Master's Thesis

Gesture based interaction for a multi modal user interface

Anton Sebastian Roderich Skrabal, Bakk.rer.soc.oec.

skrabal@student.tugraz.at

Institute for Information Systems and Computer Media (IICM)

Graz, University of Technology

8010 Graz - Austria

&

Fachgebiet Mensch-Maschine-Systeme

Berlin, University of Technology

10587 Berlin - Germany

Supervisors:
Assoc. Prof. Andreas Holzinger, PhD, MSc, MPh, BEng, CEng, DipEd, MBCS
& Dr. Ing. Jeronimo Dzaack

Graz, May 2010

This page intentionally left blank

Masterarbeit
(Diese Arbeit ist in englischer Sprache verfasst)

Gestenbasierte Interaktion für ein multimodales Benutzerinterface

Anton Sebastian Roderich Skrabal, Bakk.rer.soc.oec.
skrabal@student.tugraz.at

Institut für Informationssysteme und Computer Medien (IICM)
Graz, Technische Universität
8010 Graz - Österreich
&
Fachgebiet Mensch-Maschine-Systeme
Berlin, Technische Universität
10587 Berlin - Deutschland

Betreuer:
Univ.-Doz. Ing. Mag. Mag. Dr. Andreas Holzinger
und Dr. Ing. Jeronimo Dzaack

Graz, Mai 2010

This page intentionally left blank

Abstract:

This master's thesis describes a method to automatically recognize hand gestures using a webcam to record the workspace, skin colour extraction from the LUV colour space, artefact removal using morphological image processing, a feed forward neural network and linear correlation for training and recognition. In the context of the Chair for Human-Machine Systems (FG MMS) at the Technical University Berlin and the Institute of information systems and computer media (IICM) at the Technical University Graz, a method was created to extract the user's hand from the webcam stream, identify significant points of the hand, build up sequences of movement and train a neural network on these data. For recognition of gestures, the previously trained network is fed with new incoming data. Furthermore, a linear correlation technique is used to verify the output of the neural network. Therefore, master templates are generated over the recorded movement sequences and data to classify is correlated to those master templates. The project was implemented in C++ using the libraries OpenCV for image processing and SNNL for neural network functionality. An extensive k-fold cross validation process was implemented as well, in order to prove validity of the project's methods. Results demonstrate the quality of the approach.

Keywords:

gesture recognition, colour space extraction, neural network, linear correlation, cross validation

Kurzfassung:

Diese Masterarbeit beschreibt eine Methode für die automatische Erkennung von Handgesten mittels einer Webcam, Hautfarbenextraktion aus dem LUV Farbraum, Artefaktbereinigung mittels morphologischer Bildbearbeitung, einem Feed Forward neuronalem Netzwerk und linearer Korrelation für das Training und die Erkennung. In Zusammenarbeit mit dem Fachgebiet Mensch Maschine Systeme (FG MMS) der Technischen Universität Berlin und dem Institut für Informationssysteme und Computer Medien (IICM) der Technischen Universität Graz wurde eine Methode entwickelt, um die Hand aus dem Videostream der Webcam zu extrahieren, signifikante Punkte der Hand zu identifizieren, Bewegungssequenzen zu erfassen und anhand dieser Daten ein neuronales Netzwerk zu trainieren. Weiters wurde eine lineare Korrelation implementiert, um die Ergebnisse des neuronalen Netzwerks zu verifizieren. Dafür wurden anhand der aufgenommenen Bewegungssequenzen Grundschablonen erzeugt, mit denen bei der Klassifizierung neue Daten korreliert werden. Dieses Projekt wurde in C++ implementiert, wobei die Bibliotheken OpenCV für die Bildverarbeitung und SNNL für die neuronalen Netzwerke verwendet wurden. Weiters wurde eine umfassende Kreuzvalidierungsmethode implementiert, um die Gültigkeit der verwendeten Methoden zu überprüfen. Die Resultate zeigen die Qualität der Herangehensweise.

Schlüsselwörter:

Gestenerkennung, Farbraumextraktion, Neuronales Netzwerk, Lineare Korrelation, Kreuzvalidierung

Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

Deutsche Fassung:

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

....
Datum Unterschrift

English version:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

....
date signature

Acknowledgments

I would kindly like to thank Univ.-Doz. Ing. Mag. Mag. Dr. Andreas Holzinger for his time and his help, Dr. Ing. Jeronimo Dzaack and Prof. Dr. Ing. Matthias Rötting from the TU Berlin for giving me the possibility to write this thesis, providing a workplace and introducing me to the field of gesture recognition, my father Univ. Prof. Dr. Falko Skrabal for his invaluable criticism, my sister Dr. Katharina Hollard-Skrabal for her expertise and the proof-reading of the thesis, my grandmother Dr. Zita Steingress, Mag. Manuela Schnalzer for her kind words and her invaluable input, Duncan Bare for proof-reading the thesis, my dear friends Alex, Alex, Boris, Daniel, Tobi and Wole for being there whenever I needed any kind of support and everybody I forgot.

For Doris

Abbrevations

ANN	Artificial Neural Network
ANNL	Artificial Neural Network Library
BCI	Brain Computer Interface
C	The programming language C
C#	The programming language C#
C++	The object oriented programming language C++
CIE	Commission Internationale de l'Eclairage
CIE XYZ	The colour space CIE XYZ
CMY(K)	The colour space CMY(K)
Ctrl-d	The keyboard key control plus the key d
Ctrl-e	The keyboard key control plus the key e
Ctrl-f	The keyboard key control plus the key f
e.g.	exempla gratia, used for: "for example"
EEG	Electroencephalography
ESC	The keyboard key escape
FFNN	Feed Forward Neural Network
FG-MMS	Fachgebiet Mensch-Maschine Systeme
GRM	Gesture Recognition Module
GUI	Graphical User Interface
HCI	Human Computer Interaction
HSL	The colour space HSL
HSV (1)	Human visual system
HSV (2)	The colour space HSV
i.e.	id est, used for: "it is" or "that is"
ibid.	ibidem, used for: "see last used citation for reference"
iCOMMIC	Integrated Controller for multimodal Interaction
Lab	The colour space Lab
LED	Light Emitting Diode
LLE	Locally Linear Embedding
LTSA	Local Tangent Space Alignment
LUV	The colour space LUV
MCP	McCulloch & Pitts
mEyeInt	multimodal Eye Interaction
ms	milliseconds
MSE	Mean Squared Error
NN	Neural Network
OpenCV	Open Source Computer Vision Library

PC	Personal Computer
PCA	Principle Component Analysis
px	pixel
RGB	The colour space RGB
SNNL	Simple Neural Network Library
UIS	User Information System
USB	Universal Serial Bus
XML	Extensive Markup Language
YCC	The colour space YCC
YES	The colour space YES
YIQ	The colour space YIQ
YUV	The colour space YUV

Contents

List of Figures

List of Tables

1 Introduction

Multi modal user interfaces allow a new approach to human computer interaction (HCI), meaning a more intuitive and more flexible way of communication between humans and machines. Differing from the contemporary form of interaction, namely that of using the mouse and keyboard, the multi modal approach allows humans to use their natural senses and means of communication, such as voice and gestures, setting the focus of attention (fixation of the eye) and maybe even thinking to interact with the interface. The multi-modal approach may allow new insights into human communicational behaviour as well as into more natural and hence more intuitive ways. As an additional important benefit, the abilities of disabled human-beings could be significantly improved via the introduction of new, handicapped-accessible interfaces.

This thesis is based upon an idea evolved in the FG MMS ("Fachgebiet Mensch-Maschine-Systeme", or translated, "Chair for Human-Machine Systems") at the Technische Universität Berlin. The idea is to build a system including different modules in order to control the computer with different modalities, straying from the mouse/keyboard concept which is predominant in our time. The system is called iCOMMIC, which stands for **i**ntegrated **CO**ntroller for **M**ulti **M**odal **I**ntera**C**tion. The work was conducted in association with the interdisciplinary group Team mEyeInt (**m**ulti modal **Eye Int**eraction) that has been developing the iCOMMIC system over the past few years (Dzaack et al., 2009).

As the integration of eye tracking systems is as good as done, the next modality is about to be introduced and this thesis holds the concepts, the design and the implementation documentation for this modality carrying the name **G**esture **R**ecognition **M**odule or short **GRM**.

1.1 Scope of this thesis

The scope of this thesis includes the analysis of how to implement, all of the relevant theoretical concepts in order to be able to implement and the implementation itself of a module for gesture recognition or rather gesture based interaction with a computer system in any form. The idea is to create a system that is able to understand hand gestures acted out in front of a simple webcam. Therefore, the image from the webcam is read, the hand extracted from the image (using skin colour as a distinguisher), relevant values selected and a machine learning algorithm is trained. After training, new gestures can be recognized by sending it's values to the machine learning algorithm, which classifies the incoming gesture. With the gesture recognized, a command is sent to the iCOMMIC system, which then performs previously defined commands like the pressing of a certain key.

Therefore, currently available similar systems are assessed, the main concepts thoroughly examined and possibilities of the required fields real-time image processing, machine learning and pattern recognition are explored. The goal is to create a modular and independent user interface module for hand gesture recognition that can easily be connected to the iCOMMIC for multi modal user interaction.

1.2 Structure of the thesis

The thesis is built up in five chapters. In Chapter 2, the theoretical concepts needed for the implementation of the system are introduced, including Section 2.1 to Section 2.3 presenting the basics of human computer systems such as human-computer interaction, gestures, and eye tracking methods. Furthermore, Section 2.4 describes required fields of image processing like colour spaces, filters, morphological operations and histograms. Chapter 2.5 demonstrates methods needed for pre-processing spatial and sequential data. Additionally, Section 2.6 explains the basic concepts of machine learning, especially neural networks. Section 2.8 elaborates on the basic concepts of linear correlation.

In Chapter 3, first the basic features of the system are analyzed, such as hardware analysis, time frame analysis, the hand tracking method and the analysis of the expert interviews in the basic analysis (Section 3.2). In Section 3.3, the requirements of the gesture recognition module are set up regarding usability, feedback mechanisms, interaction methods, recording, security, configurability and the selected gestures. Additional requirements in the form of external libraries used are discussed in Section 3.4 leading to the system concept as described in Section 4.1. This includes design documents, the idea of how to integrate the GRM into the iCOMMIC and the test setup. Section 4.2 describes the configuration, the logging methods as well as the different program modes needed for the system.

Chapter 5 discusses the quality of the system, including verification, validation as well as the conclusions that can be drawn from the results. Chapter Chapter 6 draws final conclusions and points out a future outlook on the iCOMMIC program as well as the GRM.

2 Theoretical Concepts

This chapter provides the theoretical concepts that were used to create the Gesture Recognition Module, from theoretical concepts about communication in general to mathematical aspects which were used to implement the GRM system.

In order to solve the posed problem as posed in 1.1, several fields of interest have to be assessed thoroughly. As the idea is to create a real-time hand gesture recognition module by assessing a video camera, these fields include human-computer systems in general, hand gestures as well as eye-movement and -tracking for assessing how to be able to combine the two modalities. The more technical concepts are image processing for extracting the region of interest from the video stream as well as data-preprocessing for the data extracted from the video. Machine learning and last but not least validation methods for validating the correctness of the system conclude the necessary fields of interest.

2.1 Human-computer systems

The communication between man and machine has been an issue from the beginning of the information age. The first computers were fed information by punched tape, inputting basic binary data by paper slips also called punch cards. As computers evolved, the need for faster and more convenient input methods arose. Typewriter keyboards as well as the mouse were introduced and ever since have been the devices used for input. The current key layout of the keyboards however was patented as early as 1870 by Christopher Sholes, who invented the first commercially successful typewriter. The problem was that the initially alphabetic order of the keys constantly led to colliding and stuck type bars, so Sholes was forced to adapt his keyboard layout to reduce the possibility of jams and collisions (David, 1985). In other words, the current layout of the keyboard can be seen as a result of trying to optimize the writing process as far as jams and collisions are concerned, but not as far as speed is concerned, which has indeed remained unimproved or changed since the late 19th century. This means that the possibilities of human-computer interaction are limited to a more than 100 year old system in combination with the computer mouse introduced in the 1960s as a pointing device (English et al., 1965) and finally distributed in a marketed fashion in the early 1980s with the Xerox 8010 Information System (Smith and Alexander, 1988). This leads to the conclusion, that the methods of human-computer interaction may be improved significantly by introducing new modalities. This thesis aims at helping to confirm this hypothesis.

2.1.1 Human computer interaction

Human computer interaction (HCI) means the communication between human and computer and vice versa. This interaction covers user generated input as well as the response by the computer. Modern day computers are generally controlled using three main types of interaction by the user, namely textual and conceptual input, navigational input and manipulative input.

This thesis will mainly focus on navigational and manipulative input, textual input might however be the next modality included in the system currently developed at the FG-MMS at the TU Berlin, the iCOMMIC.

2.1.1.1 Textual and conceptual input

The main input modality for text and content is, as was already mentioned in Section 2.1, the keyboard. Other known text input systems include the virtual keyboard, punch cards (as used mainly in the 1960s and 70s), the stylus pen (mainly for mobile devices), data gloves and last but not least voice recognition software.

2.1.1.2 Navigational input

Navigational input signifies input concerning navigation on computer systems in order to set the focus of interest to a certain area of the system. This input is produced by pointing devices such as the computer mouse, touchscreens, touch-pads, trackballs, joysticks or eye tracking systems. Also, keys of the keyboard can be used for navigation on the screen. The main advantage of pointing devices is the quick and limitless movement potential. While the keyboard navigation cannot be used to quickly change the centre of focus on the screen, navigational devices allow for this movement to be completed swiftly. In a few studies (Schrepp, 2006; Van Buskirk and LaLomia, 1995; Sears et al., 2003), speech recognition is suggested for navigation, though these systems have to date not emerged as serious competitors for the mouse and keyboard combination.

Most pointing devices are two dimensional input devices, feeding back the relative movement of the last state to the computer. The navigation using either of the above mentioned pointing devices generally works in similar ways. A pointer is used to show the user the place of focus where manipulation of information can be executed at the moment. Most of current input devices hold some kind of manipulative device like buttons in order to directly act out physical gestures. In the case of the mouse, these include point, click and drag.

Additional navigational devices are (among others) light pens (used normally in combination with light sensitive displays), palm, foot and gyroscopic mice (gyroscopes sensing the movement of the mouse through the air), steering wheels (one dimensional pointing device only) and the Wii remote (sending infrared light to a receiver placed on top of the system to be controlled). (Tuttle, 1986; Schou and Gardner, 2007)

2.1.1.3 Manipulative input

Manipulative input denotes active user input in order to interact with things on the screen such as activating or deactivating buttons. Also, scrolling text, zooming images, grabbing a symbol in order to drag it as well as deleting certain components on screen can be seen as manipulative input. These actions could also be conducted by activating a certain button on the screen or pressing a particular keyboard key while having the focus on the desired object. In other words, manipulative input is any kind of input a user provides to the machine which is not navigational or textual in nature (Harrison et al., 1998).

2.2 Hand Gestures

First, we need to define what gestures are. The Oxford English Dictionary (2009) for example defines gestures to be expressive movements of any part of the body, which would include everything a human being could do with its body. More specifically, gestures are bodily movements used for communication, displaying feelings or thoughts. Gestures refer to expressive movements acted out when the use of language would interfere, when the communication partner is too far away or also to stress the spoken word. Gestures are defined as deliberately acted out movements, so nervous hair patting or clothing adjustments are not to be seen as gestures, even though those movements might show signs of feelings or thoughts (Turk, 2002).

2.2.1 Gesture based communication

There are two types of gestures, the codified and conventionalized form of gestures that can replace speech as well as the informal, non-codified hand movements, generated during the course of speaking (Goldin-Meadow, 1999). Speech-accompanying gestures may allow the speaker and the listener to reflect the thoughts and feelings of the speaker even though they may be relatively unexamined by speaker and listener. Speech-substituting gestures however are intentionally used to consciously communicate, namely to emit *emblems* in order to insult, praise or regulate the behaviour of a communication partner, as for example the flat raised hand as a sign for *stop*. Emblems, however, are rarely combined gesture strings and thus generally don't form a linguistic system (with a few exceptions like sign language in stock exchange). The manual modality can however support a system of gestures with a linguistic structure, as for example seen in sign language. This form of communication is generally based on the spoken language of the surrounding hearing culture and is structured at syntactic, morphological and phonological levels (ibid.). For example, in sign language (a form of silent communication used for example in military settings), several gestures might be combined to form whole sentences or commands and thus generate a linguistic system.

Many fields of research use the starting point of gestures and communication. However, in this work the focus will be set on intentionally posed emblems without attention paid to the linguistic system. See the Section 6.2 for more on this topic.

2.3 Eye movement

Eye movements consist of both the movement of the eyeball as well as the moving of the eyelid and the pupil. The human eye is capable of moving in several different ways. Although those movements are always controlled by six muscles sitting on the outside of the eyeball, they differ by the stimuli causing the movement, the characteristics of the movement and the processes involved in steering the movements. Three classes have to be distinguished:

1. Saccades: Movement of the eye in order to prevent shifting of the information on the retina. Those movements are reactions to either movements of the body, of an object or of the surrounding environment.

2. Fixations: Target acquiring movements to orient the fovea onto (new) objects.

3. Microsaccades: Micro-movements of the eye.

For further reading please see both the works of Joos et al. (2003) and Roetting (1999).

2.3.1 Methods of eye tracking

Generally, eye tracking is based on anatomical and physiological properties of the eye which are discoverable in technical observations. Retinal and limbal properties, cornea-retinal potential, corneal curvature as well as reflections at different boundary areas of the di-optic apparatus can be used for assessment of the eye movement (Joos et al., 2003). The aforementioned reflections of a infra red light source included into the system are recorded via a stereo-camera pair and after a calibration process can be translated to a position the user is looking at on screen.

Since this thesis will focus on video based eye tracking only, other methods are volunteerly omitted.

2.3.1.1 Video based eye tracking

Using a video camera or another light sensitive sensor, an image of the eye is recorded where computer aided image processing extracts the most significant properties of the eye. In practical usage, infra red cameras have proven to be a usable method to track the movement of the eye (ibid.).

2.3.2 Scopes of application

Duchowski (2002) summarized the application of eye tracking where he reports from the domains neuroscience, psychology, industrial engineering and human factors, marketing/advertising and computer science.

Generally, Duchowski states that eye tracking has proven to be valuable in diagnostic studies of reading and other information processing tasks, which he considers to be the mainstay application. Also, as eye trackers are able to quantitatively measure real-time

overt attention, they are well suited for interactive systems, where especially for disabled users, eye tracking can be an essential and indispensable form of communication.

However, Duchowski also mentions ongoing debates about the idea of giving a perceptual organ like the eye an additional motorical task such as pointing. The eye might be more useful as an indirect indicator of the user's current intention than as an actual interaction method.

2.4 Image processing

Any form of signal processing where the input is an image like a photograph or video frame is called image processing. The output of image processing can be an image, extracted image characteristics or parameters related to the image. The image is usually treated as a two dimensional signal on which standard signal processing techniques are applied.

2.4.1 Digital image representation

Images are stored in computer memory in various ways. Generally, either a two dimensional matrix or an array of values is used, with other types of representations possible. Depending on the type of image (binary, grey scale, three channel, ...), the values inside the storage container vary. For example in binary images, the values are either 0 or 1, marking a foreground (1), or background (0) respectively. In three channel images (see 2.4.2 for more information) on the other hand, the values can consist of hexadecimal values with six digits (0x000000 for white, 0xFFFFFF for black and all values in between), each two digits representing one channel. Another representation for three channel images could also be a three layered two dimensional matrix, each layer holding the channel information consisting of real values depending on the definition of the colour space.

2.4.2 Colour spaces

The choice of colour space is a very important decision in image processing and can dramatically influence the outcome of the processing, thus it is important to make a careful choice of which colour space to use.

Colour is defined as the way the human visual system measures a part of the electromagnetic spectrum, namely the range between 300 and 830 nm wavelengths. For computer representation of colour, the following colour space classifications were introduced by Tkalcic and Tasic (2003):

- Human visual system (HSV) based colour spaces include the RGB (red, green, blue) colour space, the opponent colours theory based colour spaces and the phenomenal colour spaces. These colour spaces are motivated by the properties of the HVS.
- Application specific colour spaces including the colour spaces adopted from TV systems (YUV, YIQ), photo systems (Kodak PhotoYCC) and printing systems (CMY(K))

- CIE colour spaces are spaces proposed by the CIE (Commission Internationale de l'Eclairage, the International Commission on Illumination) and have some properties which are of high importance such as device-independency and perceptual linearity (CIE XYZ, Lab and LUV)

Some selected colour spaces are (with the others mentioned above being either really similar or not used frequently):

RGB

The most common way of image acquisition is done via digital cameras using the RGB colour space, which creates an image consisting of three channels, namely the red channel, the green channel and the blue channel. By super positioning these three channels, the actual colour image is generated.

YUV/LUV

A different principle is used in the YUV and LUV colour spaces, namely the concept of illumination or luminance. A YUV or LUV image also consists of three channels, but differing from RGB (where all three channels hold colour information), in the YUV colour spaces, the colour information is only stored in the U and V channel responsible for chrominance while the Y channel (or L channel respectively) holds the luminance or lighting for each pixel (Gomez, 2002).

YES

Another colour space is the YES colour space, where the luminance component (Y) is a weighted sum of the RGB values, while the chrominance factors are spectral differences: the signal in the E factor is proportional to the difference of the red and green colour channels, while the S colour factor is proportional to "yellow minus blue".

HSV/HSL

The HSV colour space is a representation of points in a RGB colour model that attempts to describe colour relationships more accurately than RGB. H stands for hue, a main component of colour, defined technically by Pattanaik et al. (1998) as "the degree to which a stimulus can be described as similar to or different from stimuli that are described as red, green, blue and yellow". S is short for saturation, meaning the difference of a colour against its own brightness and V finally stands for value or L for lightness (thus the alternative abbreviation HSL) meaning that a colour with a low value is nearly black while a colour with a high value is the pure, fully-saturated colour (ibid.).

2.4.2.1 Human skin colour detection

In order to extract the hand from the video, skin colour detection is an essential part of this thesis. This becomes even more important as the idea of the project is to develop a *hands-free* system, where no additional markers, gloves or similar auxiliary means should be used.

Selection of the most suitable colour space is essential for detecting skin colour in digital images. Skin colour detection by itself is a complex topic, as the colour of skin in an image is influenced by various factors. Ambient light, particularities, shadows, daylight and other factors can significantly influence the skin values. Also, different cameras return different values for the same scene and the same light conditions (Gomez et al., 2002).

As it is vital to select a fitting and usable algorithm for the extraction of skin colour from the video frames, the algorithms found in literature have to be thoroughly assessed.

The RGB colour space is widely used for skin colour detection even though RGB is quite sensitive to intensity variations. Here, two three dimensional histograms are computed, one for skin and the other for non-skin areas. Dividing every slot by the total number of elements, associated probabilities on an [r,g,b] index are received. The conditional probabilities for skin and non-skin are hence:

$$P(rgb|skin) = \frac{Hist_{skin}[r,g,b]}{Total_{skin}} \tag{2.1}$$

and

$$P(rgb|!skin) = \frac{Hist_{non-skin}[r,g,b]}{Total_{non-skin}} \tag{2.2}$$

where a new pixel can be labelled as skin or non-skin if it satisfies a certain threshold θ:

$$\frac{P(rgb|skin)}{P(rgb|!skin)} \geq \theta \tag{2.3}$$

Thus, the recognition ratio is a trade-off between reducing false positives and increasing correct skin classification.

The quality of this method was assessed in a recent survey which showed, that for 95% of skin detection 20% of false positives were identified, a naturally unsatisfying result (Brand and Mason, 2000). This could be related to the initial idea that skin colour forms a cluster in one single colour space, in this case the raw RGB space. Hence, a more appropriate method has to be found (Gomez et al., 2002).

