)
14
15          def prediksjon(self, X):
16              # Extract features from query image
17              q = self.feature.extract(X)
18              # Return extracted features to classifier
19              return self.klassifiserer.prediksjon(q)
20
21          def oppdater(self, X, y):
22              q = self.feature.extract(X)
23              self.klassifiserer.oppdater(q, y)
24          def __repr__(self):
25              feature_repr = repr(self.feature)
26              classifier_repr = repr(self.klassifiserer)
27              return "Prediksjonsmodell (feature=%s,
28               Klassifiserer=%s)" % (feature_repr, classifier_repr)
```

Listing 13: Python code for prediction model

B.2 Classifier class

This class handles the classifiers, every classifier implemented, must be a subclass of the class listed in 14.

```
1       class GenerellKlassifiserer(object):
2
3         def kalkuler(self, X, y):
4             raise NotImplementedError("Hver Generellklassifiserer maa implementere kalkuler metoden.")
5
6         def prediksjon(self, X):
7             raise NotImplementedError("Hver Generellklassifiserer maa implementere prediksjon metoden.")
8
9         def oppdater(self, X, y):
10            raise NotImplementedError("Denne klassifiserer kan ikke oppdateres")
```

Listing 14: Python code for the general classifier class

B.2.1 K-NN classifier

The K-NN classifier is a subclass of the *GenerellKlassifiserer* class, and is shown in listing 15 under.

```
class NearestNeighbor(GenerellKlassifiserer):
    def __init__(self, dist_metric = EuclideanDistance(), k=1):
        GenerellKlassifiserer.__init__(self)
        self.k = k
        self.dist_metric = dist_metric
        self.X = []
        self.y = np.array([], dtype=np.int32)
        self.minimum  = None
        self.maximum = None

    def name(self):
        return "NearestNeighbor"
    def oppdater(self, X, y):
        """
        Updates the classifier.
        """
        self.X.append(X)
        self.y = np.append(self.y, y)


    def kalkuler(self, X, y):
         # Trains the classifier, if pre-training is needed
        self.X = X
        self.y = np.asarray(y)

    def prediksjon(self, q):
        distances = []
        for xi in self.X:
            xi = xi.reshape(-1,1)
            d = self.dist_metric(xi, q)
            distances.append(d)
        if len(distances) > len(self.y):
            raise Exception("Flere distanser enn klasser. Se klassifiserer.py fil.")
        distances = np.asarray(distances)
        # Get the indices in an ascending sort order:
        idx = np.argsort(distances)
        # Sort the labels and distances accordingly:
        sorted_y = self.y[idx]
        sorted_distances = distances[idx]
        # Take only the k first items:
        sorted_y = sorted_y[0:self.k]
        sorted_distances = sorted_distances[0:self.k]
        # Make a histogram of them:
        hist = dict((key,val) for key, val in enumerate(np.bincount(sorted_y)) if val)
        # And get the bin with the maximum frequency:
        try:
            predicted_label = max(hist.iteritems(), key=op.itemgetter(1))[0]
            # If the database is empty, return -1 as predicted label, triggering NewPerson()
        #If no pre-trained database exists
        except ValueError:
            predicted_label = -1
            sorted_y = 0
            sorted_distances = 0

        #return [predicted_label, sorted_distances[0]], with K = 1 The sorted_distances[0]
         #is only 1 value.
        return [predicted_label, { 'labels' : sorted_y, 'distances' : sorted_distances}]
```

Listing 15: Python code for the K-NN classifier

B.3 Feature class

This class handles the features, every feature implemented must be a subclass of the class listed in listing 16 below.

```
1       class GenerellFeatures(object):
2
3           def kalkuler(self, X, y):
4               raise NotImplementedError("Hver GenerellFeatures maa implementere kalkuler metoden.")
5
6           def extract(self,X):
7               raise NotImplementedError("Hver GenerellFeatures maa implementere extract metoden.")
8
9           def __repr__(self):
10              return "AbstractFeature"
```

Listing 16: Python code for the GenerellFeature class

B.3.1 Spatially enhanced histogram

This is a subclass of the *GenerellFeatures* class, and it handles the spatially enhanced histograms by taking the LBP image as input.