A similar approach is using the YUV colour space, where the Y stands for luminance, while the U and the V channels hold the chrominance information. Luma (or luminance) represents the brightness in an image and is thus the "black and white" (or achromatic) portion of the image while the chroma (or chrominance) represents the colour information. This colour space is mostly used for compressing the information of brightness and colour in an image in order to optimize the performance for the human visual system. Chrominance is received by a simple transformation from RGB to YUV colour space, with $U = B - Y$ (blue - luma) and $V = R - Y$ (red - luma). When this colour space is used for transmission of data for television purposes, the chrominance bandwidth is generally decreased (as in digital systems by chroma sub-sampling; Valensi, 1961).

The usage of the CIELUV (also known as CIE(L*,U*,V*) or short LUV) colour space is based on a similar idea, where the L component holds the brightness portion of an image, while the U and the V components hold information about the colour information

itself (Smith and Guild, 1931).

Disregarding the luminance channel and only focusing on the colour components of an image is a very simple approach to pre-emptively eliminate lighting differences between scenes, so that the skin colour detection can be compiled based on this information only: colour based features of skin can easily be extracted by applying simple threshold algorithms on both U and V channel, however with a trade-off for accuracy to other known methods (Stark et al., 1995).

Another very powerful method for skin extraction is the combination of several different values from different colour spaces, as proposed in 2002 by Gomez et al. (Gomez et al., 2002). The idea is to combine the E values from the YES colour space, the ratio of red/green from the RGB colour space as well as the H values from the HSV colour space. This way, a three dimensional cluster is spanned where 95% of skin colour points share a nearly convex area with a minimal overlap to non-skin colour points. Using this method, a correct classification of 96% of skin points with only 11% of false positives can be achieved by sorting new points into this 3D raster resulting in a 95% overall recognition rate for both indoor and outdoor images.

2.4.3 Filters

Filtering is one of the most important fields of signal and thereby of image processing. Generally speaking, filtering is a process, which removes unwanted components from a signal, meaning that some frequencies are removed while others remain untouched in order to suppress interfering signals and reduce background noise. In image processing, filtering does not only affect the frequency domain, but can be used to additionally impact many other areas.

In the project described in this thesis, filters are used for thresholding the U and the V channel of the LUV colour space (threshold filters, 2.4.3.1), reducing background noise (Gaussian filters, 2.4.3.2) as well as edge detection (Sobel and Canny edge detection, 2.4.3.3).

2.4.3.1 Threshold filter

A threshold filter is a very simple filter setting where all image values exceeding a certain threshold and all image values below the threshold (to a minimum allowed value) for the component are assessed (see figure 2.1 for an example). There exist several variations of this filter, for example a two-threshold filter (also called hysteresis threshold filter), which sets all values lower than the lower bound to the minimum value and all values higher than the upper bound to the maximum value, leaving the values in between at the same level. This is often used to preliminarily segment an image into fields of interest (see figure 2.2 for an example). These threshold filters can be applied to a whole image or to certain channels of the image (if available) depending on the need of the processing step (Weeks, 1996).

Figure 2.1: An example of a threshold filter applied to a grey scale image, the original image on the left, the image processed with a threshold of $t = 128$ on the right. The values exceeding the grey-value threshold are set to the maximum (255), while the ones equal or below the threshold are set to zero.

Figure 2.2: An example of a threshold filter with upper and lower bounds (hysteresis threshold) applied to a grey scale image. The left image shows the original input image while the right picture was processed with a hysteresis threshold of $t_1 = 100$ and $t_2 = 150$. The values between those two thresholds remain unchanged.

2.4.3.2 Gaussian filter

"The Gaussian filter is a two dimensional convolution operator, with convolution meaning *multiplying together* two arrays of numbers of generally different sizes but the same dimensionality in order to produce a third array of numbers of the same dimensionality." (from Fisher et al., 1994). Convolutional operators in general are used to implement operators whose output pixel values are simple linear combinations of certain input pixel. In mathematics, the Gaussian function (named after mathematician Carl Friedrich Gauss)

is a function of the form:

$$f(x) = ae^{-\frac{(x-b)^2}{2c^2}} \tag{2.4}$$

for real constants $a > 0$, b, $c > 0$ and $e \approx 2.718281828$ (Euler's number).
The graph for a Gaussian function has the characteristic symmetric *bell* shape that quickly falls off towards plus/minus infinity. a is the curve's peak, b is the position of the centre of the peak and c controls the width of the *bell*. Generally used for statistics describing the normal distributions, the Gaussian function can be used as two dimensional filter in order to perform the Gaussian blur operation. This operation produces a smooth blur, typically used to reduce image noise and also to reduce detail. Since the Fourier transform of a Gaussian is another Gaussian, the Gaussian blur applied to an image reduces the image's high-frequency components and is thus a low pass filter.
The one dimensional Gaussian function has the form:

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{x^2}{2\sigma^2}} \tag{2.5}$$

where σ is the standard deviation of the distribution. In this case, it is assumed that the mean of the distribution is zero, meaning that it is centered about the line $x = 0$ as shown in figure 2.3 a). In two dimensional space, an isotropic (i.e. circularly symmetric) Gaussian has the form:

$$G(x,y) = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.6}$$

as shown in figure 2.3 b).

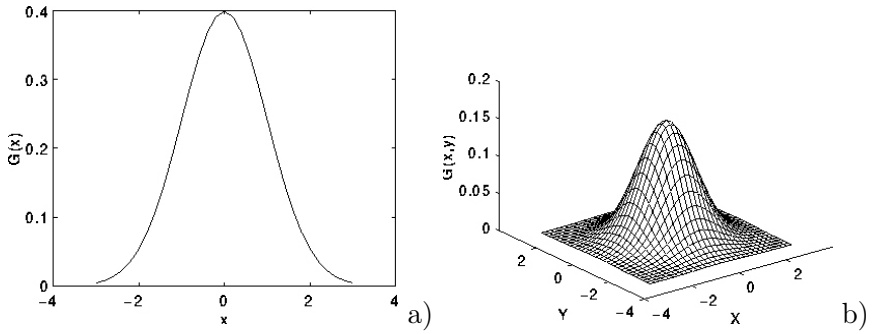


Figure 2.3: One and two dimensional Gaussian distributions

In theory, the Gaussian distribution would be non-zero everywhere to infinity which would require an infinitely large convolution element. In practical usage however, it is effectively zero from about the distance of three times the standard deviation from the mean, so it is possible to truncate the mask at this point. A suitable two dimensional integer valued mask that approximates a Gaussian function with $\sigma = 1.4$ is shown in figure 2.4.

$$\frac{1}{115}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Figure 2.4: Discrete approximation to a Gaussian function with $\sigma = 1.4$

The Gaussian function can be decomposed in such a way that the two dimensional convolution can be performed by first convolving with a one dimensional Gaussian in the x direction and then convolving with another one dimensional Gaussian in the y direction. It is because of that fact that the two dimensional isotropic Gaussian shown above is separable into x and y components. It is the only completely circular symmetric operator that can be discomposed in such a way. Figure 2.5 shows the 1D x component mask used to produce the full mask shown in figure 2.4, where the y component looks exactly the same but is vertically oriented.

$$\frac{1}{10.7}\begin{bmatrix} 1.3 & 3.2 & 3.8 & 3.2 & 1.3 \end{bmatrix}$$

Figure 2.5: The 1D x component convolution mask used to calculate the full mask shown in figure 2.4

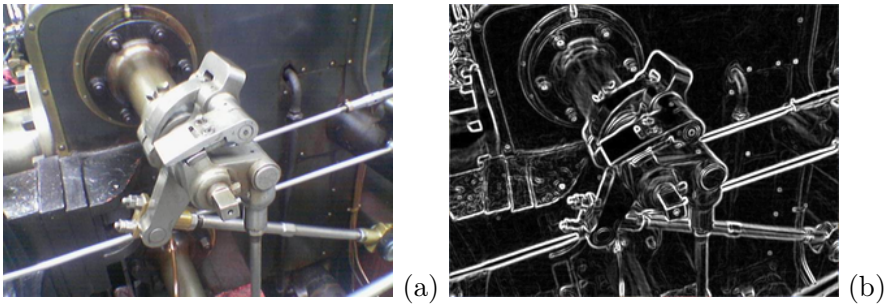
Using the Gaussian filter means smoothing the image (see figure 2.7 a) and c) for an application example). The degree of smoothness depends on the standard deviation of the Gaussian where larger standard deviations require larger convolution masks in order to be represented correctly. The Gaussian output is a "weighted average" of the neighbourhood of a pixel, with the average weighted to the centre of the mask, providing a focus on the pixel itself as well as the direct neighbours, but also taking into account the neighbors farther away. In this way, the Gaussian filter represents an excellent noise filter.

2.4.3.3 Edge detection

Edge detection is a very important field of image processing, as it can be used to find borders between areas of colour in the image. This can be used for point-of-interest detection by finding crossing points of lines, as well as to preliminarily winnow insignificant areas from the image. In this project, point-of-interest detection is implemented for better understanding the interesting points of the hand.

Sobel operator

The Sobel filter is a discrete differentiation operator, which computes the approximation of the gradient of the image intensity function invented by Sobel and Feldman (1968). The Sobel operator calculates the image intensities gradient for each point, returning the direction of the largest increase from light to dark and the rate of change in the calculated direction. In other words, the result of the Sobel operator is an image showing how *smoothly* or *abruptly* an image changes in colour at a certain point. In this way, the Sobel operator can be used as an edge detection operator, as it convolves the image with a small, separable and integer valued filter in both vertical and horizontal directions and thus returns an edge image holding all abrupt changes of the original image.



(a) (b)

Figure 2.6: The Soble operator applied to a colour image of a valve with (a) being the original image, (b) the output of the process. Images taken from WikipediaUser:SimpsonsContributer (2010)

Canny edge detector

Another edge detection filter is the canny edge detection operator invented by Canny (1986). This operator is a multi-stage algorithm to detect all kinds of edges in images. First, a Gaussian filter is applied (see 2.4.3.2), returning a slightly blurred image not affected by a single noisy pixel to any significant degree. Then, the canny algorithm uses four filters to detect horizontal, vertical and two diagonal edges in the blurred image. The filters used in this step might be any of the known edge detection operators like the above mentioned Sobel operator. From the image returned by this step, the edge gradient G and direction Θ can be determined as:

$$G = \sqrt{{G_x}^2 + {G_y}^2} \tag{2.7}$$

$$\Theta = \arctan(\frac{G_y}{G_x}) \tag{2.8}$$

where the edge direction angle is rounded to one of four angles representing vertical, horizontal and two diagonals (for example to one of the degrees 0°, 45°, 135°, 225°). Given these estimates of the image gradients, a search is carried out in order to assess if the gradient magnitude assumes a local maximum in the gradient direction. Referred to

as non-maximum suppression, a binary image is returned holding edge points which are sometimes referred to as "thin edges". This binary image is then filtered by a hysteresis threshold, where the high bound is applied first, marking out those edges that most probably are genuine. Using the directional information received earlier, edges can be traced throughout the image. While tracing an edge, the lower threshold is applied, allowing the tracing of faint sections of edges as long as a starting point is found. This way, a binary image is received with all the pixel marked as edge or as non-edge (ibid.).

The Canny operator has a couple of adjustable parameters affecting computational time and effectiveness of the algorithm, namely the size of the Gaussian filter, as smaller sizes cause less blurring and thus enable small, sharp line detection. Larger sizes cause more blurring, enabling larger, smoother edge detection. Also, the thresholds of the hysteresis threshold filter can be adjusted, where a threshold set too high can miss important information, while a threshold set too low will falsely identify irrelevant information such as noise. A generic threshold working well on all images has so far been difficult to find and no tested approach to this problem yet exists. See figure 2.7 for an example.

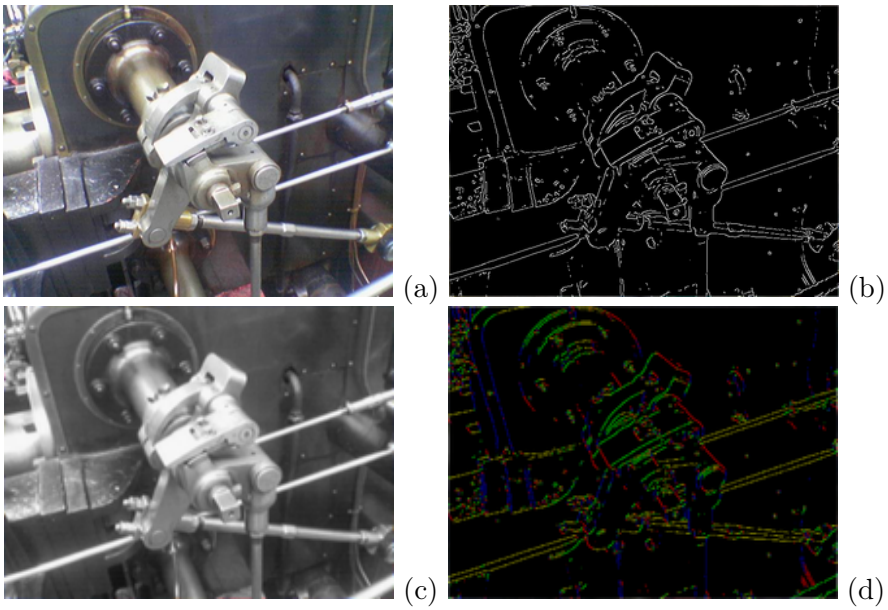


Figure 2.7: The Canny operator applied to a colour image of a valve. Image (a) is the original, (b) the result of the Canny operator. (c) and (d) show intermediate images, with (c) being the image after applying the Gaussian filter and (d) the binary image after thresholding with hysteresis (colour coded by direction). Images taken from WikipediaUser:SimpsonsContributer (2010).

2.4.4 Morphological operations

The term morphology when used in a biological context means the study of form and structure. Image processing has a slightly different definition of morphology. Mathemat-

ical morphology is a branch of nonlinear image processing and analysis that concentrates on the geometric structures within an image (Comer and Delp, 1999).

The scope of morphology includes various image processing methods including enhancement, segmentation, restoration, edge detection, textural analysis, skeletonization, shape analysis, compression and many more.

Morphological processing is geometrically based, meaning that an image is probed by a structuring element (which by itself is an image) in order to quantify how well the structuring element fits or does not fit. This fitting of course depends on the shape and size of the structuring element (probe), so a careful selection of the probe layout is needed. All morphological processing of images depends on fitting structure elements. In fact, there is only one primary operation in morphology, and that is a formal characterization of the probing concept as shown in figure 2.8 (Dougherty and Lotufo, 2003).

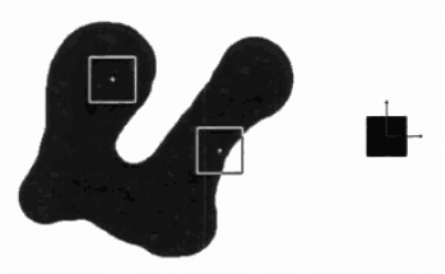


Figure 2.8: Probing an image A with a square structuring element B shown on the right

Marking the positions of where a probe fits or does not fit already returns basic structural information about the image.

2.4.4.1 Erosion

A fundamental operation of morphology is called erosion. In general, erosion means the gradual decline of something like soil (geology), which can be adopted in image processing as well. Erosion in image processing thus means the gradual decline of an area A by using a structuring element B, where only those areas of A remain where the structuring element B fits into A completely.

The erosion of a set A (image) by a set B (structuring element) is denoted by the term $A \ominus B$ and is defined by

$$A \ominus B = \{x : B_x \subset A\} \tag{2.9}$$

where $\subset$ denotes the subset relation. $A \ominus B$ thus consists of all the points x for which the translation of B by x fits inside of A. If B is called template, $A \ominus B$ consists of all template origin positions for which the whole template fits into A. Another way of describing (and also using) erosion is with robotics. Let a robot B be a probing element, trying to explore surrounding A. Wherever the robot is able to walk, he leaves a mark on the floor at the centre of his body. Since the robot is not able to reach the corners

of A, the marks he leaves are naturally smaller than A. See figure 2.9 a) for an example (ibid.).

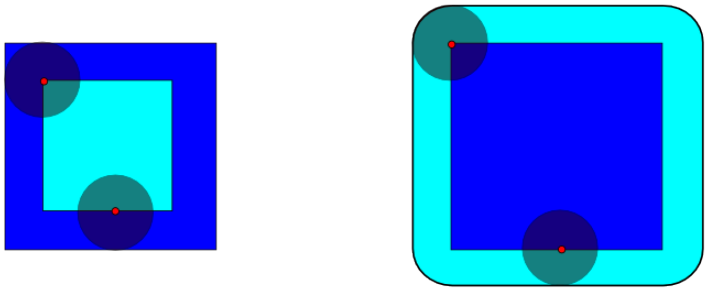
2.4.4.2 Dilation

Dilation literally means the stretching beyond normal dimensions and is hence the converse operation to erosion. Dilating an area A by a structuring element B thus signifies the act of expending A by moving the center of B, adding the exceeding area of the element B to A. An image A is convoluted and thus coiled or winded together with some structuring element B.

Dilation is denoted by the term $A \oplus B$ and is defined by

$$A \oplus B = (A^c \ominus \breve{B})^c \tag{2.10}$$

where A^c denotes the set-theoretic complement of A. To dilate A by B, B is rotated around the origin to obtain $\breve{B}$, after which A^c is eroded by $\breve{B}$. The result of the dilation $A \oplus B$ is then the complement of the erosion of A^c by $\breve{B}$. Since dilation involves a fitting into the complement of an image, it represents a filtering on the outside while erosion represents a filtering on the inside. The process can also be seen in a different way, namely that the centre of the structuring element B is moved on area A, and whenever a point is set in image A, the structuring element B leaves a full imprint of itself. See figure 2.9 b for an example (ibid.).



(a) (b)

Figure 2.9: Examples of morphological erosion (a) and dilation (b)

2.4.4.3 Opening

Opening is a morphological operation and means the dilation of the erosion of a set A by a structuring element B or expressed via the mathematical term

$$A \circ B = (A \ominus B) \oplus B \tag{2.11}$$

with the denotations $\ominus$ and $\oplus$ used as explained in 2.4.4.1 and 2.4.4.2, respectively.

Together with closing (see chapter 2.4.4.4), opening is a basic noise removal tool in image processing and computer vision. Small objects from the foreground are removed, placing them into the background. A good way to imagine opening is that the structuring element B sweeps along the inside of the boundary of A, so that it does not extend beyond boundary, and shaping the A boundary around the boundary of the element B (ibid.).

See figure 2.10 a for an example.

2.4.4.4 Closing

Closing is a morphological operation applying an erosion to the dilation of an image A by a structure element B. In mathematical terms:

$$A \bullet B = (A \oplus B) \ominus B \tag{2.12}$$

where $\oplus$ represents the dilation and $\ominus$ denotes the erosion as explained in 2.4.4.2 and 2.4.4.1 respectively. Closing removes small holes inside connected areas and connects previously unconnected areas at close range (ibid.).

See figure 2.9 b for an example.

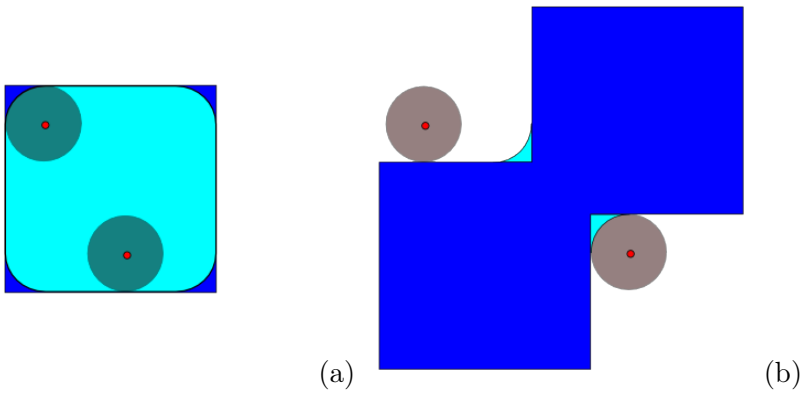


Figure 2.10: Examples of morphological opening (a) and closing (b).

2.4.5 Histograms

A histogram is a standard statistical description of a distribution of the number of occurrences of colours of a colour space as shown in figure 2.11.

Histograms (colour as well as grey scale histograms) are thus obtained by counting the number of times each colour (or grey level) occurs in the image. In general, a histogram consists of a horizontal and a vertical axis, with the horizontal axis depicting the different colors (or grey levels) while the vertical axis represents the number of occurrences of each colour/grey level (Swain and Ballard, 1990).

Grey scale images normally bring forth one histogram with horizontal values from 0 to 255 for the 256 grey levels applied. Figure 2.11 shows a grey scale image with its histogram, where a position close to 0 on the horizontal axis describes the number of

black or dark grey values, while a position close to 255 on the horizontal axis describes light grey and white values.

Colour images return an independent histogram for each of the channels used. Thus the RGB colour space would return three histograms, one for each the R, the G and the B channel.

This thesis will use only grey scale histograms, hence when the word histogram is used, it refers to grey scale histograms only.

Figure 2.11: A grey scale image and its histogram.

2.4.5.1 Vertical and Horizontal sum

Another way of representing (especially binary, also called black and white) images is counting the number of active pixel in each row and also in each column respectively, resulting in two sum images, the horizontal sum and the vertical sum. Figure 2.12 shows a real life example from the project belonging to this thesis. To the right and bottom of the image showing the extracted hand, one can see the vertical and the horizontal sums built over the lines and columns of the image (Søgaard and Olsen, 2003).

It is easy to see that the basic structural information returned by both the vertical and the horizontal sums can change significantly as the hand shape changes.

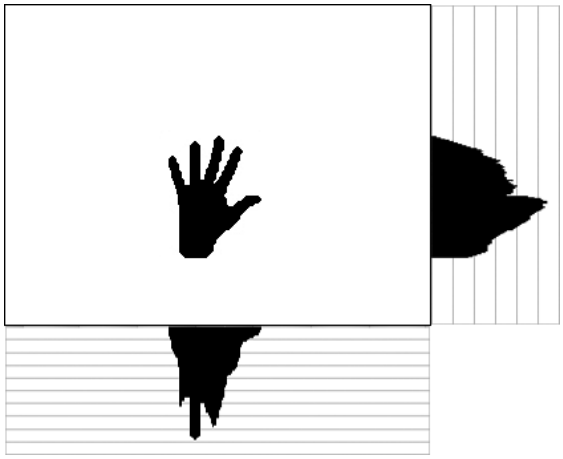


Figure 2.12: An extracted hand from a video frame with a resolution of 320x240pixel with its horizontal (right) and vertical (down) sums.

2.4.6 Bounding boxes

A bounding box is a box surrounding all points of a point set S with a minimum measure. This can be an area, volume or hyper volume in higher dimensions. In a two dimensional case, this box can be called minimum bounding rectangle, though in this thesis the term bounding box is also used for the 2D case.

Two different alignments of this box are possible: the *axis-aligned minimum bounding box*, where the edges of the bounding box are parallel to the axis of the Cartesian coordinate system. The other possibility is the *arbitrarily oriented minimum bounding box*, where the edges of the bounding box are not constrained to being parallel to the axis of the coordinate system, but can spread in any direction as long as all the edges of the box are either parallel or right-angled to each other (Barequet and Har-Peled, 1999).

2.4.6.1 Centre points

The centre of this bounding box can be easily computed by using vector arithmetics. Taking the lowest x and y values (and more if the dimension exceeds 2) values of the bounding box and adding half the distance to the highest x and y (and more if not 2D) will return the centre point of the bounding box.

$$\overrightarrow{C} = \overrightarrow{minP} + \frac{\overrightarrow{maxP} - \overrightarrow{minP}}{2} \tag{2.13}$$

where $\overrightarrow{C}$ denotes the centre point and $\overrightarrow{minP}$ and $\overrightarrow{maxP}$ denote the minimum and the maximum edge point of the bounding box.

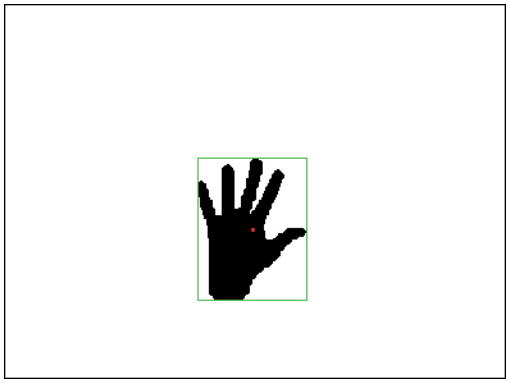


Figure 2.13: A point set S, it's bounding box in green and its centre point in red.

2.4.7 Image acquisition in C++

For this thesis, it is necessary to retrieve images from a camera in order to recognize the posed gesture. In C++, there are several libraries available on the Internet to read data from a camera. Generally, the camera is started by the C++ application and is requested to take a frame every few milliseconds. Depending on the library used, the image taken gets stored in a container object which may give various options on what to do with the image. Some libraries return the raw image in a 1D vector, while others return three 2D matrices, one for each colour channel, depending on the colour space used.

Many image acquisition libraries are actually image processing libraries, which bring several useful features with them, facilitating image processing. The focus on image acquisition and processing is set to speed, consistency and handling of errors, as well as the includability to C++ projects. Thus, it is wise to use libraries as mentioned above, since they are optimized with a special regard to exactly these requirements.

2.5 Data pre-processing

Data pre-processing is an essential method for reducing the dimensionality of data, cleaning of outliers or removing offsets in data. It is necessary to bring several data sets to equivalent bases so if classification methods are used, the different classes can be more easily distinguished. Several methods can be found in literature, such as offset correction, cluster analysis, dimensionality reduction, data transformation like normalization and aggregation as well as data discretization and data integration. A basic flow chart for data pre-processing and classification can be found in figure 2.14, displaying an optimization method for both data pre-processing and classification (Candolfi et al., 1999).

Some selected pre-processing methods are presented in 2.5.1 to 2.5.4.

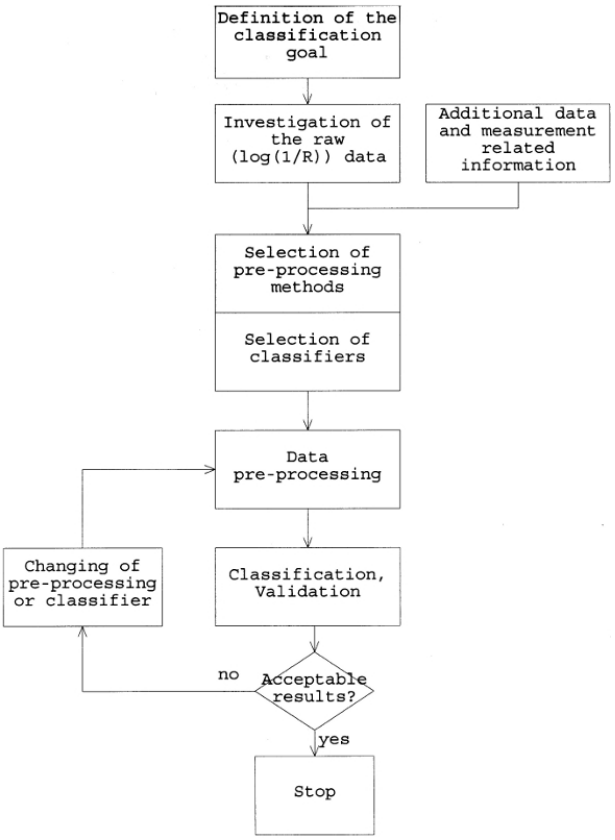


Figure 2.14: A flow chart for pre-processing and classification method optimization. Image taken from Candolfi et al. (1999).

2.5.1 Dimensionality reduction

Data is often received in high dimensional forms, such as matrices with thousands of values for each of the observations. These data often include redundant data or closely correlated data. As this redundancy can significantly slow down the classification methods and as it often ambiguously obscures the data, it is important to reduce the dimensionality by two main methods, namely feature selection and feature extraction.

Feature selection tries to find a subset of the original variables by either filtering or wrapping the data. The methods used here are for example the information gain, also called Kullback–Leibler divergence for filtering. For further readings see Kent (1983). Another method is the genetic algorithm for wrapping, see Mitchell (1998) for further readings.

On the other hand, feature extraction tries to map the original data from a multidimensional space down to a space with fewer dimensions. Some methods to be mentioned here are the PCA (Principle Component Analysis) as explained by Jolliffe (2002), the LLE (Locally Linear Embedding) as mentioned by Roweis and Saul (2000) as well as the LTSA (Local Tangent Space Alignment) as explained by Zhang and Zha (2004).

2.5.2 Offset removal

Offset removal can mean different things in different fields of application. For waveform analysis (e.g. signal processing) for example, a time offset often exists, which has to be removed in order to assess similarities of the waveform, namely amplitude, wavelength and so on. See Abidi (1995) for further readings.

On the other hand, and this is more relevant for the project described in this thesis, offset removal can mean "bringing the movements to the same starting or ending point", as movements are not always acted out on the same position in the camera's field of view. The idea here is to analyze the values of a movement (for example the movement of the center point of the bounding box) and subtracting the minimum value of the whole movement from all values both for the x-axis and for the y-axis separately. This leads to a new movement vector of the same size and form at a different place of the image as demonstrated in figure 2.15.

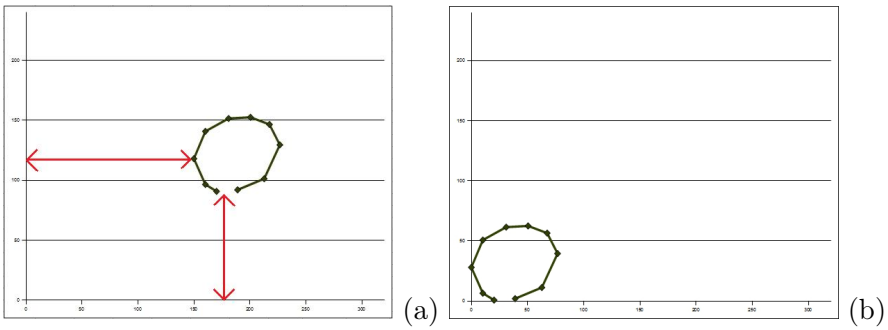


Figure 2.15: Offset removal of a movement. (a) showing the movement at it's original place, and (b) showing the new movement with the spatial offset removed (indicated by the red arrows in (a)).

2.5.3 Cluster analysis

Cluster analysis stands for a couple of methods for both classification as well as data pre-processing. Clusters are built over a data set - normally consisting of point sets in either 2, 3 or higher dimensional spaces - using manifold techniques. Outliers that cannot be classified as being of any introduced cluster can be filtered out easily, in case the clusters can be easily distinguished (Anderberg et al., 1973). Figure 2.16 shows a simple example where the outlying samples (not included in the encircled areas) are excluded from further usage.

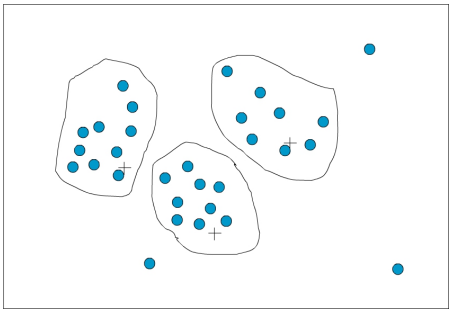


Figure 2.16: A simple example of a cluster analysis

2.5.4 Normalization, standardization

Normalizing or standardizing data means indicating how many standard deviations an observation is above or below the mean. To do so, some preliminary calculations over the whole data need to be conducted.

First, the population mean has to be calculated by summing up all values of all samples and then dividing the result by the number of values:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2.14}$$

Then, the standard deviation of the population has to be calculated by summing up the square of all the raw values of all of the samples minus the population mean, dividing the result by the number of values and finally taking the square root

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2} \tag{2.15}$$

Once those two values are calculated, a new value is normalized to a standard score (or so called z-score) by subtracting the mean value μ from the raw value to be standardized and finally dividing the difference by the standard deviation σ.

$$z = \frac{x - \mu}{\sigma} \tag{2.16}$$

This way, we receive a quantity z which represents the distance between the raw score x and the population mean μ in units of the standard deviation σ.

It is important to understand, that neither the sample mean nor the sample deviation can be used here, but both the population mean and population standard deviation are required. Often, knowing those two required values is unrealistic, so in a lot of cases - where the entire population can not be measured - the standard deviation may be estimated by using a random sample (Larsen and Marx, 2001).

Changing the interval

Another way to normalize data is by changing the interval in which they reside. For an interval $[-1, 1]$, this can be done by applying the formula

$$\hat{x} = \frac{x - midrange}{range} \tag{2.17}$$

to each of the values. The *midrange* and *range* stand for the formulas

$$midrange = \frac{(\min(\vec{x}) + \max(\vec{x}))}{2} \tag{2.18}$$

and

$$range = \max(\vec{x}) - \min(\vec{x}) \tag{2.19}$$

For this method, no knowledge about the the complete set is necessary, as only the current example is taken for further calculations. Using previously offset corrected data, *midrange* and *range* change to $midrange = \frac{\max(\vec{x})}{2}$ and $range = \max(\vec{x})$, as the minimum value of an examples is zero for all examples.

2.6 Machine learning

Machine learning is a scientific discipline for developing methods which allow a computer to learn complex relations based on data. It is mainly used to automate certain processes such as pattern recognition and to make intelligent decision based on data presented to the computer. Thus, machine learning is closely related to the fields of statistics, artificial intelligence, pattern recognition, data mining and probability theory (Michie et al., 1994).

2.6.1 Neural Networks

A neural network (NN), also called artificial neural network (ANN), is a computational paradigm whose structure is based on biological nervous systems, such as the human brain. Composed of neurones, highly interconnected processing elements, NN's learn by example. NN's are configured for each specific application by a learning process, implying adjustments of synaptic connections existing between the neurones.

Neural networks are often used to identify the intent in complex or imprecise data and are therefore able to extract patterns and detect trends too complex for human beings. Depending on the layout of the network (as described in 2.6.1.3), neural networks can generalize and classify well after a structured training (Haykin, 1994).

2.6.1.1 Simple Neurons

In the human brain, typical neurons collect signals from others through a host of fine structures, called dendrites (figure 2.17). The neuron sends out spikes of electrical activity through the main strand, called an axon. The axon then splits into thousands of branches and at each end connects with the dendrite of another neuron. This connection is called a

synapse. When a neuron receives enough input surpassing a certain inhibitory threshold (also received from other neurons), the neuron sends a spike of electrical activity down its axon. Learning now occurs via the changing of the influences of the synapses (Cross et al., 1995).

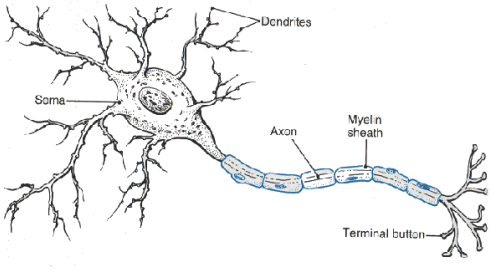


Figure 2.17: A neuron and its components. Image taken from CedarCrestCollege (2010).

The same basic idea is also used for artificial neurons, namely the essential features of neurones and their interconnections, where a computer is programmed to simulate these features. However, since the knowledge of biological neurones is incomplete and computing power is limited, the models generated are gross idealizations of real neural networks.

2.6.1.2 MCP Neuron and Perceptron

McCulloch and Pitts (1943) introduced a new model, the MCP neuron, since the simple version of an artificial neuron was not very useful . The difference from the previous model is the *weighting* of the inputs, meaning that the decision making is dependent on the weight of the particular input. This weight is a number which is multiplied with the input, resulting in the weighted input. These weighted inputs are then added together and if this sum exceeds a certain threshold, the neuron *fires*.

Adaptation to particular situations by this very flexible and powerful neuron is done by changing its weight and threshold. Various algorithms for this adaptation exist, for example the Delta rule (in feed-forward networks, see 2.6.1.5) and the back error propagation (in back propagation NNs, see 2.6.1.6) (McCulloch and Pitts, 1943). In 1958, Rosenblatt introduced a new concept building on the ideas of McCulloch and Pitts: the perceptron. The perceptron (figure 2.18) is essentially an MCP neuron where the inputs are first passed through some *pre-processors*, also called association units. These association units detect the presence of certain specific features in the inputs. In fact, as the name suggests, a perceptron was intended to be a pattern recognition device, and the association units correspond to feature or pattern detectors (Rosenblatt, 1958).

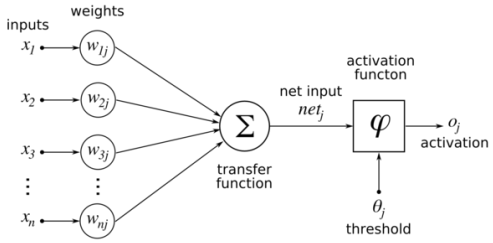


Figure 2.18: A single-layer perceptron. Inputs x_n are weighted by weights w_{nj}, summed up by the transfer function $\sum$. The activation function φ in combination with the threshold θ_j finally leads to the activation o_j. Image taken from WikipediaUser:Chrislb (2010).

2.6.1.3 Layouts of Neural Networks

Figure 2.19 shows the standard layout of a neural network. From left to right, we have the input layer, the hidden layer and finally the output layer returning the result. The hidden layer is so named because it works like a black box, processing data internally without the possibility to actually measure the outcome at the hidden unit directly. Of course, more than one hidden layer can be implemented (Haykin, 1994).

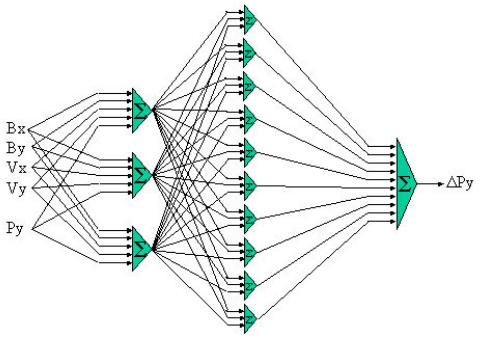


Figure 2.19: A simple feed-forward network. Image taken from Intel (2009).

Feed-forward Neural Networks

Feed-forward Neural Networks (FFNN's) allow signals to travel in one direction only, from input to output (figure 2.19). No feedback machanisms or anything similar are implemented, meaning that the output of one layer has no influence on the same layer. FFNN's tend to associate inputs to outputs and are thereby extensively used in pattern recognition (Haykin, 1994).

Feedback Neural Networks

In feedback NN's, a signal can travel both ways, back and forward in the network, meaning that the outcome of one layer may influence the same layer in the next step.

Their states are constantly changing until they reach a point of equilibrium. These NN's are also referred to as recurrent or interactive NN's (ibid.).

2.6.1.4 Activation functions

Activation functions are the functions that decide whether a neuron fires or not. Several different functions are used in neural networks, where the identity function is the most simple. The input equals the output. Another possibility is a threshold function (figure 2.20) which changes inputs higher than some threshold to a maximum value (normally 1.0) and inputs lower than that threshold to a minimum (normally -1.0 or 0.0). This is often used in perceptrons (Stone, 1986).

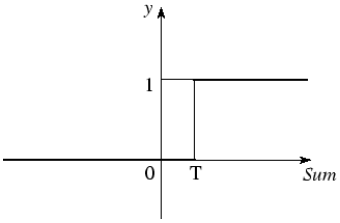


Figure 2.20: Linear activation function, hard transition.

A third activation function is a "soft" threshold function, also called sigmoid (figure 2.21 with x and y axis from -1 to 1, smooth transition), which smoothes out the values near the threshold, with the formula

$$g(h) = \frac{1}{(1 + e - h)} \tag{2.20}$$

where h is the input to the unit (ibid.).

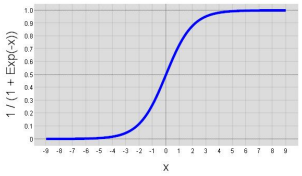


Figure 2.21: Sigmoid activation function.

2.6.1.5 The Delta Rule

The Delta Rule is used in supervised learning with feed-forward networks with multiple output units and continuous activation functions and is also called *the least mean squares rule* (Stone, 1986). When the activation function of the output unit is the identity

function, the delta rule looks like this:

$$\triangle w_{ji} = \eta(t_j - x_j)x_i \tag{2.21}$$

where x_i represent activations, t represents a target, and η is a learning rate between 0.0 and 1.0 with i and j being the indices (Stone, 1986). To derive the delta rule for the more general case, i.e. when the activation function is a differentiable function, the gradient descent learning is used. Gradient descent learning means learning by moving in the direction that seems to be the best locally (Stone, 1986). For supervised neural network learning, the best direction to move in weight space is found by watching the change of global error according to the weight, for each weight.

A global error function is used in order to find the "best" direction for each pattern. First, we calculate the sum of the errors over all outputs with $\sum 1/2(t_j - x_j)^2$.

Then, we want to move in "weight space" in the direction opposite to the direction of the slope of the error function, which will move us towards a region with a lower error. The size of the move should be related to the descent of the slope, otherwise we might step over the error minimum we want to find. This could lead to oscillation around this minimum (ibid.).

The following describes the calculation process (ibid.):

1.) To find the slope, we take the partial derivative of the error with respect to the weight. But the only element in the sum of error terms that depends on the weight is the one for the output unit where that weight ends (j in what follows).

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial[\frac{1}{2}(t_j - x_j)^2]}{\partial w_{ji}} \tag{2.22}$$

2.) Using the chain rule, we can decompose this derivative into two that are easier to calculate:

$$(\partial[1/2\frac{(t_j - x_j)^2}{\partial x_j})(\frac{\partial x_j}{\partial w_{ji}}) \tag{2.23}$$

3.) The first derivative is easy to figure; it is

$$-(t_j - x_j) \tag{2.24}$$

4.) The second derivative can be decomposed using again the chain rule, if we remember that the activation of unit j is a function of the input to the unit, h_j, which is in turn a function of the weights into the unit.

$$\frac{\partial x_j}{\partial w_{ji}} = (\frac{\partial x_j}{\partial h_j})(\frac{\partial h_j}{\partial w_{ji}}) \tag{2.25}$$

5.) Since the activation of an output unit is the activation function g applied to the input h, the first derivative on the right-hand side of 4.) is just

$$g'(h_j) \tag{2.26}$$

that is, the derivative of whatever the activation function is at the value of the current input to unit j.

6.) The second derivative on the right-hand side of 4.) can be derived as follows:

$$\frac{\partial h_j}{\partial w_{ji}} = \frac{\partial(\sum x_k w_{jk})}{\partial w_{ji}} = x_i \tag{2.27}$$

since none of the other weights or input activations depend on w_{ji}.

7.) Putting all of the parts together, we get

$$\frac{\partial E}{\partial w_{ji}} = -(t_j - x_j)\, g'(h_j)\, x_i \tag{2.28}$$

8.) Remember that we want the weight change to be proportional to the negative of the derivative with respect to the weight. So, with a learning rate to control the step size for weight changes, we get the more general delta (least mean squares) learning rule

$$\triangle w_{ji} = \eta(t_j - x_j)\, g'(h_j)\, x_i \tag{2.29}$$

For the identity activation function the derivative is 1, so equation 2.21 is just a special case of equation 2.29 (ibid.).

2.6.1.6 Backpropagation

Backpropagation is a method suggested in 1974 by Paul Werbos (Werbos, 1974), but ultimately introduced in the mid 80's, among others by Rumelhart, Hinton and Williams (Rumelhart et al., 2002). The general idea is to have feedback from the output of a network as to how the weights and biases are updated. In order to do so, the output of the network for a backpropagation set is compared to the actual values of this set, where the difference is called error signal d of the output layer neuron. Since the output values of the internal neurons in the hidden layers are not known, it is impossible to directly calculate the error signal for those neurons. The way to train a multi-layered

network efficiently is by propagating the error signal δ back to all neurons which had an influence on the output layer neuron in question (figure 2.22). The weight coefficients are the same as in the direction of learning, the only parameter changed is the direction of the data flow. For each neuron reached, a new error signal δ_n is computed and the weight coefficient of this neuron might be modified (ibid.).

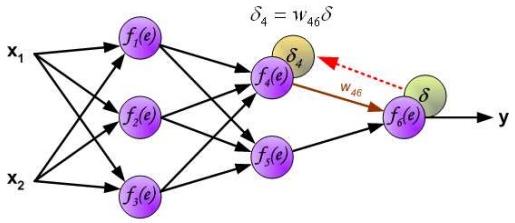


Figure 2.22: The first step of the backpropagation algorithm. Image taken from Bernacki (2005).

When all the neurons are updated, the next training step is conducted and the next backpropagation step follows.

2.6.1.7 Overfitting, early stopping and stopping criteria

Overfitting means the effect when a network trains on the training set and gets better as far as the output is concerned. At some point, this only relates to the training set and not to unseen examples, where the error actually increases with the training frequency. The network thus becomes very good in dealing with one set (the training set) while it becomes poorer dealing with any other set, namely the test set (Prechelt, 1998).

In order to avoid this effect, the training set is split into a rather large new training set and a smaller validation set. Then, early stopping is used to abort the training as soon as the error on a validation set increases (figure 2.23: vertical axis: validation set error, horizontal axis: time (in training epochs)). The network with the best performance (e.g. with the lowest error) on the validation set is then also used for actual testing with a separate set of data (ibid.).

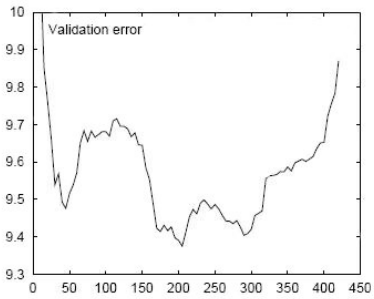


Figure 2.23: A real validation error curve.

Early stopping is often used in neural network training and often results in well generalizing networks while it does not do so in a mathematically well defined way. The time to actually stop the training early is hard to define, since the performance on a validation set can increase during a short instance and decrease afterwards to an even lower level. In order to avoid stopping too early, different stopping criteria can be used. Stopping criteria can be divided into several classes, according to how the criterion is calculated (ibid.):

The *first class* of stopping criteria is defined by stopping as soon as the loss of generalization exceeds a certain threshold. Sometimes however, if the training is still progressing very rapidly, it is better to avoid stopping. The cause is explained easily. When the training error still decreases quickly, loss of generalization has a higher chance to be "mended". Overfitting often only sets in when the error starts to decrease slowly. To write this down in a formal way, a training strip of length k is defined to be a sequence of k epochs numbered $n+1 : n+k$ with n divisible by k. E signifies the training error with $E_{tr}(t)$ being the error calculated on the training set up to the epoch before the current one. The measurement of the progress of training $P_k(t)$ (in thousands) after such a training strip is then

$$P_k(t) \; = \; 1000 * [\frac{\sum_{t'=t-k+1}^{t} E_{tr}(t')}{k * min_{t'=t-k+1}^{t} E_{tr}(t')} \; - \; 1] \tag{2.30}$$

That is, how much larger the average training error during the strip was than the minimum training error. Note that this measurement of the progress of training is high for unstable phases of training, meaning phases where error on the training set goes up rather than down. This is intended, because a lot of training algorithms sometimes oscillate by taking steps too large in weight space. The measurement of progress $P_k(t)$ will, however, approximate zero in the long run if the training is globally stable (ibid.).