```
    class SpatialHistogram(GenerellFeatures):
        def __init__(self, lbp_operator = ExtendedLBP(), sz = (7, 7)):
            GenerellFeatures.__init__(self)
            if not isinstance(lbp_operator, LocalDescriptor):
                raise TypeError("Error, see features file.")
            self.lbp_operator = lbp_operator
            self.sz = sz
            self.lbp_name = self.lbp_operator.__class__.__name__

        def kalkuler(self, X, y):
            features = []
            for x in X:
                x = np.asarray(x)
                h = self.histogram_spatiallyEnhanced(x)
                features.append(h)
            return features

        def extract(self,X):
            X = np.asarray(X)
            return self.histogram_spatiallyEnhanced(X)

        def histogram_spatiallyEnhanced(self, X):
            # Kalkulerer LBP bildet.
            L = self.lbp_operator(X)
            # Kalkulerer blokkstorrelsene i bildet.
            lbp_height, lbp_width = L.shape
            grid_rows, grid_cols = self.sz
            #Hvis bildet har storrelse (100, 100)
            # og sz er valgt til (5, 5) vil denne returnere med py, px = 20, 20
            py = int(np.floor(lbp_height / grid_rows))
            px = int(np.floor(lbp_width / grid_cols))
            E = []
            if self.lbp_name == "ExtendedLBP":
                for row in range(0, grid_rows):
                    for col in range(0, grid_cols):
                        C = L[row * py : (row + 1 ) * py, col * px : (col + 1) * px]

                        H = np.histogram(C,
                                         bins = 2 ** self.lbp_operator.neighbors,
                                         range = (0, 2 ** self.lbp_operator.neighbors),
                                         normed = True)[0]
                        # probably useful to apply a mapping?
                        E.extend(H)
                return np.asarray(E)
            elif self.lbp_name == "uniformLBP":
                n_bins = L.max() + 1
                for row in range(0, grid_rows):
                    for col in range(0, grid_cols):
                        C = L[row * py: (row + 1) * py, col * px: (col + 1) * px]
                        H = np.histogram(C,
                                         bins=n_bins,
                                         range=(0, n_bins),
                                         normed=True)[0]
                        # probably useful to apply a mapping?
                        E.extend(H)
                return np.asarray(E)
```

Listing 17: Python code for the spatially enhanced histogram feature

B.4 Local descriptor class

This class handles the local descriptors, namely the LBP descriptors. Every local descriptor must be a subclass the class shown in listing 18.

```
1       class LocalDescriptor(object):
2           def __init__(self, neighbors):
3               self._neighbors = neighbors
4
5           def __call__(self,X):
6               raise NotImplementedError("Hver LBP Operator må implementere __call__ metoden.")
7
8           @property
9           def neighbors(self):
10              return self._neighbors
11
12          def __repr__(self):
13              return "LBP Operator (Naboer = %s)" % (self._neighbors)
```

Listing 18: Python code for the abstraft feature class

B.4.1 Uniform LBP class

The uniform LBP class is defined as shown in listing 19.

```
1       class uniformLBP(LocalDescriptor):
2           def __init__(self, radius=1, neighbors=8):
3               LocalDescriptor.__init__(self, neighbors=neighbors)
4               self._radius = radius
5               self.method = 'nri_uniform'
6
7           def __call__(self, X):
8               X = np.asarray(X)
9               resultat = local_binary_pattern(X, self.neighbors, self.radius, self.method)
10              return resultat
11
12          @property
13          def radius(self):
14              return self._radius
15
16
17      def __repr__(self):
18          return "Utvidet uniform LBP (Naboer = %s, Radius = %s)" % (self._neighbors, self._radius)
```

Listing 19: Python code for the uniform LBP descriptor

B.4.2 Extended(Circular) LBP class

The extended(Circular) LBP class is defined as shown in listing 20.

```
    class ExtendedLBP(LocalDescriptor):
        def __init__(self, radius=1, neighbors=8):
            LocalDescriptor.__init__(self, neighbors = neighbors)
            self._radius = radius