Thereby we can define the *second class* of stopping criteria, which is using the quotient of generalization loss and progress. The *third class* of stopping criteria is completely different: it stops when the generalization error increases in a number of s successive strips. When the validation error increases more than once (in this case during s consecutive strips), such increases indicate the beginning of final overfitting and due to this fact, training is stopped.

Because neither of these criteria can guarantee abortion of training, we need a further rule, namely that training is stopped when the progress drops below a threshold or after a certain amount of epochs. Stopping criteria are all used the same way: When they stop the training after some t epochs, the result of training is the set of weights that showed the lowest validation error (ibid.).

2.7 Mean squared error

In order to be able to implement early stopping and also for training and validation output assessment, an error estimator has to be used. A standard error estimator is the so called mean squared error (short MSE), which is the difference between the real value and the output of the neural network. In mathematical terms, the error value e can be

described by:

$$e_i = Y_i - \hat{Y}_i \tag{2.31}$$

where Y_i is the real value and $\hat{Y}_i$ stands for the output of the neural network. This e_i is now first squared and then summed up over the number of occurences and then divided by the number of occurences. Thus, a mean of the squared error estimator is calculated:

$$MSE = \frac{1}{n}\sum_{i=1}^{n} e_i^2 \tag{2.32}$$

Using this error estimator, the output error of the neural network can be assessed for the training error and the validation error as well as for the test set output, which can in combination be used as a measure of quality of the training (Allen, 1971).

2.8 Linear correlation

Linear correlation indicates the relationship between two variables, especially the strength and the direction of this relationship. There exist several correlation coefficients for different situations. The best known is probably the Pearson product-moment correlation coefficient, which is obtained by the dividing the covariance of the two variables by the product of their standard deviations (Rodgers and Nicewander, 1988).

In mathematical terms, the correlation coefficient $\rho_{X,Y}$ between two random variables X and Y with expected values μ_X and μ_Y and the standard deviations σ_X and σ_Y is defined as:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E((X - \mu_X)(Y - \mu_Y)}{\sigma_X \sigma_Y} \tag{2.33}$$

where E is the expected value operator and *cov* stands for covariance. The correlation is defined only if both standard deviations are finite and both of them are nonzero. The correlation cannot exceed 1 in absolute value due to the corollary of the Cauchy-Schwarz inequality (please see Buskes and Van Rooij (2000) for further reading). The correlation will be 1 in case of an increasing linear relationship, and -1 in case of a decreasing linear relationship. It will become 0, if there is no linear dependence between the two variables. The closer the correlation value is to either 1 or -1, the higher is the degree of the linear relationship between the two variables.

In case there are n measurements of X and Y (written as x_i and y_i with $i = 1, 2, 3, ..., n$), then the Pearson product-moment correlation coefficient can be used as a measure of the linear relationship of the two variables X and Y. In this case, the Pearson correlation coefficient is written:

$$r_{xy} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{(n-1)s_x s_y} \tag{2.34}$$

where $\bar{x}$ and $\bar{y}$ are sample means of X and Y, s_x and s_y are the sample standard deviations

of X and Y and the sum is from $i = 1$ to n. This can be also written in the form:

$$r_{xy} = \frac{\sum x_i y_i - n\bar{x}\bar{y}}{(n-1)s_x s_y} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}} \tag{2.35}$$

The square of the sample correlation coefficient r_{xy}, R^2 is also known as the coefficient of determination and gives some information about the goodness of fit of a model and is thus used for assessing the quality of the linear relationship. It is important to state, however, that correlation does not imply causation, which means that correlations may provide valuable information about causal relationships among variables, but a high correlation does not necessarily provide evidence, that the change in one variable results in the change of the other variable as well (Rodgers and Nicewander, 1988).

2.8.1 Master templates

In order to correlate any values, there has to be some kind of template to correlate to (s_x and s_y in equation 2.35). These templates are patterns built up over several real life examples of values. The values are for example summed up and then divided by the number of patterns used. This way, a mean curve can be deducted of how for example a signal normally looks like (see figures 2.24 and 2.25 for real life examples from this project). Hence, new examples of the same or different patterns can be correlated to these master or mean templates as shown in figure 2.26.

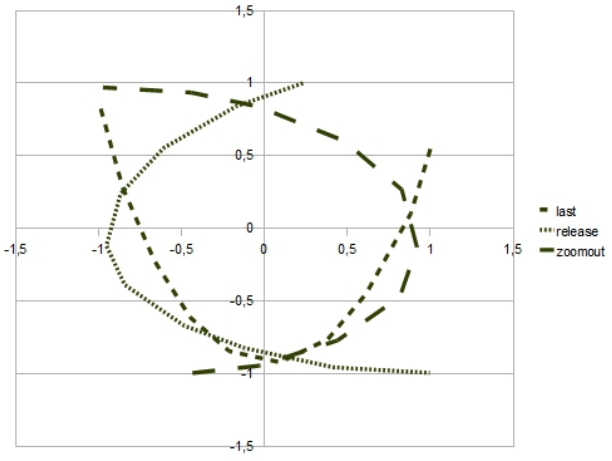


Figure 2.24: The master templates of the centre point movement for selected gestures.

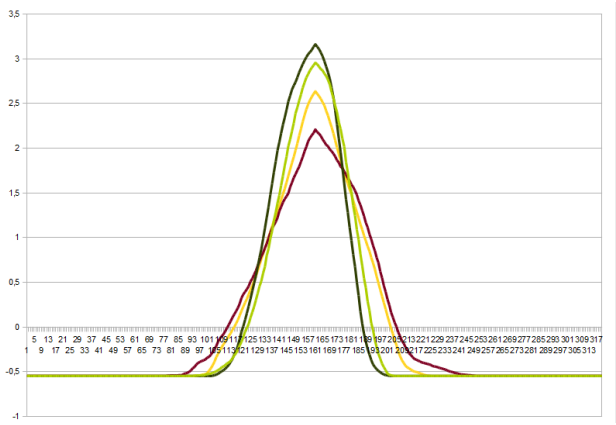


Figure 2.25: The master templates for the form vector (horizontal sum only) for selected gestures.

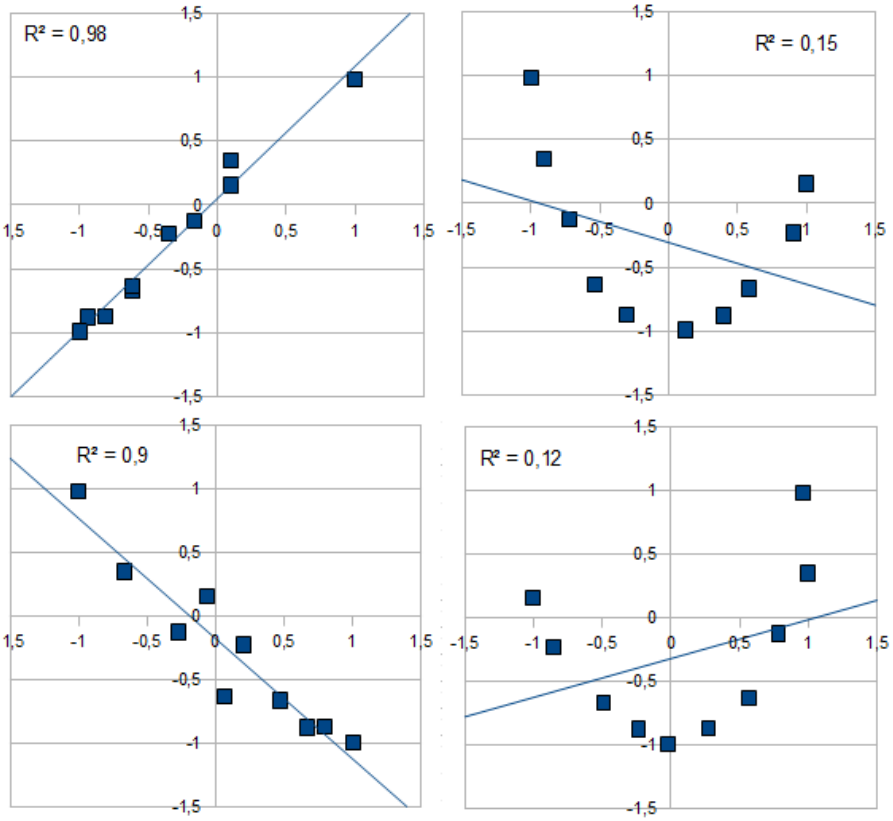


Figure 2.26: Correlations of a new movement vector to the single master template of a few movements.

2.9 Validation methods

The validation techniques used in prediction systems are essential for creating a working system. Without validation, the system with the best concept is prone to failure, as it would not be sufficiently tested for correctness. Both the integrity of the system in itself as well as the validity of the systems output have to be extensively tested, preferably in an automated way. This is especially true when using classification methods, as the automated decision making processes of the machine have to be thoroughly tested, so that as few as possible false positive predictions or recognitions may occur. Several validation methods are suggested in literature, only a few used should be mentioned here.

2.9.1 Cross-validation

Introduced by Geisser (1974), cross-validation is a method commonly used in machine learning in order to estimate error rates of prediction rules. For this method, the training set is split into one, new training set and a validation set, which is then used to estimate the error on the outcome of a prediction rule by testing the machine learning algorithm with the validation set and comparing the output with the actual values related to the single observation.

2.9.1.1 Leave-one-out cross-validation

If only a few training examples are available or new training examples are costly to get, the leave-one-out method may be applied. As the name suggests, one training example is extracted from the rest of the training set and stored as a test example. The rest of the training set is used for training of the used algorithm while the test sample is then used for assessing the quality of the prediction method. This method is then applied incrementally on the whole training set, so that every training example is used as a test example once with the remaining set used for training. So if for example a training set with a size of $n = 25$ examples is given, applying the leave-one-out method would lead to 24 separate training/testing combinations. The overall validation error is then calculated by dividing the number of wrongly estimated values by the number of total examples (Kohavi, 1995).

2.9.1.2 K-fold cross-validation

The idea is to split up the training set into some number of n equal parts, for example $k = 10$ (called 10-fold cross-validation in this case). From these parts, one is selected as a validation set, while the other $k - 1$ subsets are concatenated to a new training set. The cross-validation process is then applied k times with each of the subsets used once as a validation set. The results of these k cross-validations can then be averaged to have one single prediction. The main advantage of this method lies in the fact that all of the observations of the training set are actually used for training as well as for validation, which suggests a high possibility to have a well generalized machine learning algorithm for predicting new samples (ibid.).

3 Analysis

As already mentioned earlier, the idea is to create a gesture recognition module that can react to hand gestures in order to support the already existing eye tracking system implemented in the iCOMMIC. In the first part of this chapter, the analysis of the current situation and currently available systems is conducted. Deducing from the analysis, the basic requirements are built up and analyzed as well. Building up on the analysis of the requirements, the functional requirements are set up and are analyzed. As a last step, the final requirements are computed and the systems concept, design and implementation are built up.

3.1 Strategy of the analysis

There exist several systems that could be used to build up a gesture recognition module. By assessing the available ideas, the most suitable combination of methods should be found to implement a system as close to the requirements as possible. As a main source of information, expert interviews have been conducted to be able to assess the main requirements of the application. Furthermore, a detailed hardware analysis is done, deciding about the hardware aspects of the system. Additionally, a time frame analysis is conducted in order to assess the time given for each of the processing steps, further examining the possibilities of the project as far as (for example) programming language is concerned. With relation to the time frame analysis, available hand tracking methods are analyzed and decisions about which methods to use are made. Assessing the possibilities of the recognition/classification methods as well, the analysis is concluded. Deducting from the expert interviews and the complete analysis (hardware, time, hand tracking and recognition methods), the requirement analysis and finally the functional analysis are conducted leading to the final concept and thus, the design of the project.

3.2 Preliminary Analysis

3.2.1 Expert interviews

In the course of the build-up of this project, expert interviews were conducted concerning several aspects of gesture recognition. The idea was to interview experienced computer users - spending eight or more hours on the computer per day - about their thoughts about gesture recognition, their ideas for usable gestures and the ideas for usage of the gestures. Furthermore there was a second part, where similar questions about gesture recognition in combination with eye tracking were asked. The goal was to get an overview about the possible usage of the system, about some basic features needed at any cost

and if the idea of gestures as an input modality would be accepted by the interviewees (and thus the greater public) at all.

The questionaire (see Appendix A) was developed in cooperation with Mag.rer.nat. Sandra Trösterer, a specialist on work psychology at the FG-MMS.

3.2.1.1 Interviewees

The fields of interest of those interviewed range from work psychology to web development, software design and software development. All of the interviewees thought of their computer knowledge as advanced to highly advanced, so that the expert criteria was met. As far as education was concerned, all of the questioned experts finished their diploma in their related field being in their doctorate studies with one in a post doctorate year. All in all, five experts were interviewed.

3.2.1.2 Gesture device experience

Half of the interviewees had experience with these kind of tools, while half did not. Of the half of the interviewees, all of the group has experience with both single touch devices as well as with multi touch devices. Asked about the amount of time they use gesture based user interfaces per day, our interviewees answered to using the devices one hour per day on average.

3.2.1.3 Commands and related hand gestures

After a short introduction about the field of hand gesture recognition and the planned set up as mentioned in 3.2.2 to 3.2.4, the first questions about possible commands and the corresponding hand gestures were asked. The setup of the system suggested usage of one hand only, as the field of view of the camera prefered such a usage. All of the experts came up with similar ideas about which commands should be implemented, of course with slightly different gestures for each of the commands. Those commands were:

- *activate*:
  to activate a certain button or something similar as used with the mouse, the single click, pressing enter.
- *next*:
  to go to the next element in a list, to turn a page, to the right.
- *last*:
  to go to the prior element in a list, to go back one step, to the left.
- *up*:
  to turn up the volume, to zoom in, to scroll up.
- *down*:
  to turn down the volume, to zoom out, to scroll down.

- *mark/select/grab*:
  to mark or select an element on the screen, also: to grab an element for moving or removing.
- *unmark/deselect/release*:
  opposite gesture to mark/select/grab, to release an object.
- *stop*:
  to stop a certain action.

Further commands included *delete*, *rotate* and *copy* as well as some application related commands like *open inventory* in games, *start chat* in games and *begin spell check* in word processing.

Please see 3.3.7 for the commands and gestures derived from the interviews as well as from the additional analysis.

3.2.1.4 Usability

The next question was about the usability of the system and how well fitted the user would see a hand recognition module for every day use. The answers were unambiguous about this topic; all of the experts answered with either *very usable* or *rather usable* without failing to note, that they would need a certain amount of time to get used to the system.

Asked about whether the experts would use such a system themselves, all of the interviewees answered with *yes* and asked about the reason for this, the answers all pointed in the same direction, namely, that the system would allow a more natural, more intuitive control of the computer.

Finally, the question about whether it should be possible to modify the gestures by themselves, all of the experts answered with a *definite yes* and when prompted for a reason, said that it is comfortable to be able to set up their own gestures for the commands, just like (for example) setting up their own short cut keys on the keyboard.

3.2.1.5 Combination with eye tracking, usability

The second part of the questionnaire concerned the combination of the hand gesture recognition system with an already existing eye tracking system. Introducing the idea shortly, the experts were asked about whether they would see a benefit of combining those two modalities, where the answer was a uniform *definitely benefitting*. Asked about why they would think so, the interviewees answered, that the eye tracking could be used for navigation which they already discovered was not favoured by the gesture recognition system in and of itself. In order to decrease the need for the mouse, eye tracking could be used for navigational input while gestures could be used for modificational input and thus this combination would be very favourable.

3.2.1.6 Combined commands

Subsequently, the experts were asked, if they could imagine additional commands and thus gestures that could come along with the combination of gesture recognition and eye tracking. The outcome was similar in all the users as well, where they imagined the following additional commands:

- *zoom in*:
  as the gesture *up* described in 3.2.1.3, but with its own gesture and related to the position of the centre of attention as returned from the eye tracking.
- *zoom out*:
  same gesture as *zoom in*, but other direction (down).
- *delete*:
  deleting the element in the focus of attention.
- *move*:
  move an element first selected by *mark/grab/select*, looking at the destination, releasing the element by *unmark/deselect/release*.
- *close*:
  close a certain window or application.

3.2.1.7 Application examples

As the next question, the experts were asked to think about certain application examples and more general, the fields of application which they could imagine such a system being used in. Depending on the field of interest of the individual users, the answers naturally differed slightly, but again, there was a consensus reached with regards to several fields:

- Reading text:
  using *scrolling*, *zooming*, *next* and *last* to navigate while reading text.
- Image viewing:
  using *next*, *last*, *rotating* and *zooming* for browsing image catalogues.
- File browsing:
  using the combination of eye tracking and gesture recognition to browse files in (for example) Windows Explorer. Includes copying, moving and deleting of files.
- Internet browsing:
  using the navigational commands like *scrolling* and *zooming* to read internet pages, while using command *activate* to follow links or show images, *next* and *last* to go back and forth in the browsing history.

3.2.1.8 Projected acceptance of the system in public

The final question of part two of the questionnaire asked the interviewees about their feelings concerning the acceptance of the gesture recognition module in combination with eye tracking in widespread public usage. The answers here were similar, but not unanimous. However, the range of public acceptance was generally seen as *high* to *very high*, where most of the experts said, that it would definitively take some time to introduce the system to the public and that acceptance would only come about after a certain amount of time went by, seen as the inducing phase. Furthermore, the system would have to work flawlessly; otherwise acceptance would not rise at all.

3.2.1.9 Further modalities

The last part of this interview was about the future of this project and thus about further modalities to be included into the iCOMMIC. Various additional fields were mentioned:

- Emotions:
  Recognize facial expressions and further indicators for emotional state such as heart rate and skin resistance in order to understand the current emotional state of the user.
- Speech recognition:
  Including speech recognition software could make interaction even more intuitive and make the system a real challenger to the current mouse and keyboard combination.
- Electroencephalography (EEG) and the brain computer interface (BCI):
  Reading the brainwaves of the user could reveal to be a powerful tool to understand the user's ideas faster. BCI is already in a progressed state of development, it would however work against the touchless principle of the iCOMMIC.
- Projected keyboard:
  A keyboard that only appears if necessary for textual input.

Generally, speech recognition was mentioned as to be seen as the next modality to be implemented into the iCOMMIC, as this would already cover the main output of the human body, namely voice and gestures with the eye movement used as a navigational device. All of the experts suggested that the acceptance of the system would further rise by implementing another modality, mentioning that in the time of introduction, the system could be used as an addition to the current modalities of mouse and keyboard only.

3.2.2 Hardware analysis

The hardware analysis assesses the up to date methods as described in the theoretical concepts chapter (see Chapter 2) and deducts the necessary hardware requirements from there. Furthermore, some project specific decisions are made and substantiated by reasoning. As a final part, the decisions made are summarized in 3.2.2.5.

3.2.2.1 Touch based gesture recognition and (multi-)touch pads

In the past few years, more and more touch based gesture recognition modules have come on to the market such as touch screens for portable computers and multi-touch pads such as those present in the *iPhone* and *iPod touch*. These modules react on gestures executed with fingertips on the surface of the screen, where sensors included into the surface track the touch and react accordingly. Gestures used in this field include navigational and modificational commands such as browsing multiple page entries, selecting icons and entries, inputting text on a virtual keyboard as well as other similar actions.

3.2.2.2 Vision based hand gesture recognition

Several methods of video based gesture recognition have already been published. Several of these take advantage of wearable devices and appendages to the human body, such as data gloves or other markers fixed on the fingers for quick hand detection (Takahashi and Kishino, 1991) while others try to segment the image into hand and non-hand by simply assessing the colour image (Stark et al., 1995). Additionally, different methods of classification have been used, such as neural networks (Nolker and Ritter, 1998), hidden Markov models (Liu and Chen, 2003) as well as pure correlation (Stark et al., 1995). Some try to assess postures (*i.e.*, static gestures) (Assan and Grobel, 1998), while others try to recognize movements of the hand, namely sequence gestures (Rigoll et al., 1997).

The general idea though is mostly the same. Gestures posed in front of a camera should be detected and recognized. In only a few of the articles mentioned above however, is the system used for actual feedback to the computer in the form of commands issued to control the computer.

3.2.2.3 iCOMMIC multimodal interaction

iCOMMIC so far is a system that allows a touch free communication between human and computer. By using an infra red camera to get the position of the eyes on screen, the user can sit in front of the system without needing to touch either a keyboard, mouse or anything else for that matter. In keeping with this idea, the gesture recognition module should be touch free as well, eliminating the touch based gesture recognition modules from further consideration and adapting to use a vision based approach instead.

3.2.2.4 Camera positioning

Another main issue of the system is where to position the camera. Since the system should combine gesture recognition and eye tracking, the position of the eye tracking system has to also be considered.

The eye tracking module is fixed under the screen of the system, which leaves the camera used for hand tracking only a position on the top of the screen. Since the idea is to use colour for the tracking of the hand (see 3.2.4 for deduction), it seems favourable to position the camera so that it points downwards in order to avoid the head of the user obstructing the field of view of the camera so as to provide for easier tracking. Figure

3.1 shows an example of how the camera could be set up with the field of view marked in red. To set up the camera on one side of the screen might also be favourable, since there is no need to extract the second hand, so long as it does not appear in the image. Regardless, if a second hand appears in the field of view, it has to be extracted from the image.

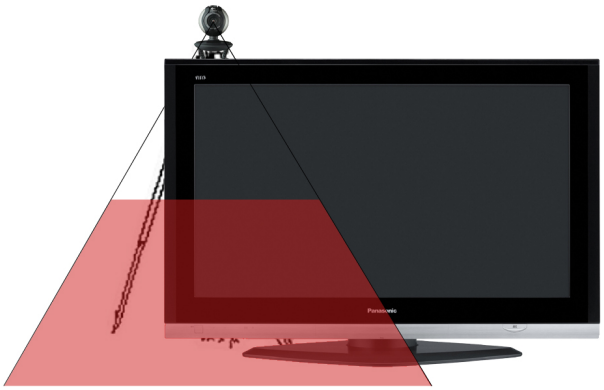


Figure 3.1: Positioning of the camera

3.2.2.5 Deduction of the hardware analysis

Deducting from the gesture recognition module analysis in 3.2.2, the hardware has to be some kind of camera. As connectivity is also an issue for *one-for-all* systems, the camera should be connectable to the PC via a simple interface such as the USB 2.0 interface. Furthermore, it would be practical to use low cost cameras, suggesting that simple webcams should be used. Since different webcams return a different quality of images, the cameras should be high-end webcams. To be able to change the basic settings of the camera, the market leader Logitech was chosen, as there exists a good interface for changing the camera's settings. Furthermore, a very important benefit of the Logitech high-end webcams is, that they include a *natural light* feature, returning a very good image illumination and thus colour representation, only gradually influenced by the surrounding lighting of the scene.

A very important decision that has to be made is about which method to use to extract the hand from the image. Since the main idea of this thesis is to implement a touch free system, hand tracking gloves will not be used. Furthermore, no additional sensors or coloured patches should be needed in order to make the system a *one-for-all* system. Thus, the skin colour segmentation method should be used for the realization of the system.

3.2.3 Time frame analysis

To meet the needs and standards of the iCOMMIC the gesture recognition method has to be a real time application, meaning that recognition has to respond to input very

quickly. As the frame rate for webcams can be up to 30 frames per second, that would leave a calculation time of as little as 33 milliseconds for image processing (step one), movement detection (step two) and recognition (step three). As we don't have to trick the eye though, high frame rates are not especially needed. To stick with the idea of having a real-time system, however, and in order to have a timely resolution of the hand gestures, the frame rate will be set to 15 frames per second, leaving the system a time frame of 66 milliseconds to process all three steps mentioned before.

3.2.3.1 Programming language

Additionally, the timing of the system is greatly influenced by the programming language chosen. Different languages have different time concepts. For example Java, which is a very convenient programming language, as there is no need for memory handling, brings a major timing setback with itself. The concept of the garbage collector, which collects and recycles unused memory used for variables, objects and containers, greatly influences the timing of the system. Without the programmer having significant influence on the timing, the garbage collector uses some processing time every 150 milliseconds to recycle the collected memory. Since the timing of this cannot be influenced greatly, Java is not a language for real time systems as is needed for this project.

Thus, another language has to be chosen. Since in C# the concept of the garbage collector exists as well and as C is very low level and has little library support for webcam connections and image processing, C++ is the language of choice. The object oriented approach (also an advantage to C) combined with the possibility to manage the memory by hand and thus not be dependent on the garbage collector makes this language ideal for this project as it satisfies all necessities.

3.2.4 Hand tracking and recognition method analysis

As already suggested in 3.2.2.5, the hand tracking should be implemented by extracting colour components from the video frames. The only question remaining is that of which method to use. As the conversion of images to different colour spaces is generally a time consuming process, the method suggested by Gomez et al. (2002) and shortly described in 2.4.2.1 might not be the best solution for this time critical pre-processing step. The RGB extraction rate as described in 2.4.2.1 does not look very promising either due to the high occurrence of false positive recognition. Hence, the third method suggested in 2.4.2.1 and by (Stark et al., 1995) - the extraction of the colour components from the YUV/LUV colour-space - seems to be the most promising and will be considered to be the method of choice.

3.2.4.1 Quality assessment of the hand tracking module

The hand tracking module is the combination of first the image processing for each frame, extracting the hand features and returning an image holding the shape of the hand, as well as the bounding box and the centre point for each frame. Secondly, the movement detection and thus the hand tracking have to be calculated as well.

Hand tracking has to be of high quality, meaning that tracking of the hand can never be lost as long as it is in the field of view of the camera. Furthermore, the calculations needed have to be done as quickly as possible, so that the frame rate achieved by the system can be seen as real time. Since these two steps have to be handled in two thirds of 66 milliseconds (as deducted from 3.2.3), the time frame given for the hand tracking module is about 44 milliseconds.

Automatic validation is very difficult here, so the quality has to be determined by hand, meaning assessing the extraction quality by inspection of each and every frame. In order to be able to assess the quality of the hand extraction, a video debug mode needs to be implemented showing the binary output image of the extraction algorithm if wanted. This way, it is possible to recalibrate the camera during runtime of the program by shifting camera settings using the camera's own lighting settings.

As high resolution images are not needed for this task, the resolution of the camera can be set to a low 320x240 pixel, speeding up the process by a factor of 4 as compared to the next highest resolution setting (640x320 pixel), with an image 4 times the size of the lower resolution to be processed.

3.2.4.2 Static gestures or gesture sequences?

The next question that has to be answered is which kind of gestures the system should be able to recognize, rather static postures as suggested by Assan and Grobel (1998) or sequences of movement as suggested in Rigoll et al. (1997).

Since the static postures demand a recognition phase for every single frame and as there are fewer possibilities for overall gestures, the idea would be to implement a gesture sequence classifier instead of the static gesture recognition. The benefits of either one of the methods are multifarious; however, moving the hand is a more natural form of communication then posing static gestures.

3.2.4.3 Recognition method analysis

For the recognition method, there exist various possibilities. Hidden Markov models as proposed by Liu and Chen (2003) could be one option, as could be neural networks (Nolker and Ritter, 1998). Additionally, a simple correlation in combination with pre-built master templates for each of the gestures as described in (Stark et al., 1995) might also be a method with a satisfying outcome.

In order to get satisfying results, a combination of two methods will be used. Neural networks will by used as a pattern recognition method. However, this method is not flawless, as previous experiences have shown, especially, since classification via a neural network returns probabilities, not only of which class the input could be of, but assigns a class for input which is itself classless. Thus, a simple correlation for pre-filtering will be used in order to decide if an incoming gesture is in fact a gesture at all or not. Also, a combination of the output of the neural network and the output of the correlation might be an interesting approach, as this approach will probably yield a more accurate classification than using just one of the methods.

3.2.4.4 Problems of gesture recognition

Three main sources of problems can be differentiated:

- *Speed of gestures:*
  A very important issue that might occur in gesture usage is that gestures are not always acted out in the same speed and that different gestures need different amounts of time to be conducted, which might interfere severely with the recognition process. In order to prevent problems, sequences have to be recalculated to a certain length, omitting surplus frames if a gesture takes longer than the predefined number of frames or adding blank frames to the end of a gesture if the gesture needs less frames for execution.

- *Bad lighting:*
  Another problem to be removed is bad lighting of the scene, which can interfere with the tracking of the hand as the colour of the hand could be altered significantly by bad lighting. As artificial light has a high share of red components, it will change the colour perception decisively, so white light has to be used. Examples of white light include that emitted by the sun, but also by fluorescent lamps and light emitting diodes (LED's). Also, hardware including the "natural light" feature can help significantly in improving this issue.

- *Background of the scene:*
  The background of the scene is vital, as background colours can be similar to the colour of skin. Hence, the background of the scene has to be a uni-colour board or desk with a high difference to colours with high red/green proportions. Black or white colours without any traces of colour are definitely preferential.

3.2.5 Discussion

Finishing the analysis, several important decisions have been made. First, the hardware decision was made, where the decision came to using an average (but not too simple) webcam. Secondly, the frame rate used for this project was set to be at 15 frames per second, leaving the system enough time to pre-process every frame, detect movement and if necessary classify a gesture.

Furthermore, the method of hand extraction was determined, bringing forth the decision of using the extraction of the hand by colour assessment in the YUV/LUV colour space. Choosing between static postures and gesture sequence analysis, the decision was made to use gesture sequences, as the form of gesturing by moving the hand seems more natural than posing fixed gestures.

With regards to the recognition method, the decision came to using neural networks in combination with correlations with master templates.

Furthermore, some decisions about the workspace for which this system is designed have been made, namely the surrounding lighting settings and the background of the scene filmed by the video camera.

This allows a short overview of the system. It will be a real-time gesture sequence analyzer using a USB webcam with a timely resolution of 15frames/second and an image resolution of 320x240pixel. The hand tracking will be done by colour property extraction and the gesture recognition will be conducted by a neural network in combination with a correlation of a new incoming gesture and master template gestures built up by using the data also used for the training of the neural network. The output of the module will then be fed back to the iCOMMIC, which is responsible for acting out the recognized commands.

The only thing missing up to now is a name for the project. Since it is going to be a gesture recognition module made in Germany, it will carry the name GRM.

3.3 Requirement analysis

Here, the requirements of the project are deducted by considering the main demands assessed in the expert interviews (3.2.1) and some own thoughts about how the system should work. The fields considered here are usability, feedback mechanisms, interaction methods, recording, security and configurability of the application with a strong focus on usability (to which also the feedback mechanisms in some way relate) and the interaction methods, so that the system will be as usable as possible while covering the main requirements (interaction methods).

3.3.1 Usability

Usability is one of, if not the most important factor in human computer interfaces. This is due to the fact that the usability of any system has major influences on the design of the system, especially with regards to the user interface of the device. The user interface (or graphical user interface) has to be easy to use, and is optimally intuitively comprehensible without the need for instructions. Additionally it is important for the system to have feedback mechanisms to guide the user securely through the application. Holzinger (2005)

More concretely speaking, the user interface will always give details about its current internal state to the user (compare 3.3.2) and thus will guide the user through execution of the application without the need for the extensive study of manuals. If the graphical user interface will be available, it will be set up in a way that unclickable items will be greyed out, the current mode the system is running in (see 3.3.3) will be shown at any time and furthermore, the system will return information about the quality of recognition by showing a green/orange/red bar, where green means high quality of recognition by the currently loaded correlation/neural network combination, orange means mean quality and red means low quality of recognition.

3.3.2 Feedback mechanisms

Feedback mechanisms are mechanisms, that either feed back information about the current state of the system to the system so that the system can handle program errors. On

the other hand, feedback can also be given to the user of the system in order to guide the user through the usage of the application. Both parts of the feedback mechanisms have to be implemented thoroughly in order to guarantee a safe and stable running of the application as well as safe and stable guidance of the user as already implied in 3.3.1.

In other words, it is necessary to implement an internal feedback mechanism for creating a fail safe application on the one hand, and on the other hand a well balanced info system for the user needs to be implemented.

3.3.2.1 Internal feedback mechanism

The internal feedback mechanism will be implemented in a way, that the calling methods always know whether the called function terminated correctly or with errors. This will be done either by an integer number returned by the called function holding a negative value for an error state and zero or a positive number for a correct state. Or, if the function returns a variable, this variable can be checked for correctness, so if an array or vector of values is not empty, a pointer is not null and so on. This way, segmentation violations can be avoided and with an additional exception handling implemented this should lead to a robust and stable system.

3.3.2.2 User information system

The user information system should be implemented in such a way, that the user always knows, what is going on in the application without giving too much information to avoid confusion. The user should be able to immediately see what state the program is in at any point of execution, namely the mode the system is currently running in, the state in which the system is running in (for example the paused state during the recording and recognition mode) and maybe the possibilities the user has to change the current state or mode. Please see 3.3.3 and 4.2.4.1 to 4.2.4.4 for further details about the different modes and states.

This user information system (UIS) will be implemented either in a dialogue box inside of the GUI (if available) or as textual output on the console.

3.3.3 Interaction methods

Generally, two different methods of user input exist. Interaction with the system can be seen as the first kind of user input. This input is fed to the system by executing hand gestures in front of the camera and is used for issuing commands to the test bed or operating system, but not to the system itself. This input could be generally be seen as either manipulative or navigational input, between which we will have to decide. As the iCOMMIC already uses a very capable system for navigational input, namely the eye tracking module, the gesture recognition module and thus this thesis focuses on manipulative input only with navigational input completely set aside.

Controlling the functionality of the system and thus issuing commands to the system itself is the second kind of interaction with the system. The control of the system is

managed by either pressing keys of the keyboard or using the mouse to control the graphical user interface (GUI) if available.

Both interaction methods have to be defined clearly and stated in some kind of user manual for easy introduction of the system to novice users. For commands issued by hand gestures, please see 3.2.1 and 3.3.7 for further details. As far as the control of the system itself is concerned, the following questions have to be considered:

1. *How should the control of the system work?*
   As the system should send recognized commands issued by executing hand gestures in front of the camera to the operating system or the test bed, the other modalities (mouse and keyboard) can be used for direct control of the system. Shortcut keys or mouse control can be used to switch modes, switch the display of the video window and enable or disable the recognition mode. Please see 3.3.3.1 for details about the shortcut keys.

2. *What are the different modes the system needs to run in and how can the modes be changed during runtime?*
   For the different modes, it is important to know, what the system is able to do. Five modes are needed to have a complete running system, including the three main modes recording mode, training mode and recognition mode. The fourth mode is a debug mode for program debugging only, which the user will not be able to start. The fifth mode is a so called autotest mode (4.2.4.5), which uses prerecorded sequence information (see 4.2.4.1) for testing the quality of the recognition system. During runtime, the user will be able to switch between modes by simple key commands issued on the keyboard or activated by the mouse using the GUI (if available). For a complete list of those keys, please see 3.3.3.1.

3. *What does the recording mode do?*
   First, as described in more detail in 3.3.6, the system has to be configurable in many ways. This means that the user should be able to introduce new and different gestures for existing commands. This necessitates a recording mode in order to save self recorded gestures in a file which can then be read in training mode.

4. *What does the training mode do?*
   For the training mode, the system has to either load an existing, already trained network from file or load a previously recorded training set. The network will be trained by the methods suggested in Chapter 4.2.2 and master templates for the correlation based approach described in chapter 4.2.3.4 will be created. Both the network and the master templates will then be held in storage or saved to a new or existing file for the recognition phase.

5. *What does the recognition mode do?*
   The recognition phase is the actual core of the system, as all classification is done in this module. Gestures which are acted out by the user are translated into the right format and finally fed to the recognition module consisting of the correlation

module and the neural network; once again, both are either loaded from file or trained in the training phase.

6. *What does the debug mode do?*
   The debug mode consists of all three modes mentioned before. The only difference of the debug mode is the amount of (debug) information written to the console about the inner state of the program. This will generally only be used by programmers working on the GRM.

3.3.3.1 Keyboard control and hot keys

The following keys are defined for controlling the system:

key	function	key	function
+	pause/unpause	ESC	quit
ctrl-d	toggle debug video mode	ctrl-e	toggle overlay video mode
ctrl-f	toggle debug line detection mode		

Table 3.1: Keyboard controls and hot keys

These commands can only be issued while the video window is active.

3.3.4 Data recording and Logging

Recording data can mean different things. First, there is a recording mode for the gestures, where new gestures can be recorded for later training of the system. See 4.2.4.1 for more information. Furthermore, statistics about the usage of the system can be recorded, meaning storage in (for example) XML (Extensive markup language) files. This information could possibly include timestamps of gestures issued and the kind of gesture recognized. Please see 4.2.6 for further details.

3.3.5 Security

Security of the system is also a must. Both stability and security have to be mentioned here. The system must never fail disgracefully, so in other words, it must not terminate with a segmentation violation or something similar without feeding the error back to the user. To assure this, C++ exceptions should be introduced in order to always know, where the program fails and under which condition. Additionally, no person from the outside should be able to manipulate the system in order to compromise data on the host computer. Most of those issues are solved by the operating system itself. However, it has to be made sure, that the socket connection used to connect to the iCOMMIC system (as described in 4.1.2) can not be tampered with. However, as this is an outgoing connection, the risk is minimal.

3.3.6 Configurability

The system has to be as configurable as possible, meaning that the user can change and modify many of the settings. As this system is fairly straight-forward, there are not many things to be configured about the system itself, thus configurability means something a bit different.

The idea is to allow the users to record their own movements for the existing commands (or even add completely new commands) as the expert interviews (see 3.2.1) demanded. This means, that it has to be possible to go to record mode, select the file you want to save to and start recording. For detailed information about this, please see 4.2.4.1.

Additionally, the user has to be able to select the foldernames for the logging file(s) and change them during runtime or at least before starting a new test. This can either be done by inputting the desired foldername in the text mode on the console or by clicking and creating a new file or selecting an existing file in a GUI (if available).

3.3.7 Gestures

The predefined gestures to be implemented were deducted from the commands defined by the expert interviews as well as from the suggested gestures by the experts (see 3.2.1 for the starting point of this deduction). As far as commands are concerned, the focus was set on the most relevant commands for manipulation, as the gesture recognition module should be primarily used for manipulation (as described in 2.1.1.3 as well as 3.3.3).

The commands are:

1. *activate:*
   used for activating (="clicking") buttons and similar events.

2. *grab:*
   for picking up symbols, images, files.

3. *release:*
   for releasing previously picked up items.

4. *next/last:*
   the commands for going to the next item (next) or the previous item (last), also usable as right/left.

5. *rotate left/right:*
   generally used for rotating pictures.

6. *zoom in/out:*
   used to either zoom in or out of pictures, text or similar events. Can also be used as a replacement for the up and down keys for example for scrolling.

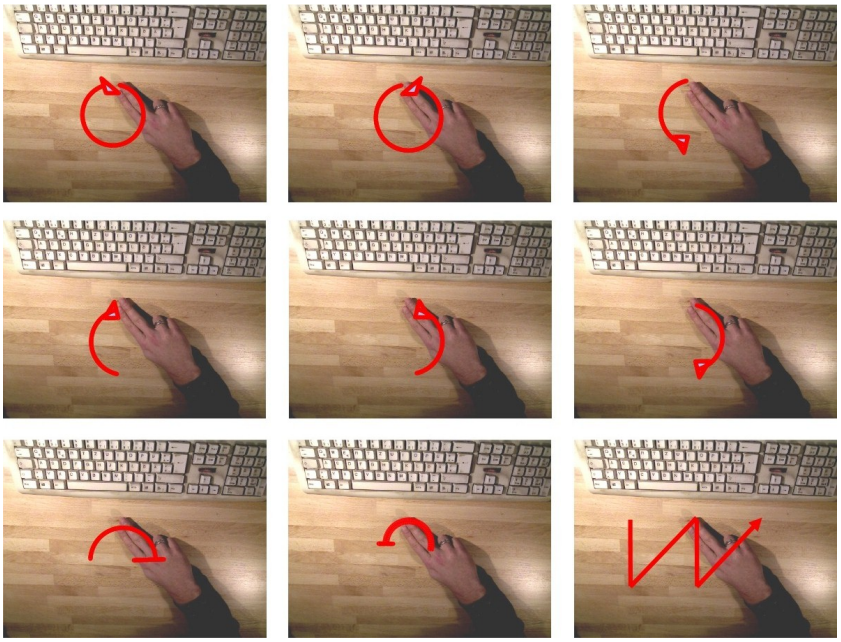


Figure 3.2: The selected gestures for the deducted commands. From left to right, top to bottom: rotate right, rotate left, grab, release, zoom in, zoom out, next, last, activate.

3.4 Additional requirements

In order to not implement already existing libraries all over again and thus not to reinvent the wheel, several libraries are used. This way, well tested and secure libraries are introduced, minimizing the possibility of faulty programming in important modules of the software. User libraries (as well as professional libraries) are available for a lot of fields in programming, as many programmers code a project for themselves and publish the results on the internet. In order to find out, which kind of libraries are needed a quick assessment is conducted.

3.4.1 Available libraries

3.4.1.1 Image Processing

First, the field of image processing has to be considered. As this will be the core part of the hand tracking of the system, it has to be implemented in a clean way with optimized functions as far as speed, processor usage and memory usage are concerned. Additionally, the images coming from the webcam should already be in the right format in order to spare costly transformation calculations. In other words, a library is needed that can both capture frames from a camera (or video) as well as perform the needed image processing operations in a time effective way. Additionally, the library should include an extensive

documentation in order to be able to read into the usage of the library quickly and to understand how the principles described in Section 2.4 are implemented in this software. Several libraries like that exist on the World Wide Web, from the CImg Library over to Mathtools (as known from Matlab programming) to the Open Computer Vision library (also known as OpenCV).

Considering time efficiency and memory usage of the libraries mentioned in 3.4.1.1, the difference of the libraries is first and foremost difficult to assess before actual usage and furthermore, they are all optimized according to those requirements, hence, they should be not so far apart. As the documentation of the OpenCV library extends the rest of the libraries and the usage appeared to be fairly straightforward, this library was chosen for later usage.

3.4.1.2 Machine learning

The second large and important part of the software that is available already in libraries is the machine learning part. Since neural networks are going to be used, libraries holding only this aspect are sufficient for the system. Considerations about the different types of neural networks are influencing the choice of library, as well as speed and memory usage. Again, the speed and memory usage are difficult to assess before actually using the library, so the main focus lies on the functionality and documentation of the software. Here, several libraries are available for usage, from SNNL (the Simple Neural Network Library), the ANNL (Artificial Neural Network Library) over to professional (and often costly) libraries.

From the libraries mentioned above, the integration of the SNNL seems the most simple and the documentation is rather good and readable. Additionally, the structure, which can be fed to the SNNL is easy to build up from the way the data is stored on drive. These attributes helped in making the final decision to utilize this library.

3.4.1.3 Data recording and logging system

Recording and logging important data is another very important aspect of this project, so a well structured way of saving data in any form has to be found. Using the XML (Extensible Markup Language) file format seems to be a viable solution to this problem, hence, a library which is both easy to use and integrate has to be selected.

Only basic XML features are needed to implement data recording, thus the tinyXML library is the library of choice here.

3.4.2 Discussion

For the calculation intensive parts of the GRM project, external libraries are used as they are generally well tested and easy to integrate into new systems. These libraries will take over the main calculation parts concerning image processing (OpenCV), machine learning (SNNL) and logging/recording data (tinyXML) and thus will guarantee correct and time-efficient calculations in those key fields of the project.

4 GRM - Gesture recognition module

4.1 System concept and design

Deducting from the analysis in Chapter 3, the system has to consist of several self contained parts. The main parts indicated by the analysis are the extraction of the hand from the live video, a system that understands to form complete movement sequences and finally the recognition module recognizing the executed gesture. Furthermore, there has to be a central structure that makes the parts work together smoothly.

Additional functionality required is a tool to pre-process new incoming data, so that the recognition can always work with the same form of data. In order to conveniently store important information for both single frames and sequences as well as previously calculated pre-processing information, a few additional components have to be introduced. Furthermore, for debugging information and to see whether the timing as described in 3.2.3 can be achieved, an additional part responsible for timing has to be introduced.

For the output, an additional section will be needed, so that output to both files (i.e. for storage) and the debugging console can be done easily and uniformly. Additionally, the usage of three external open source libraries is suggested in order to facilitate the implementation of image processing, XML output as well as machine learning as described in 3.4.1.

Summing up these requirements, 11 main classes, 2 additional structs for storage as well as the connection to the three external libraries have to be implemented. A more detailed overview of the classes and their functionality will follow in the upcoming Chapters.

4.1.1 Classes and functionality

As described in Section 4.1, the GRM - Gesture Recognition Module is built up by 11 parts or in this case classes all in all, combined with a few structs for storage and three external libraries for image processing, XML support and neural networking functionality.

The core components are the *GRMImagePreprocessor*, responsible for extracting the hand from the video image and the *GRMClassifier*, responsible for teaching the system using the implemented learning algorithms as well as the online classification of sequence images created by the *GRMSequencer*. This *GRMSequencer* uses the previously preprocessed images returned from the *GRMImagePreprocessor* to build up sequences of frames where movement has occurred.

Furthermore, the class *GRMDataPreprocessor* is an essential part of the system, as it pre-processes both the sequence images received by the *GRMSequencer* as well as the stored data for learning. In order to be able to store information of single frames, the class *GRMImage* was created, holding information about width, height, the horizontal

sum, the vertical sum, the bounding boxes and the frame id of each frame. Derived from this class, there are two more specialized classes, the *GRMFrame* for storing information about single frames as received from the *GRMImagePreprocessor* (including the original IplImage, the binary image and an optional line image), as well as the *GRMSequenceImage*, storing a vector of bounding box points for each of the frames included in the sequence and a classifier name for the relevant sequence.

The connection between the single parts is implemented in the *GRM* class as described in 4.1.1.6. See 4.1.1.1 to 4.1.1.6 for the details of the classes. Additionally, a full class diagram including the connections between the different parts is displayed in figure 4.1 on the next page.