        def __call__(self,X):
            X = np.asarray(X)
            ysize, xsize = X.shape
            #Definerer sirkel
            angles = 2 * np.pi / self._neighbors
            theta = np.arange(0, 2 * np.pi, angles)
            # Kalkulerer punkt rundt sirkel med gitt radius
            punkt = np.array([-np.sin(theta), np.cos(theta)]).T
            punkt *= self._radius
            # finner boundaries of punkt funnet
            minY = min(punkt[:, 0])
            maxY = max(punkt[:, 0])
            minX = min(punkt[:, 1])
            maxX = max(punkt[:, 1])
            # Kalkulerer størrelsen på blokken,
            #Hver LBP koding er regnet ut innen blokkstørrelsen BsizeY * bsizeX
            blocksizeY = np.ceil(max(maxY, 0)) - np.floor(min(minY, 0)) + 1
            blocksizeX = np.ceil(max(maxX, 0)) - np.floor(min(minX, 0)) + 1
            # Origokoordinater i blokken (0, 0)
            origY =  0 - np.floor(min(minY, 0))
            origX =  0 - np.floor(min(minX, 0))
            # Kalkulerer størrelsen på output bilde
            dX = xsize - blocksizeX + 1
            dY = ysize - blocksizeY + 1
            # Finner midtpunkt
            C = np.asarray(X[origY : origY + dY, origX : origX + dX], dtype = np.uint8)
            resultat = np.zeros((dY, dX), dtype = np.int64)
            for i, p in enumerate(punkt):
                # henter koordinatene i blokk
                y, x = p + (origY, origX)
                # Kalkulerer floors, ceils og rounds for x og y.
                fx = np.floor(x)
                fy = np.floor(y)
                cx = np.ceil(x)
                cy = np.ceil(y)
                # Kalkulerer fraksjonene
                ty = y - fy
                tx = x - fx
                # Kalkulerer interpolasjonsvektingene
                w1 = (1 - tx) * (1 - ty)
                w2 =      tx  * (1 - ty)
                w3 = (1 - tx) *      ty
                w4 =      tx  *      ty
                # kalkulerer  interpolert bilde
                N = w1  * X[fy : fy + dY, fx : fx + dX]
                N += w2 * X[fy : fy + dY, cx : cx + dX]
                N += w3 * X[cy : cy + dY, fx : fx + dX]
                N += w4 * X[cy : cy + dY, cx : cx + dX]
                # Oppdaterer LBP
                D = N >= C
                resultat += (1 << i) * D
            return resultat

        @property
        def radius(self):
            return self._radius
```

Listing 20: Python code for the extended LBP descriptor

List of Figures

1.1 NAO robot - ©2012 Aldebaran Robotics. All rights reserved 2

2.1 Face recognition process flow. Source: [14] 5
2.2 Effects of Gaussian blur. Original image to the left, right image shows the effects of too much blurring; important facial information is lost. Source: [1]. 7
2.3 Example of results obtained with the bilateral filter. Source: [18] 8
2.4 Example of some features found by matching local contrast differences. The eye regions are usually slightly darker than the cheek regions, as shown in the centre. The image pair to the right shows the intensity difference between eye regions and the nose bridge. Source: [32] 9
2.5 LBP thresholding 10
2.6 Varying Radius and Points for LBP descriptor. Source: [10] 11
2.7 Face representation with Local Binary Patterns. Image source: [19] 13
2.8 Two class problem. 15

3.1 NAO features. ©Gigabotics 2016, all rights reserved. 18
3.2 The NAOqi broker tree. Broker-Libraries-Modules.Source: [26] ©2016 Aldebaran Robotics. All rights reserved 23
3.3 The NAOqi broker tree. Broker-Modules-Methods. Source: [26] ©2016 Aldebaran Robotics. All rights reserved 23

4.1 Simplified flowchart of the system. 28
4.2 Graphical user interface showing the learning phase of the face recognition algorithm 31

4.3 Graphical user interface showing a successfully recognised face. 31
4.4 Connecting to the robot. 32
4.5 Image processing routine 36

5.1 Sub sample of images from one subject in the AT&T face database. 42
5.2 Sub sample of images from one subject in the yale face database. 43
5.3 Mean error rate as a function of K - value 43
5.4 Normal distribution of the mean score for correct and incorrect classifications. 48

List of Tables

3.1 NAO technical overview. Source: [22] 19
3.2 Supported parameters for NAO camera. [24] 20
3.3 Supported resolutions [24] 20
3.4 Supported colorspaces [24] 21
3.5 Supported frame rates [24] 21
3.6 Observed frame rates [20] 21

5.1 Performance for state of the art face descriptor with various image processing techniques. Evaluation for this method was done on the Yale face database 41
5.2 Performance for normal and uniform LBP descriptors with varying blocks. 44
5.3 Performance for the distance metrics 45
5.4 Performance for various image processing techniques. 46
5.5 Mean score and std. dev. for correct and incorrect classifications 48