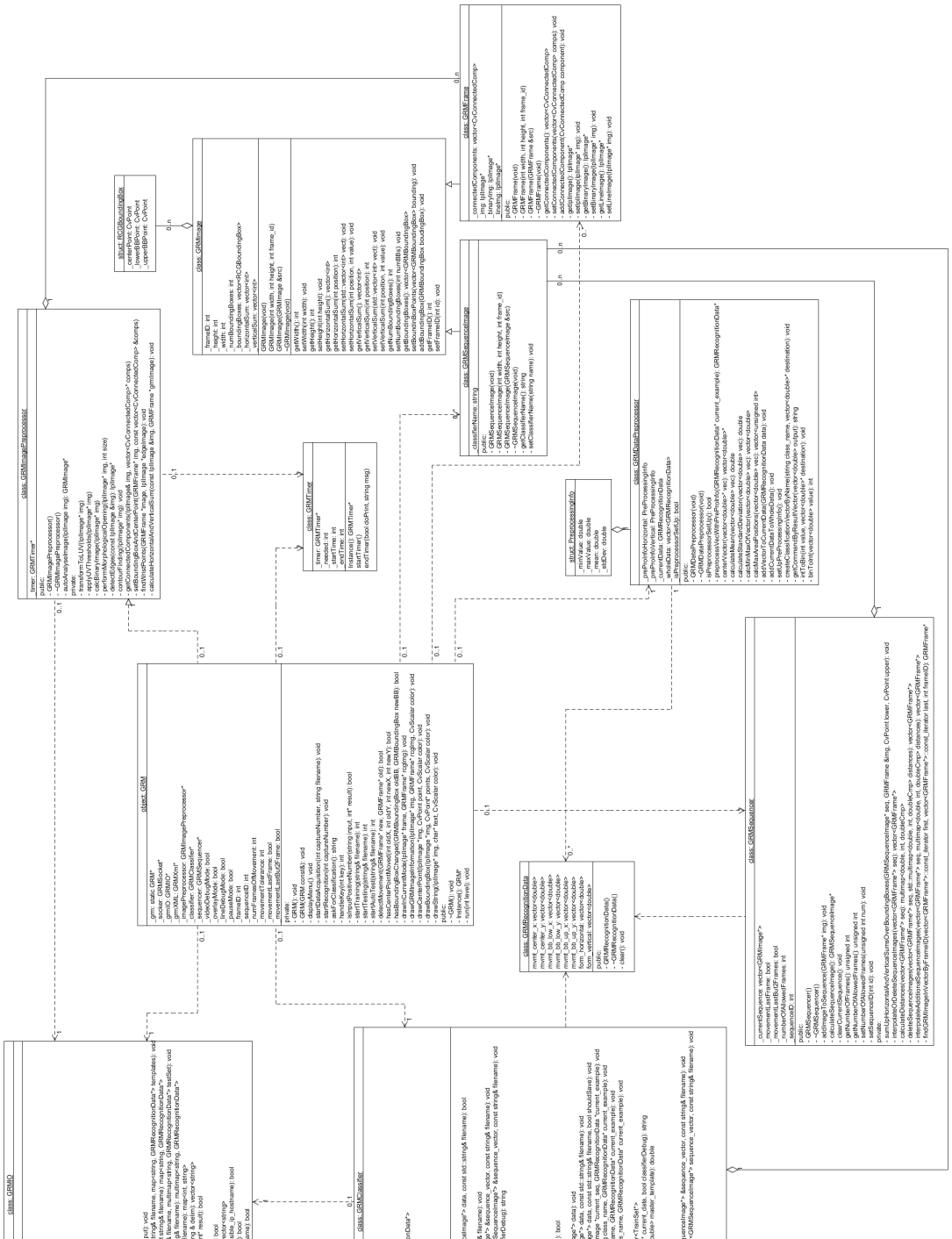


Figure 4.1: The class diagram

4.1.1.1 GRMImage, GRMFrame, GRMSequenceImage

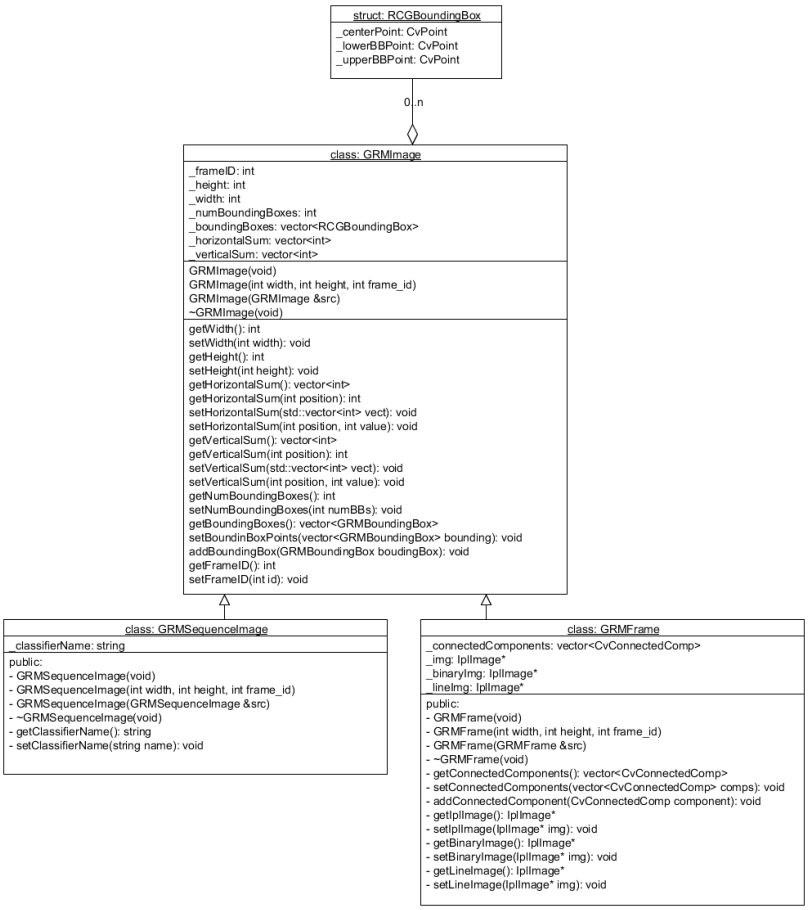


Figure 4.2: The class GRMImage and its derived classes GRMFrame and GRMSequenceImage

These three classes are container classes created for storage of vital information about either single frames or sequences of frames. The base class *GRMImage* includes information about the width and height of an image, as well as the the struct *RCGBoundingBox* for storage of the centre point and the corner points of the bounding box surrounding the extracted hand. Additionally, it holds two vectors holding both a horizontal and a vertical sum. Being a container class, the *GRMImage* also provides methods for manipulation of its content. This class is never actually instantiated, instead its two derived classes are used.

Derived from this base class, two additional classes are used in this project, namely the *GRMFrame* and the *GRMSequenceImage*. Consisting of the same basic functionality as the *GRMImage*, both are more specialized for their purposes. The *GRMFrame*

is a container for single frames which will later be used to build up sequence images. It includes storage facilities for IplImages as received from the OpenCV library for the original image, the calculated binary image and the line image. Furthermore, the *GRMFrame* includes a vector of connected components, so information about all connected areas in the binary image of the current frame. Please see the description of the *GRMImagePreprocessor* in 4.1.1.2 for how these images are created and how the connected components are assessed.

The *GRMSequenceImage* on the other hand includes additional information about the classification of the current sequence, so a string containing the associated class if available (which is the case only in recording mode). This class is put together in the *GRMSequencer* using *GRMFrames* where movement has been detected. Please see the description of the *GRMSequencer* in 4.1.1.3 for details about how these sequence images are created.

4.1.1.2 GRMImagePreprocessor

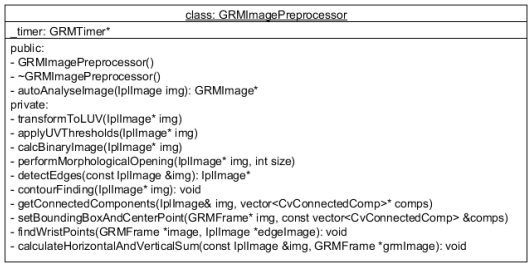


Figure 4.3: The class GRMImagePreprocessor

Fired up by the *GRM* class by calling the *autoAnalyseImage* method, the *GRMImagePreprocessor* receives a single frame from the capture device which is an *IplImage* pointer as returned by the OpenCV library. This frame is then copied to a new IplImage for further processing while the original is stored inside of a *GRMFrame* object. Using the copied frame, the image is transformed from the basic RGB colour space to the LUV colour space by using the OpenCV command *cvCvtColor* inside the method *transformToLUV*.

```
cvCvtColor(img, img, CV_BGR2Luv);
```

A hysteresis threshold is then applied to the U and the V channel of the thereby transformed frame inside of the *applyUVFilter* method. In order to do so, the frame is split up into it's channels by the OpenCV command *cvCvtPixToPlane*, which takes the original image as well as three new image pointers as arguments and writes the single channels into the three new image pointers (also from the type IplImage). As already described in 2.4.2.1, the L channel is not used, thus it is set to zero immediately. The U and the V channel of the image however are manipulated by applying two filters on each of

the channels by the OpenCV command *cvThreshold*, which writes the thereby resulting image into a new IplImage.

```
// apply threshold to U
cvThreshold(uImg, upper, 106, 255, CV_THRESH_BINARY);
cvThreshold(uImg, lower, 118, 255, CV_THRESH_BINARY_INV);
IplImage* destU = cvCreateImage(cvGetSize(img), img->depth, 1);
cvAnd(upper, lower, destU);

// apply threshold to V
cvThreshold(vImg, upper, 125, 255, CV_THRESH_BINARY);
cvThreshold(vImg, lower, 135, 255, CV_THRESH_BINARY_INV);
IplImage* destV = cvCreateImage(cvGetSize(img), img->depth, 1);
cvAnd(upper, lower, destV);
```

The values for the thresholds were assessed by previous manual tryouts for the method in the software Adobe Photoshop.

After rebuilding the image to its previous state of a three channel image by the OpenCV command *cvCvtPlaneToPix*, an binary image containing all the pixel set in either the U or the V channel is generated by using the OpenCV command *cvXor* inside of the *calcBinaryImage* method. Returning a new IplImage pointer containing an image with a single black and white channel only, this new image is then stored in the same *GRMFrame* object as the original IplImage before.

The binary image is then morphologically processed in the method *performMorphologicalOpening* by applying a morphological opening (as described in 2.4.4.3) in order to diminish noise in the image. This is done by performing the OpenCV commands *cvErode* and *cvDilate* with a previously generated structuring element as can be seen in the following code section:

```
IplConvKernel* kernel = cvCreateStructuringElementEx(element_size,
    element_size, (element_size-1)/2, (element_size-1)/2,
    CV_SHAPE_CROSS, 0);
cvErode(img, img, kernel, 1);
cvDilate(img, img, kernel, 2);
cvReleaseStructuringElement(&kernel);
```

The now "clean" image is then handed over to the next pre-processing step, namely the method *getConnectedComponents*, which runs through all pixel, checking if a pixel is set (recognized by the value 0xFF, so all binary digits set to 1) and then performing the OpenCV command *cvFloodFill* on this pixel. This command marks all N4 connected pixel of the current pixel with a given index. N4 neighborhood means the adjacent pixel to the left, right, up and down of the current pixel. If the thereby received area of active

pixel has a bigger height and width then 10 pixel, the current component is saved into a vector of *cvConnectedComp*'s.

```
for(int count = 0; count < width*height; count++)
{

    uchar pixval = data[count];
    // if this pixel has the value of 0xff
    // this component has not yet been recorded
    if(pixval == 0xff)
    {

        // fill this component in with the next lowest value
        CvConnectedComp current;
        cvFloodFill(img, cvPoint(count%width, count/width),
             cvScalar(currentCompVal+1), cvScalar(0),
             cvScalar(0), &current, 4);
        if((current.rect.width >= 10)
             && (current.rect.height >= 10))
           comps->push_back(current);
    }

}
```

The next step is to extract the bounding box points from the hand. In order to do so, the following assumption is made: The largest area of "active" pixel is seen as to be the area of interest, thus the hand. The values of this area (as received from the *cvConnectedComp*) is then used to set the values in the same *GRMFrame* object as the two other images before.

Finally, the horizontal and vertical sums are calculated in the method *calculateHorizontalAndVerticalSums* and also stored in the *GRMFrame* object, which is then returned to the calling *GRM* class.

4.1.1.3 GRMSequencer

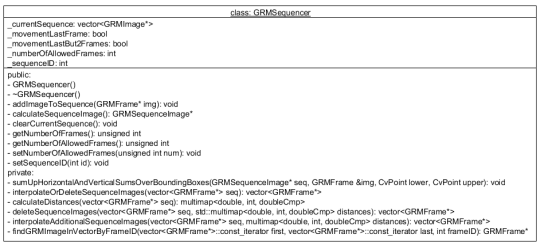


Figure 4.4: The class GRMSequencer

The *GRMSequencer* is responsible for creating clearly defined sequences out of a series of frames with movement detected. Since the duration of a gesture can be different between users as well as the different gestures also differ as far as their number of frames are concerned, it is necessary to define an upper limit of frames per gestures, otherwise it is very difficult to train any classification method to deliver satisfying results. Taking this information, sequences are limited to 10 frames per sequence, which means that an average gesture is considered to be about 700ms long.

The *GRMSequencer* is created by the *GRM* and whenever a frame considered to hold motion in relation to one of the two preceding frames, the *GRM* adds the frame to the sequencer. This incoming frame is stored in a vector called *_currentSequence*. When a sequence stops (so no more motion is detected), the *GRM* calls the method *calculateSequenceImage*, where the sequence is preprocessed. Please see Chapter 4.2.3.1 for details of internal calculations.

When the sequence is created and returned to the *GRM*, the *_currentSequence* vector is emptied by the *GRM* by calling the method *clearCurrentSequence* in order to be able to process the next incoming sequence.

4.1.1.4 GRMDataPreprocessor

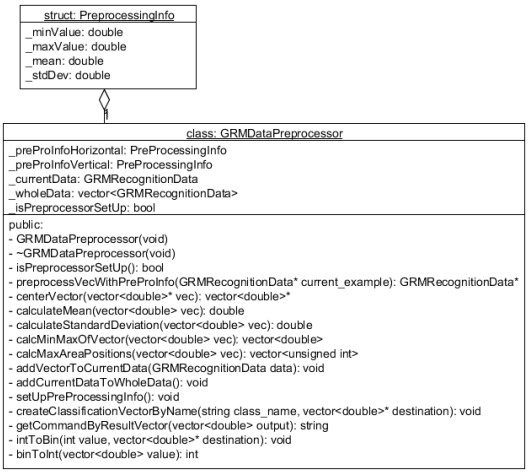


Figure 4.5: The class GRMDataPreprocessor

For the *GRMDataPreprocessor*, the name is program. Data loaded either from disk or coming from the GRMSequencer get pre-processed by previously calculated pre-processing information. The *PreprocessingInfo* struct is used to hold the necessary information for pre-processing both the movement vectors as well as the form vectors (= horizontal and vertical sums) such as the minimum and maximum values of a vector and the mean and the standard deviation for the same vector.

For pre-processing of sequences to classify (coming from the *GRMSequencer*), it is

necessary to pre-process that data with the same information used for pre-processing the recorded data, otherwise the system will not be able to classify any of the incoming gestures/commands.

First, the movement data is pre-processed by offset removal as described in 2.5.2. To do so, the minimum values of both x and y values for each of the three movement vectors (centre point movement vector, bounding box lower point and bounding box upper point movement vectors) calculated using the method *calcMinMaxOfVector* using the following algorithm (after setting the min to the maximum value of a double and max to 0):

```
for(unsigned int count = 0; count < vec.size(); count++)
{
    if(min > vec.at(count))
        min = (double)vec.at(count);
    if(max < vec.at(count))
        max = (double)vec.at(count);
}
```

The minimum value of x and y values are then subtracted from the movement vectors, leaving values independent from the location of movement in the field of view of the camera. Then, the values created this way are normalized to the interval [-1, 1], as described in 2.5.4, so that they can be used as input for the neural network.

Finally, the form vectors (e.g. horizontal and vertical sum vectors) are also sent to the *calcMinMaxOfVector* method. In this processing step, the positions of all the zero points of the form vectors are calculated by:

```
// if the first entry of the vector is not 0.0, add position
if(vec.at(0) != 0.0)
    positions.push_back(0);
for(unsigned int count = 1; count < vec.size(); count++)
{
    // if current != 0 & before == 0, push back a minimum
    if((vec.at(count) != 0.0) && (vec.at(count-1) == 0.0))
        positions.push_back(count);
    // if before != 0 & current == 0, push back a maximum
    if((vec.at(count) == 0.0) && (vec.at(count-1) != 0.0))
        positions.push_back(count-1);
    // since after a min there always has to be a max
    // at pos x%2 = 0, there always has to be a minimum
    // and at position x%2+1 = 0, there has to be a maximum
```

```
}
// if the last entry of the vector is not 0.0, add position
if((vec.at(vec.size()-1) != 0.0) && (positions.size()%2 != 0))

    positions.push_back(vec.size()-1);
```

This returns a vector holding positions of the beginning of an incline followed by the end of an area. Now, the distances between those min/max tuples are calculated and only the one with the biggest distance is seen to be the relevant area holding the movement of the hand. This area is now centered onto the 320 (for the horizontal sum) or 240 (for the vertical sum) entries vector, so that the highest peak is set to the position

```
int new_centre_position = vec.size()/2;
```

Copying the values from the old positions to the new positions relative to whether the new center is more to the left or right of the old centre, we receive a vector holding form information centered onto the middle position of the vector which can be easily compared to other form vectors pre-processed in the same way.

4.1.1.5 GRMClassifier

Figure 4.6: The class GRMClassifier

The classifier is started up by the *GRM* when the recognition mode, the training mode or the autotest mode are started. The classifier is set up by the method *clearAndConfigureClassifier*, which deletes all previous instances of classification methods, clears the

contained vectors holding the training sets, validations sets and test sets and finally starts up new instances of the network and the data pre-processor.

For each of the modes, there are different methods used. Training of the network occurs in the two *trainNetwork* methods, where one is specialized for the training mode and the other one handles the k-fold cross-validation of the autotest mode. The build-up of the correlation data occurs in the method *setUpClassifier*, which receives a vector of sequence images and a foldername where to store the relevant information. Before any training can occur, templates, train sets and/or correlation test sets have to be loaded or created from the incoming data (if no files are available for loading) and the data pre-processor has to be fed with the same data via the *setUpPreprocessor* method. Then, depending on the mode, either a network is trained (as described later in this chapter), the k-fold cross validation is started for either network or correlation validation or a loaded network and/or correlation set is made available for classification of new incoming data in recognition mode.

Recognition is made available by the two public methods recall, where one is set up for network usage, while the other one takes care of correlation recognition only.

Training is started by calling the *startTraining* method, which starts both network training and the calculation of the correlation master templates. The output of both are then stored to file for later usage. Training of the network is done for a maximum of 500 epochs with early stopping implemented. To have early stopping available, the train set of the network has to be split into two parts, one large one with about 90% of the test cases and a random selection of validation examples, which make up about 10% of the complete training set. In case the training error drops below 0.0005, the validation error drops below 0.005 or the validation error doesn't change for more than 0.00001 for three circles of validation (so 30 epochs maximum), training is stopped and the last settings of the network are used either directly for classification (if in recognition mode) or the information is stored to a file for later loading (in training mode).

4.1.1.6 GRM

Figure 4.7: The class GRM main class

The GRM is the core part of the system presented. It combines all the other parts, and thus controls the socket for iCOMMIC connection, the image pre-processor, the sequencer, the classifier, the input/output module and is additionally responsible for camera handling and the viewing of the video stream. It starts up a console, that displays a menu of available actions for the user. These actions are:

- *connect/disconnect to/from the iCOMMIC:*
  Takes care of the socket connection to the iCOMMIC if needed. Should be called first if connection to the iCOMMIC is wanted. This mode includes a preset connection (available by pressing the button "d" when suggested) which connects to the iCOMMIC on the localhost (127.0.0.1) with the iCOMMICs default port 1111. Alternatively, other IP-addresses, hostnames and port numbers can be entered.

- *recognition:*
  Starts up the recognition mode which asks for a foldername from where to load the classification algorithms if the classifier is not set up yet.

- *data acquisition:*
  Starts up the data acquisition mode, which initially prompts for a foldername to where it should save the recorded data.

If the folder and the *sequence-information.grm* file are already available, it will continue the saving after the last sequence recorded previously. This mode starts up in paused mode, so that no unwanted gestures are recorded.

- *training:*
  This mode as well asks for a foldername first in order to know which previously recorded data should be used for training. This way, it very easy to specify several profiles by just changing between recording folders. In training, the data is loaded from disk and depending if network and/or master templates are already available, only the missing ones are calculated.

- *autotest:*
  Fires up the autotest mode in which there are three available options: the k-fold cross-validation of the correlation, the leave-one-out cross-validation for correlation and the k-fold cross validation method for the neural network. Also here, the user is prompted to specify a foldername from where to load the data.

Image acquisition and recognition

For both the image acquisition and the recognition mode, additional functionality is needed from the *GRM*, namely the motion detection between frames, as it handles the camera and the image pre-processor and additionally holds all necessary information about the previous and current frames. Whenever a frame is identified to represent a motion to the frame before or the frame before that one, the current frame is added to the sequencer by calling the function *addImageToSequence*. Whenever there is no more movement for at least two frames, the sequencer is run with the input stored inside of itself, processing the sequence and returning a *GRMSequenceImage* as described in 4.1.1.1. This image is then sent to either the GRMIO responsible for saving the information to disk or to the classifier for classification. Then, the used sequence is deleted and the process can start from the beginning.

4.1.2 Integration of the GRM into the iCOMMIC

The GRM and the iCOMMIC communicate via a TCP/IP connection established by a socket as already shortly described in 4.1.1.6. The iCOMMIC needs to be running on a machine reachable via networking, preferably on the same machine as where the GRM is running. As iCOMMIC listens on port 1111 for incoming connections, the GRM connects by default to the *localhost* (127.0.0.1) on port 1111. Once connected, the iCOMMIC accepts incoming messages from the GRM with the following encoding:

$$command : timestamp$$

This means, that the iCOMMIC waits for a string holding the command bundled with a number that is interpreted as a timestamp in milliseconds, so a string of numbers. If either of the values is not of the format the iCOMMIC expects, an error is printed on iCOMMIC (= server) side and the connection is dropped.

A correct example of such a command/timestamp tuple would be *activate* : 1234 which would lead to the iCOMMIC simulating a *pressKey* action followed by a *releaseKey* action of the keyboard button previously defined in the iCOMMIC setup. The timestamp is for logging affairs only.

The iCOMMIC setup is predefined in the *systemProperties.xml* file which can be found in the *configuration* folder of the iCOMMIC_gesture.

4.2 System details

In this chapter, details about the internal processes are explained more explicitly.

4.2.1 External configuration

Configuration of the system is loaded from the programs sub folder *config*, from the file *config.cfg*. In this file, all known commands are specified. Additional commands can be inserted here, for the connection to the iCOMMIC it is necessary to additionally specify the new commands in the iCOMMIC_gesture's sub folder *configuration* inside of the file *systemProperties.xml*.

Additional external configuration of the system is not necessary and thus not available.

4.2.2 Neural network configuration

4.2.2.1 Input units

As already described in 4.1.1.3, sequences always consist of 10 frames and hold information about three movements, namely the movement of the centre-, the lower bounding box- and the upper bounding box point, meaning 30 points of movement are available. With x and y values for each of these movements, 60 input units are necessary for the network.

4.2.2.2 Output units

Deciding about the output encoding of the network, it is necessary to assess the number of gestures that should be allowed. For the moment, at least 9 gestures are needed (as described in 3.3.7), which makes it difficult to just use one output unit, as an output unit produces only one number in the range of 0 to 1. This would make it very difficult to decide between gestures. A better idea is to encode the output by using binary numbers, so to have a combination of binary numbers to represent a gesture. As we have 9 gestures to encode, at least four bits are necessary to encode the gestures with binary codes. This also leaves some room for five additional gestures if needed. Gestures thus can be represented as 0000 up to 1111. This would suggest 4 output units necessary.

4.2.2.3 Hidden units

As there exist no calculation methods to fix the number of hidden units, educated guesses have to be made. There are several rules of thumb which influence the number of hidden units to be selected. One rule of thumb was introduced by Blum (1992), which says that the number of hidden units should lie somewhere between the number of input units and the number of output units. Generally, the number of hidden units influences the performance of the network to a great extent. Too many hidden units can make the network prone to overfitting, while a too low number of hidden units can make the network to be prone to underfitting and thus both way may lead to bad generalization results.

As these are very vague rules, the ideal number of hidden units has to be assessed by testing networks with varying numbers of hidden units and the network setup with the lowest generalization error (on the test set) should be selected.

Extensive testing has shown that a network with one hidden layer with 20 hidden units produces the lowest test set error, so this is the number of hidden units used.

4.2.2.4 Further network settings

Additional settings have to be selected for the network in order to perform in a satisfying way, namely the learning parameter and the momentum term parameter. These values also significantly influence quality of training and thus have to be tested thoroughly as well in order to have the optimal settings for the neural network used. After testing, the following settings were established:

- The learning parameter, defining the stepping distance of one epoch to the next is set to 0.11.
- The momentum term parameter defining the persistence of a weight change for several adjustment cycles has allowed values between 0.5 to 0.9 and is set to 0.7 after testing.

4.2.3 Calculations

This section gives an overview of how which kind of calculations were conducted, starting with the sequence compression calculations, over to the pre-processing of the same information for later usage in the training mode over to the calculation of the master templates, the correlation of new gestures to those master templates and the neural network output.

4.2.3.1 Sequences

Sequences are built up by comparing each frame grabbed by the camera to the one before. If the bounding box or the centre point has moved from one frame to the next, the frame is seen as part of a sequence and thus added to a vector of images. Movement is defined by the movement of at least one of the points by at least 5 pixel from one frame to the

other. When the next frame is processed, it is again checked for movement. In case this frame contains no movement, but the frame before contains movement, it is still added to the sequence vector in order to fill gaps in the movement which sometimes occur. Then, the next frame is assessed. If movement occurred, the frame is again added to the sequence vector. If no movement occurred, the frame is still added to the sequence image due to the same argument as before. Only if three frames in a row do not contain any movement, the frame is not added and the resulting sequence vector is analyzed.

For sequence analysis, the it is checked, how many frames were added to the current sequence. Three possibilities arise here:

1. If the sequence contains more than the standard ten frames, the distances between the single centre points of each of the frames in the sequence is calculated. This is conducted by calculating the Euclidean distance between centre point of frame f_i to the centre point of frame f_{i+1} (with i from $1 \dots n-1$ and n being the maximum number of allowed frames per sequence, in this case 10) with the formula

$$dist_i = \sqrt{(x_{i+1} - x_i) + (y_{i+1} - y_i)}$$

   keeping only those ten frames with the nine biggest distances between one to the next, omitting the others.

2. If the sequence contains less than ten frames, empty frames are added to the sequence until the sequence is filled up. By definition, the bounding box (as well as the centre point) of the last frame is used as the bounding box (and centre point respectively) for the new frames, as it facilitates the next steps of further processing.

3. If the sequence contains exactly ten frames, nothing is done in this step.

The next step of sequence calculations is the summing up of the horizontal and the vertical sums of the single frames as described in 2.4.5.1. This way, the horizontal and vertical sums of the resulting sequence image holds the information about the whole sequence in two vectors, one with the size $1 * width$ and one with the size $1 * height$ of the image.

After this processing step, the sequence image thus consists of three vectors of ten points each for the centre points and the lower and upper bounding box points, with each one of the points holding two values for x and y. Additionally, the sequence image consists of two vectors holding the horizontal and the vertical sums of the frames with the dimensions $1 * width$ and $1 * height$ respectively.

This is also the format in which data is stored on disk when recording gestures in the data acquisition mode.

4.2.3.2 Pre-processing of data

Pre-processing of the data is maybe the most important step in enabling recognition at all. If the data are not pre-processed thoroughly, even the most powerful recognition

algorithms may fail in classifying anything. Thus, the methods used have to be well considered.

As far as the bounding box and centre points are concerned, a way is needed to allow for all the movement to start at the same initial points in both axis (x and y) as described in 2.5.2. In order to do so, the minima of each of the x and the y of all vectors (lower bounding box points, upper bounding box points and the centre points) have to be subtracted from each of the x and y values using the formulas $xn_i = x_i - \min(x_n)$ and $yn_i = y_i - \min(y_n)$. This way, an example movement A with the centre points

$$PA_1 = (110, 200), PA_2 = (120, 200), PA_3 = (140, 215)$$

and another example vector B with

$$PB_1 = (202, 110), PB_2 = (222, 110), PB_3 = (242, 125)$$

would both return a movement vector of

$$P_1 = (0, 0), P_2 = (20, 0), P_3 = (40, 15)$$

which makes classification independent from the place where the gesture was posed. See figure 4.8 for an example containing the centre movement vector of example sequences with at two different locations with the third one near the 0 point of the axis being the pre-processed movement vector of both examples.

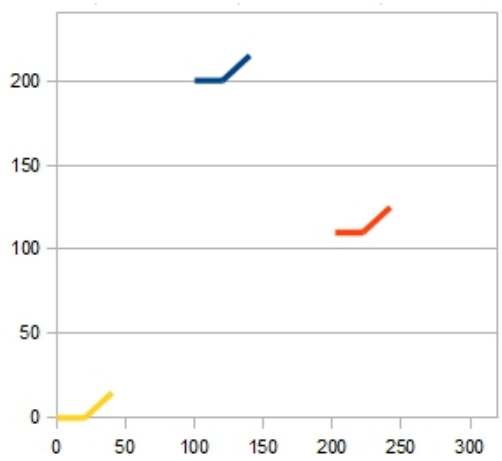


Figure 4.8: Example pre-processed movement vector (offset removed)

Furthermore, the data are preprocessed by bringing them to an interval [-1, 1], so that zero values always become -1 while maximum values become 1 respectively, with all values in between possible. See figure 4.9 for an example using the same values as in the example above.

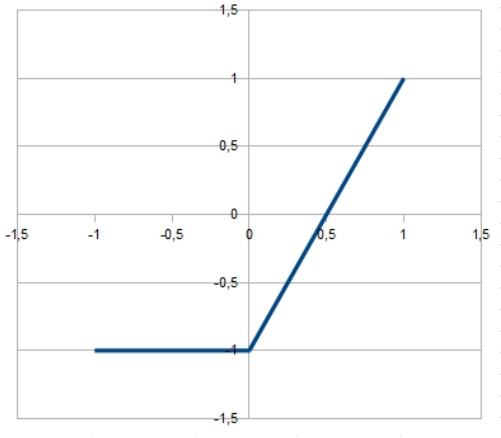


Figure 4.9: Example pre-processed movement vector (normalized to interval [-1, 1])

A completely different method is used to pre-process the horizontal and the vertical sums in order to make the position of execution of the gesture spatially irrelevant, as described in chapter 4.1.1.4. The idea is to find the highest peak of both the horizontal and the vertical sum, centering this peak on the centre of the sum vector (at the positions $\frac{width}{2}$ and $\frac{height}{2}$ respectively). This way, any performed gesture appears to be acted out in the centre of the image, if it was acted out in the lower left quadrant of the image or anywhere else. Thus, it becomes absolutely irrelevant, where inside of the boundaries of the field of view of the camera a gesture was performed.

Furthermore, the values of the sums are pre-processed as described in 2.5.4. Previously recorded sequence information is used to build up the population mean μ and the standard deviation σ as described by the equations 2.14 and 2.15, which are then used to pre-process incoming raw values of new samples to get the corresponding *z-scores* as described by equation 2.16.

4.2.3.3 Master templates

The master templates are generated by using the previously pre-processed data. For all available examples of a single gesture/command (activate, grab, last, next, release, scroll down, scroll up, zoom in, zoom out), the values for all available data (the two bounding box points, the centre point and the horizontal and vertical sums) are summed up and afterwards divided by the number of occurrences of each of the gestures/commands. In this way, master templates for each of the gestures/commands are built up which can then be used for the calculation of the correlation between two gestures. Figure 4.10 shows the master templates of the centre point movement. The gestures/commands demonstrated here are: last (dashed line), zoom out (wide dashed) and release (ultra fine dashed). Clearly visible are the significant differences between the single movement templates, suggesting a promising output for the calculation of correlations.

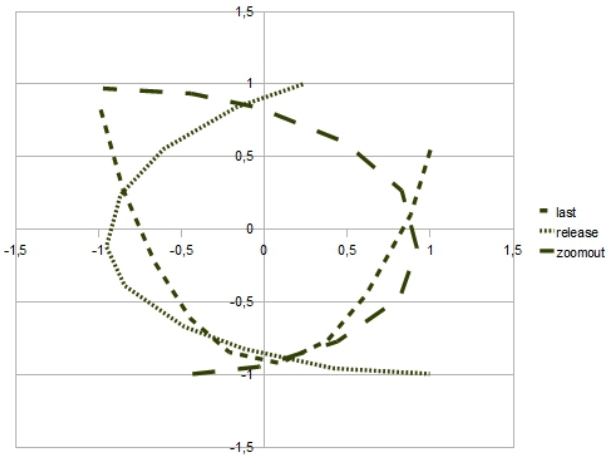


Figure 4.10: The master templates of the centre point movement for selected gestures

Figure 4.11 shows the master templates for the form vector of some gestures/commands with the left 320 values showing the horizontal-sum master templates while the right 240 values show the vertical sum master templates. As can be seen already, the differences are not as significant as hoped for, suggesting that this approach of the form vectors does not contain enough significant information for a real usage of this part.

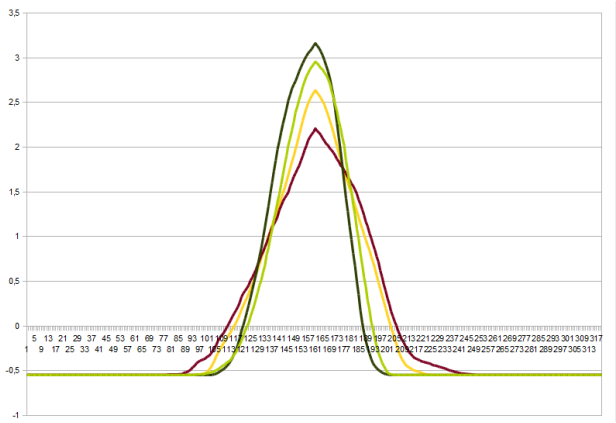


Figure 4.11: The master templates for the form vector (horizontal sum only) for selected gestures

4.2.3.4 Correlation

The correlation between each master template and a new gesture vector is calculated as described in Section 2.8 by formula 2.35. Each of the relevant vectors are correlated separately, meaning the x values of the centre point, the y values of the centre points, the x and y values of both the lower and the upper bounding box points as well as the

form vectors.

As negative correlations mean different gestures as well, only positive r values are then squared to receive the term R^2 with negative r values being omitted and further correlation for this the current gesture and the current master template stopped.

Testing has shown that the R^2 values returned by this calculation lie in the range of 0.90 and 1 in case of a positive match, while the results stay at a low 0.0 to 0.55 for negative matches. This is why a threshold has been introduced classifying a gesture only if the R^2 value is higher than 0.90 for each and every one of the separate correlations, while it returns an empty result for any lower R^2 value. Figure 4.12 shows the correlations of the x values of the center movement of new gestures/commands “grab”, “last”, “zoom in” and “rotate left” with the master templates of the gesture/command “grab”. Easy to see is the high positive correlation between the master template of “grab” and the new incoming gesture of the same type with a R^2 of 0.98. On the lower left, a high negative correlation is shown; as mentioned before, results like this are immediately omitted from further consideration.

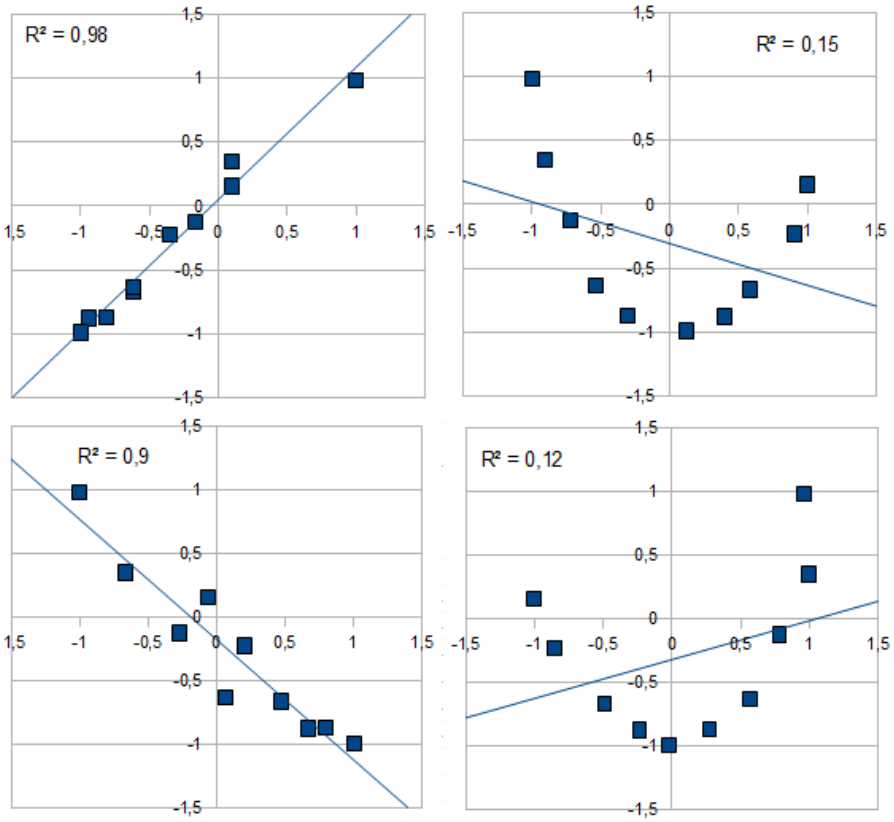


Figure 4.12: Correlations of a new movement vector to the single master template of each movement

4.2.4 Program modes

Different modes of program execution are available. This chapter shortly describes their functionality, output and the outputs interpretation.

4.2.4.1 Recording mode

The recording mode also known as data acquisition mode enables the user of the program to specify his or her own gestures for the previously defined commands as described in 3.3.7. The idea is to make available an easy to use recording mode where the user can define at start up where the new gestures should be saved. This way, several different files of examples can be created, for example for storing the individual gestures of different users of the systems.

The system starts up in a paused state until the pause button is pressed so that the user can position the hand to the initial position of the gesture. When the user presses the pause button (as described in 3.3.3.1), the system unpauses and starts recording one single sequence of frames, defined by the start of the movement to the end of movement. Depending on the length of the gesture issued, the program then assesses the gesture and either deletes excessive frames or adds empty frames to get to the standard gesture length of 10 frames as explained in 4.2.3.1. After this, the system immediately changes back to the paused state and directly requests the user to classify the current gesture. Here, all available commands are listed (loaded from the file *..\config\commands.cfg*), as well as a return entry, in case the quality of the gesture doesn't satisfy the needs of the user. In order to allow the user to assess the gesture just posed, the last frame of the sequence is displayed in the video window. If the user is satisfied with the gesture issued and hence classifies the gesture, the sequence information is directly written to the folder specified at start-up of the recording mode, into the file *sequence_ info.grm*.

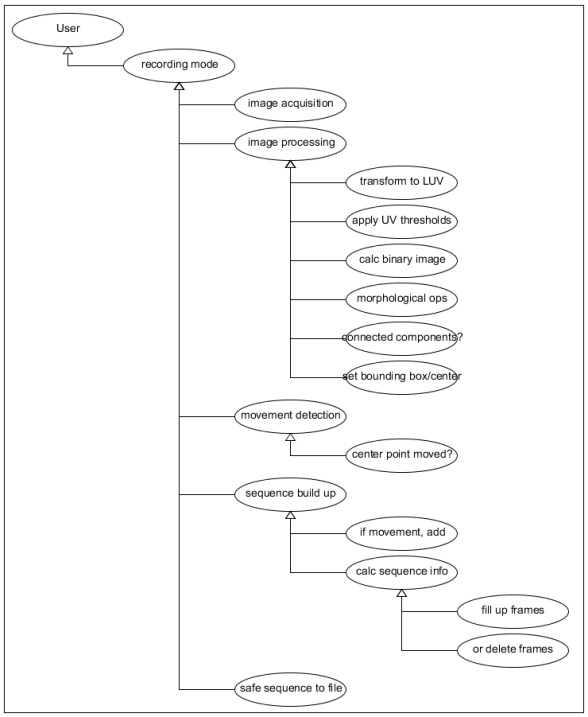


Figure 4.13: Data acquisition mode

4.2.4.2 Training mode

The training mode is the counterpart to the recognition part. Here, files containing sequence information as created in the data acquisition phase as well as network layouts and master templates can be loaded by the user at start up of the training mode. First, the user is asked to specify the name of the folder to use. Then, the user is asked if he or she wants to update the master templates and the neural network files if existing. If they are not existing, training is started automatically. This is particularly useful, for example if additional gestures have been recorded in the same file. Several different situations have to be considered here:

1. If no folder with the specified name exists, an error message is returned to the user.

2. If a the file *sequence_info.grm* in a folder with the given name exists but no other file, the sequence information contained in this file is pre-processed (as described in 4.2.3.2) and with this information master templates are calculated (as described in 4.2.3.3) and stored to a file within the given folder under the name *templates.grm*. Finally, the neural network is trained using the information from the sequence file. The obtained network layout is then saved to a file with the name *neural_network.grm*.

3. If the sequence information file and only a master template file exists inside of the given folder, the user is asked if he wants to update the master templates file. In case the user answers with "yes", the system updates the master templates using the sequence information, loads the new master templates into memory and saves the new master templates to file. Then, the neural network is trained with the data from the sequence information, stored in memory for later usage and saved to a file with the name *neural_ network.grm*.

4. If a sequence info file and a neural network file exist, the master templates are calculated, stored in memory and saved to file with the name *templates.grm*. For the neural network file, the user is asked if he would like to update the neural network (and thus start new network training and saving the results in the existing file), otherwise the neural network information is loaded into memory from file.

5. If only the master template file and the neural network files exist, the information is loaded into memory for later usage, but no information is saved to file as well as no user input is demanded if updating is wanted (as no sequence info is available to do so).

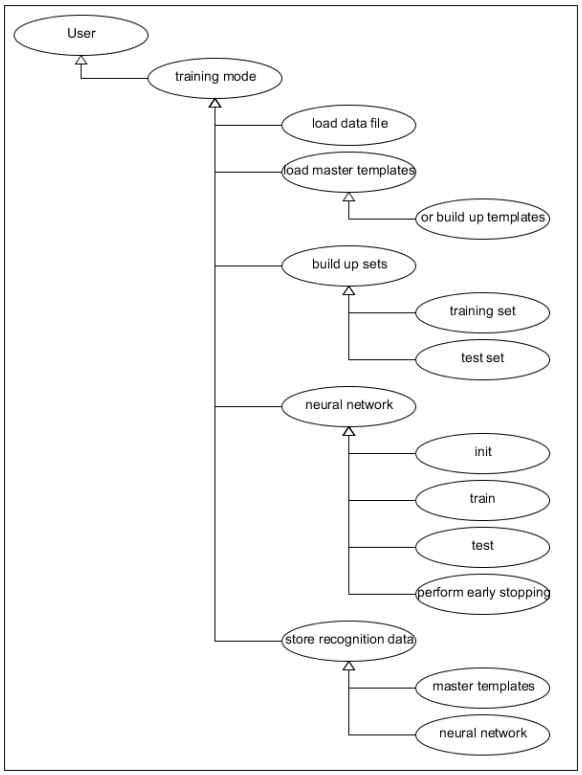


Figure 4.14: Training mode work flow

4.2.4.3 Recognition mode

The recognition mode is the core part of the system, the mode that this system is built for. Internally, it works similar to the recording mode, as it starts up the image acquisition and processing modules. If the classifier has not been set up yet, the system inquires the user which folder to use. Classification information is then loaded into memory for later usage. Movement is detected, sequences are built up and assessed for length with excessive frames deleted or additional frames added. However, the recognition mode does not save any information to sequence information files, but calls the recognition module, handing over the information previously recorded and pre-processed as described in 4.2.3.2. The recognition mode then calculates the neural network output of the incoming new gesture. If the recognition module returns a positive result, this information is then put out to screen and handed over to the iCOMMIC via the socket connection if available.

The recognition mode is started up in a paused state just as the recording mode and is activated by pressing the pause key (as specified in 3.3.3.1).

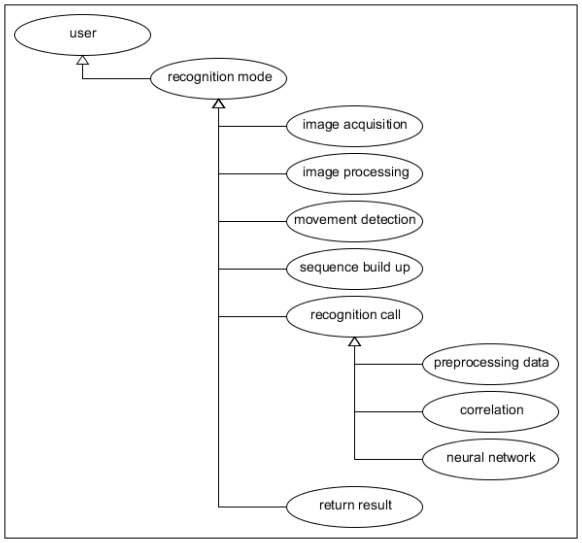


Figure 4.15: Recognition mode work flow

4.2.4.4 Debug mode (optional)

As the system presented is a prototype only, the debug mode is switched on at all times, so no separate debugging mode was included into this project as premediated.

4.2.4.5 Autotest mode

In this mode, the system uses either k-fold cross-validation (for both the neural network and the correlation) or the leave-one-out method (for correlation only) as described in 2.9.1 to assess the system's quality. To do so, the sequence information is loaded from

the file *sequence_info.grm* in order to use the data stored in it as the complete test set. Furthermore, the master templates are loaded from the file *templates.grm*. The calculations are done as described in 4.2.3.4 for the correlation and 4.2.2 for the neural network.

The output produced here is both written to the console as well as it is stored in a file in the same folder as the example set named after the method used, so *validation_output-k_fold_valid.txt* and *validation_output-network.txt* for the k-fold cross-validation and *validation_output-leave_one_out.txt* for the leave-one-out method. 4.2.5.4 shows examples of the output of the k-fold cross-validation for both methods.

4.2.5 Output

For each of the modes, different output is created. This includes video output, output send to the iCOMMIC via TCP as well as debugging and validation output. This chapter will specify the output given by each of the system's modes and explain the meaning and interpretation methods for each of the modes output.

4.2.5.1 Recording mode output

Both video output and textual output is generated in this mode. The video output initially shows the normal image with a superposed green bounding box around the area recognized as the hand and the corresponding center point of the bounding box. Using the hotkey commands as described in 3.3.3.1, the video output may be altered to the video debug mode, showing the binary image with a superposed black bounding box. Furthermore, the output can be set to a different video debug mode, the overlay mode, where the red channel is set to the binary image, turning the rest of the image green with the areas recognized as being hand colour in a brownish shade.

The textual output of the data acquisition mode is very important for all the other modes, as movements are recorded and the corresponding information is stored as sequence information in single entries in the output file *sequence_info.grm* as already described in 4.2.4.1. The file is built up in a basic XML style. See the following section for an example.

```
<?xml version="1.0" ?>
<SequenceImages>
  <sequence>
    <id>0</id>
    <classifier>grab</classifier>
    <num_bbs>10</num_bbs>
    <center_points_x>
    180 175 163 160 154 154 156 166 170 178
    </center_points_x>
    <center_points_y>
    145 144 146 149 161 166 175 183 185 185
    </center_points_y>
    <bb_points_lower_x>
    150 143 130 126 119 119 122 131 135 145
    </bb_points_lower_x>
    <bb_points_lower_y>
    106 105 110 114 129 135 145 153 154 153
    </bb_points_lower_y>
    <bb_points_upper_x>
    211 207 197 194 189 189 191 201 205 211
    </bb_points_upper_x>
    <bb_points_upper_y>
    184 183 183 185 193 197 205 214 216 217
    </bb_points_upper_y>
    <horizontal>
    0 0 0 0 0 0 ... 0 0 0 0 0 0
    </horizontal>
    <vertical>
    0 0 0 0 0 0 ... 0 0 0 0 0 0
    </vertical>
  </sequence>
<SequenceImages>
```

Each of the sequence entries holds an id value, an according classifier name (equals a command) and 10 values for each the centre x, the centre y, the lower bounding box x, the lower bounding box y, the upper bounding box x and the upper bounding box y. Furthermore, it holds 320 values for the horizontal sum as well as 240 values for the vertical sum.

4.2.5.2 Recognition mode output

The video output of this mode is quite the same as in the data acquisition mode with all the possibilities of using the hotkey commands as described in chapter 3.3.3.1.

The more relevant output of the system in recognition mode is composed of two separate entries, one for the gesture recognized and one holding the timestamp of the begin-

ning of the movement of the current gesture. These two values are then send to the iCOMMIC via a TCP connection (as described in 4.1.2) in the format *command* : *timestamp*. The iCOMMIC then handles the incoming input according to the predefined key mappings for each of the commands.

4.2.5.3 Training mode output

In the training mode, the data previously recorded in the data acquisition mode is processed as described in 4.2.3. The output is the correlation file *templates.grm* which holds the master templates for each of the gestures/commands on the one side and the neural network file *neural_ network.grm* on the other hand.

For the master template file, each template for a gesture/command is written in a single line with the name of the command as the first entry. The next 6x10 values are for the x and y movement of the center point, the x and y movement of the lower left bounding box point and the x and y movement of the upper right bounding box point respectively. The next 320 values hold the horizontal sum, while the next 240 values hold the vertical sum. Each of the subsets is marked by an ending semicolon, while the numbers themselves are separated by commas.

The *neural_ network.grm* file holds the internal configuration of the neural network training, meaning the layer information, the activation functions and the unit weights determined via training.

In both recognition mode and autotest mode, these two file are parsed (e.g. loaded) and then used for the neural network or correlation calculations needed for the corresponding mode.

4.2.5.4 Autotest mode output

In this mode no video out is present as the system only produces validational output. Two different methods can be used here, firstly the neural network autotest and secondly the correlation autotest.

As already described in 4.2.4.5, the output of this mode, independent if it comes from the neural network or the correlation autotest, is written to console and to files.

Neural network autotest output

Table 4.1 demonstrates the output of the neural network training. The MSE (as described in 2.7) is calculated both for the training set as well as for the validation set during training. The output is written on console as well as to the file *validation_ output-network.txt*.

epoch	training error/MSE	validation error/MSE
i	e_{train}	e_{valid}

Table 4.1: Neural network training output

Table Table 4.2 on page 81 describes the output of neural network testing, as done in auto test mode after every fold of the k-fold validation algorithm. For each entry of the test set the classification of the network just trained on the training sets (with early stopping by the validation set) is estimated, the MSE for each of those entries is calculated and finally, the real classification keyword is received by translating the binary output back to the original gesture/command name.

The first row of the output demonstrates the actual binary encoding of the current test case ($b1$, $b2$, $b3$, $b4$), the second line shows the classification by the network as a floating point numbers (w, x, y, z) between 0 and 1. If a value is close to 0 (and lower than 0.65), it is seen to be binary 0, if the value is close to 1 (or above or equal 0.65), it is seen to be binary 1. The third line now shows the MSE e for the current example. If this MSE exceeds a certain threshold, classification is stopped and an empty classification string is returned (and thus line 4 remains empty). This threshold was set to 0.1, so about 10% deviation is allowed for a classification to be still valid. The fourth line shows the actual command the current test case represents, while the last line displays the interpretation of the classification result from line two by the machine and remains empty if the MSE of the current example tested is above the threshold 0.1.

current test optimal output	b1	b2	b3	b4
classification result	w	x	y	z
current test MSE	e			
current test command	command1			
classification command	command2			

Table 4.2: Neural network testing output

This output is created for all k folds of the algorithm with a short summary at the end of each of the folds as can be seen in table Table 4.3 on page 81. The report represents the testing results of the current test set with $x + y + z = sizeof(testset)$. e is again the MSE, but this time the mean over the complete test set.

num correctly classified	x
num not classified	y
num incorrectly classified	z
total mean squared error	e

Table 4.3: Neural network k-fold report

At the end of the k-fold validation of the neural network, a final report is shown with the same form as the interim reports, but showing the results over all test sets with the mean squared error over all test entries.

Correlation auto test output

Table 4.4 shows the output produced for each of the examples. The first two columns indicate the current split index and the current example index, both starting at 0. The third column shows the actual classification of the current example and the fourth column the classification predicted by the correlation method. The last column shows the highest R^2 for the current example.

Split index	Example index	Actual command	Recognized command	Highest R^2
i	j	command	command	maxR^2

Table 4.4: Output example for k-fold cross-validation

Table 4.5 shows the summary output for the same examples with an entry for each of the commands and a summary for the complete set. For each of the commands, the total number of examples m, the number and percentage of the correctly classified examples a (s as the according percentage), the number and percentage of the not classified examples b (t the according percentage) with a too low R^2 for classification and the number and percentage of the incorrectly classified examples c (u the according percentage) are shown. Additionally, the mean R^2 of all the correctly classified examples is shown. The following formulas have to be fulfilled:

$$m = a + b + c$$

$$s + t + u \equiv 100\%$$

In the complete summary, the total number of examples and the numbers and percentages of the correctly-, not- and incorrectly classified examples are shown, with the formulas $n = d + e + f$ and $x + y + z = 100\%$. n signifies the number of all examples, d the number of the correctly classified examples (with x being the according percentage), e the number of not classified (with y being the according percentage) and f the number of incorrectly classified (with z being the according percentage).

command			
total num of examples	m		
num recognized	a	percentage	s%
num not recognized	b	percentage	t%
num false positives	c	percentage	u%
mean R^2 (recognized)	meanR^2		
Summary			
total num of examples	n		
num recognized	d	percentage	x%
num not recognized	e	percentage	y%
num false positives	f	percentage	z%

Table 4.5: Summary output for the k-fold cross-validation

Interpretation of the output is quite straight forward. If the percentage of the number

of recognized values in each of the command outputs as well as in the summary is high, the generalization of the system has a high relevance and the system is thus likely to produce usable output for manipulative input by gesture recognition. If any of the percentages from the command output of the number of recognized values is low (i.e. below 70%), the system is likely to fail on recognizing the according gesture/command. If there is a percentage different to 0 for any of the incorrectly classified gestures for any of the commands, it is very likely that the system will produce faulty output for the according gesture/command. It can also mean, that two gestures are too much alike in both movement and form of the hand, thus a quick solution for the problem would be in deleting the entries of the command from the sequence information file and choosing a different gesture for the command.

4.2.6 Logging

Logging of the recognized gestures and thus the outgoing commands sent to the iCOMMIC is done via the iCOMMIC system. Internal logging of correlation output and similar information is currently done by printing to the console. No additional logging of internal processes was planned. Please see the chapter future outlook (Section 6.2) for further details.

5 System assessment and correction

A very important step of every software developing process is the verification and validation of the system implemented. As any system is split up into smaller groups of problems, these smaller groups can be assessed independently. Automated tests should be used where possible to ensure highest possible correctness of the single sub-solutions to the given problems. Sometimes automated testing is very costly as far as time and calculational power is concerned, especially when trying to do so online. In such situations it is necessary to find other understandable and unquestionable methods to prove correctness.

5.1 Verification

Image pre-processing

Image pre-processing was visually examined at every step of the implementation of this part of the project. The correctness of the image processing and hand extraction was non-automatically verified by assessing each and every recorded frame by visual inspection. The feedback mechanism used here is the video debug mode as described in 3.2.4.1. Please see the future outlook in Section 6.2 for further ideas on this topic.

Data recording and loading

As far as correct data recording is concerned, online self-checking was used to examine recorded data for length of entries, in other words it was verified that the size of the entries lies within the given boundaries. For example, the number of entries of form vectors was not allowed to exceed the image height for the horizontal vector and the width for the vertical form vector. The number of movement points per sequence was not allowed to exceed the self defined number of frames per movement (10 in this case).

Furthermore, the movement vectors were checked for valid entries, so that no entry of the movement vectors would exceed the height or width of the image. For the form vectors, it was checked that the contained entries would not exceed the 10 frames times the activation of maximum 320 entries for the horizontal vector and 240 entries for the vertical form vector. In other words, no entries with values higher than 3200 for the horizontal and 2400 for the vertical form vector were allowed.

The same process was also used to verify that data is loaded correctly. Additionally, the loading of data was conducted using the XML loading mechanisms of the external tinyXML library, where the structure of the XML file is checked for correctness online, returning a failed state if incorrect.

Connection to iCOMMIC

As far as the connection to the iCOMMIC and thus the correct transmission of data is concerned, the verification was conducted by assessing the immediate output of the iCOMMIC, that means if the correct gesture recognized and transmitted from the GRM to the iCOMMIC led to the pressing of the correct button. For follow-up ideas on this topic, please see the future outlook in Section 6.2.

5.2 Validation

For validation of the program, it was necessary to thoroughly execute validation steps throughout the development of the project.

The part most relevant for validation is definitely the recognition module itself. Validating the recognition process was done implementing automatic testing in form of the validation methods described in Section 2.9. The following two chapters show the output of automatic testing (and thus validation) for a self recorded data set using the gestures as described in 3.3.7. For the output format and its interpretation details please see 4.2.5.4.

The data set used contains 179 recordings of the presented gestures, with at least 20 examples each. By using k-fold cross-validation with $k = 10$, a data set of 179 examples leads to 9 sets of 18 examples and one of 17. Before splitting the data set, the order of sets is randomized so that it is implausible that any gesture/command is not represented in a subset.

5.2.1 Validation of the neural network method

In the k-fold algorithm for neural network testing, there are always one subset of data used for validation and one subset of data for testing. This is due to the fact, that with the validation set, early stopping is performed on the neural network learning process as described in chapter 2.6.1.7. An actual significant statement about the quality of training can thus only be obtained by using a specific test set. Accordingly, the complete data are still split into 10 parts, however only eight sets are used for training of the network, the two remaining sets are used for validation and subsequent testing. This leaves a training set of either 142 or 143 examples, with both the test and the validation set containing either 17 or 18 examples.

The validity of the neural network method as shown by table Table 5.4 on page 87 can be seen as very high with 178 examples correctly classified (99.5%), 1 example not classified as a valid gesture (0.5%) and 0 sequences classified as a wrong gesture (0.0%). This suggests that the setup of the network (as described in chapter 4.2.2) was chosen such that the network is able to generalize very well and thus can be very well used for classification of new incoming gestures.

Tables 5.1 to 5.4 show the example output for one of the training/validation/test set combinations, namely with the validation set #4 and the test set #5, with #1-3 and #6-10 used as the training set.

Table 5.1 demonstrates the training output for this one fold of the k-fold algorithm, where neither the training error goal (0.0005) nor the validation error goal (0.005) was hit. Early stopping occured after 100 epochs due to the insignificant change of less than 0.00001 of the validation MSE for three validation rounds (so 30 training epochs).

epoch	training error/MSE	validation error/MSE
10	0.0504695	0.0105454
20	0.00922807	0.0186824
30	0.00511237	0.026321
40	0.00374059	0.0173212
50	0.00307116	0.0175942
60	0.00311905	0.00836974
70	0.00319815	0.00812515
80	0.00250864	0.0124869
90	0.00248192	0.0124869
100	0.00248192	0.0124869

Table 5.1: Training output

After this fold of training was finished, the current test set (#5) was used for testing. The whole set of 18 was used for testing. For demonstration purposes, two examples will be explained in more detail. In 5.2, the correct classification was achieved with a low MSE of 0.0022, while for the example in table 5.3, the classification failed due to incorrect outputs from the network and a thus too high MSE of 0.15. Line 4 in table 5.3 thus remains empty, as the MSE exceeds the maximal level of error allowed for correct classification (0.1).

current test optimal output	0	1	0	0
classification result	0.0110	0.9067	-0.002	0.0023
current test command	zoomOut			
classification command	zoomOut			
current test MSE	0.0022048			

Table 5.2: Example output correctly classified

current test optimal output	1	0	0	0
classification result	0.7116	0.1844	0.4615	0.5557
current test command	rotateLeft			
classification command				
current test MSE	0.159775			

Table 5.3: Example output not classified

Subtable a) in table 5.4 shows the summary for the test set #5, where 17 of the 18 test examples where correctly classified, while one was not classified. Subtable b) shows the complete summary over all test sets, with 178 of 179 examples correctly classified while one example could not be classified correctly.

a)

Test set #5 summary	
num correctly classified	17
num not classified	1
num incorrectly classified	0
total mean squared error	0.0141777

b)

Complete summary	
num correctly classified	178
num not classified	1
num incorrectly classified	0
total mean squared error	0.0054874

Table 5.4: Summaries for a single test set (left) and the complete summary for the whole process (right)

5.2.2 Validation of the correlation method

In this method, we use 9 subsets for building up the master templates and one subset for validation, so either 161 or 162 examples for training with 17 or 18 examples to test. This way, all of the examples of the data set are used 9 times for training and get tested once without being used for training in this turn.

The validity and thus the quality of the correlation method for classification can be easily assessed by inspecting the summary output (last subtable in Table 5.5 on page 88) and shows to be very high in this example (and thus the prime setup of the system) with a recognition rate of 97.2%, with only 2.8% not recognized at all and 0.0% false positives (so no classification of a gesture as a different type of gesture). More detailed output for the single commands can be seen in the rest of Table 5.5 on page 88.

Additionally, it is possible to see the quality of each of the commands/gestures by itself with additional information about the mean R^2 (for further reading on the R^2, please see Section 2.8). Assessing the single gestures/commands, you can see that each of the gestures is recognized with a rate of at least 95.2% with not more than 1 example not recognized and (as can be already seen from the summary output), with no false positive

recognitions. This shows that the recognition method returns very usable results for the given data set.

grab			
total num of examples	20		
num recognized	20	%	100,0
num not recognized	0	%	0
num false positives	0	%	0
mean R^2 (recognized)	0,938386		

last			
total num of examples	22		
num recognized	21	%	95,45
num not recognized	1	%	4,54
num false positives	0	%	0
mean R^2 (recognized)	0,90157		

next			
total num of examples	23		
num recognized	22	%	95,65
num not recognized	1	%	4,35
num false positives	0	%	0
mean R^2 (recognized)	0,932658		

release			
total num of examples	23		
num recognized	23	%	100,0
num not recognized	0	%	0
num false positives	0	%	0
mean R^2 (recognized)	0,930146		

rotateLeft			
total num of examples	25		
num recognized	24	%	96,00
num not recognized	1	%	4,00
num false positives	0	%	0
mean R^2 (recognized)	0,886338		

rotateRight			
total num of examples	23		
num recognized	22	%	95,65
num not recognized	1	%	4,35
num false positives	0	%	0
mean R^2 (recognized)	0,914984		

zoomIn			
total num of examples	21		
num recognized	20	%	95,24
num not recognized	1	%	4,76
num false positives	0	%	0
mean R^2 (recognized)	0,914984		

zoomOut			
total num of examples	22		
num recognized	22	%	100,0
num not recognized	0	%	0
num false positives	0	%	0
mean R^2 (recognized)	0,914984		

Summary			
total num of examples	179		
num recognized	174	%	97,2
num not recognized	5	%	2,8
num false positives	0	%	0%

Table 5.5: Example of the k-fold validation for the correlation method

6 Discussion

In this thesis, an approach towards hand gesture recognition for multi-modal user interfaces was presented. Using low cost equipment, the idea was to generate a system that recognizes previously defined gestures, making it possible to pass commands to the computer without having to use either mouse or keyboard. As navigational input was already available from the iCOMMIC system and it is furthermore difficult to cover navigational input without using extensive calibration and three dimensional imaging, the focus was set upon manipulative input.

Following the idea of creating a "one-for-all" and "hands-free" system, a simple, single webcam is used to record the working space of the user. Hand extraction is done by a simple colour space transformation and hysteresis thresholds on the colour channels of the LUV colour space. The extracted hand is tracked, sequence images are created and the sequence information is stored onto the hard drive. For training, the stored data is loaded from disk, the data is preprocessed, a neural network is trained which is used to classify gestures acted out in the field of view of the camera. Additionally, master templates are created from the stored, to which new data can also be compared for recognition. Commands associated to the recognized gestures are sent to the iCOMMIC system which finally acts out the actual command by emulating the pressing of a previously defined key.

6.1 Conclusion

The project presented in this thesis fulfills the requisitions as established during the planning phase, including the expert interviews. The system is able to extract the hand from the video, build up sequence images and recognize moved gestures in real-time and is invulnerable to incorrect user input. External tempering of recorded data has not been dealt with, but as this is a prototype for scientific purposes only, this fact is negligible. Thus, the system presented here is ready to be used in controlled environments for scientific purposes, but not yet for broad public usage. This is also to the fact, that the user interface is currently controlled by textual input via the console only and not via a graphical user interface.

In a controlled environment (as far as lighting, background colour and camera positioning is concerned) the quality of recognition can be described as very high. As soon as the environmental conditions change however, the quality of recognition suffers, as the hand extraction method is prone to become unstable when either the lighting or the background colour changes as artifacts in image processing may occur. Furthermore, it is vital to make sure that recorded gestures are acted out in a rather similar way every time, as the training system is susceptible to faults and cannot deal with outliers. Additionally,

the form recognition of the hand is not of much relevance, as the idea of summing up the hand form over time blurs the actual hand form for single frames and thus makes it almost impossible to differ between different postures. Due to this fact it is important to make sure that the gestures chosen for the system differ significantly as far as movement is concerned while the form of the hand is not as relevant at the moment. Please see the next Section 6.2 for ideas about how to improve the system regarding stability of hand extraction and recognition methods as well as some other ideas.

Generally speaking, the system can be seen as very fit for classification of previously recorded hand gestures. By the possibility of recording own gestures easily and training the system with the new gestures quickly, the system presented here might be one of the most configurable systems for hand gesture recognition there is. By the connection to the iCOMMIC which is also connected to an eye tracker, manifold possibilities for usage arise, from ease-of-use-testing of user interfaces to complete control of both navigational as well as manipulative computer input. In combination with the speech recognition as suggested in the future outlook (Section 6.2), this system is definitely a very interesting working base for a new alternative of computer control, both for "normal" as well as and especially for impaired computer users.

6.2 Future outlook

There are manifold possibilities to better the presented system.

Hand extraction

For example, the method for extraction of the hand from the image can be improved significantly by using the combinational approach using information from the HSV, the RGB and the YES colour spaces presented in 2.4.2.1 and proposed by Gomez et al in 2002 (Gomez et al., 2002). This was not done due to the fact that the transformation from one colour space to another needs time and computing power, so that three conversions and the extraction of the necessary information from those colour spaces may surmount the time available for single frame analysis. The paper doesn't treat the timing issue at all as this approach was implemented for offline usage, so no deduction could be drawn from there. It would definitively be interesting to find out how much the hand extraction could be improved by using such a combinational approach.

Significant point extraction

Instead of using the bounding box points for the movement, significant points could be extracted from the point cloud of the hand directly. For example it would be an interesting approach to calculate the center of gravity and the extremas of the hand, namely the fingertips. Creating vectors from the center of gravity to each of the extremas, it would be possible to calculate the angles between the extremas, thus allowing a more detailed description of the form of the hand. and thus allowing a more precise classification of smaller changes of the hand. This way, also fixed postures of the hand could be recognized easily, as the angles could be identified and an additional neural network could

be used to classify the postures. This would significantly improve the level of recognition as well as the possibilities of gestures to be used.

Logging

Logging could be essentially improved, as the current logging system depends heavily on the logging system of the iCOMMIC, which was so far not really used for logging incoming gesture commands, but rather for eye tracking input.

Validation of image processing & hand extraction

Automatic validation of the image processing and the hand extraction is so far not done at all, but there would be manifold possibilities to do so. One way would be to feed predefined images into the frame acquisition system with a certain amount of hand coloured pixel, calculate the output of the system and finally examine the actual number of valid pixel with the number of pixel extracted by the image processing process.

Validation of iCOMMIC connection

A simple test to validate the connection between the GRM and the iCOMMIC systems would be to store the command and the timestamp sent from the GRM to the iCOMMIC in a file, do the same on the server side (so at the iCOMMIC) and finally compare the two files.

Combination with linguistic systems

It would be very interesting to see if the additional modality of speech recognition would further enhance the comfort of usage of the system. As systems used for speech recognition are already available for many years, their quality can be seen as very high. Thus, a combination of eye tracking, gesture recognition and speech recognition might be the most interesting possibility for future projects focusing on new ways of communicating with computers.

Bibliography

Abidi, A. (1995). Direct-conversion radio transceivers for digital communications. *IEEE Journal of solid-state circuits*, 30(12):1399–1410.

Allen, D. (1971). Mean square error of prediction as a criterion for selecting variables. *Technometrics*, 13(3):469–475.

Anderberg, M. et al. (1973). *Cluster analysis for applications*. Academic press New York. ISBN-13: 978-0120576500.

Assan, M. and Grobel, K. (1998). Video-based sign language recognition using hidden markov models. *Lecture Notes in Computer Science*, 1371:97–110.

Barequet, G. and Har-Peled, S. (1999). Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 82–91. Society for Industrial and Applied Mathematics.

Bernacki, M. (2005). Principles of training multi-layer neural network using back-propagation. `http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop_files/img09.gif`. [Online; accessed 02-April-2010].

Blum, A. (1992). Learning boolean functions in an infinite attribute space. *Machine Learning*, 9(4):373–386.

Brand, J. and Mason, J. (2000). A comparative assessment of three approaches to pixel-level human skin-detection. In *International Conference on Pattern Recognition*, volume 15, pages 1056–1059.

Buskes, G. and Van Rooij, A. (2000). Almost f-algebras: commutativity and the Cauchy-Schwarz inequality. *Positivity*, 4(3):227–231.

Candolfi, A., De Maesschalck, R., Jouan-Rimbaud, D., Hailey, P., and Massart, D. (1999). The influence of data pre-processing in the pattern recognition of excipients near-infrared spectra. *Journal of pharmaceutical and biomedical analysis*, 21(1):115–132.

Canny, J. (1986). A computational approach to edge detection. *IEEE transactions on pattern analysis and machine intelligence*, 8:679–698.

CedarCrestCollege (2010). Neuron, impulse, synapse. `http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop_files/img09.gif`. [Online; accessed 02-April-2010].

Comer, M. and Delp, E. (1999). Morphological operations for color image processing. *Journal of Electronic Imaging*, 8:279.

Cross, S., Harrison, R., and Kennedy, R. (1995). Introduction to neural networks. *The Lancet*, 346(8982):1075–1079.

David, P. (1985). Clio and the Economics of QWERTY. *The American economic review*, 75(2):332–337.

Dougherty, E. and Lotufo, R. (2003). *Hands-on morphological image processing*, volume TT59. Society of Photo Optical.

Duchowski, A. (2002). A breadth-first survey of eye-tracking applications. *Behavior Research Methods Instruments and Computers*, 34(4):455–470.

Dzaack, J., Trösterer, S., Nicolai, T., and Rötting, M. (2009). icommic: Multimodal interaction in computing systems. *Proceedings of the 17th World Congress of the International Ergonomics Association, Peking*, ID 3EP0100.

English, W., Engelbart, D., and Huddart, B. (1965). Computer Aided Display Control—Final Report.

Fisher, B., Perkins, S., Walker, A., and Wolfart, E. (1994). Hypermedia image processing reference. ISBN-13: 978-0471962434.

Geisser, S. (1974). A predictive approach to the random effect model. *Biometrika*, 61(1):101–107.

Goldin-Meadow, S. (1999). The role of gesture in communication and thinking. *Trends in Cognitive Sciences*, 3(11):419–429.

Gomez, G. (2002). On selecting colour components for skin detection. In *International Conference on Pattern Recognition*, volume 16, pages 961–964.

Gomez, G., Sanchez, M., and Sucar, L. (2002). On selecting an appropriate colour space for skin detection. *Lecture Notes in Computer Science*, 2313:3–18.

Harrison, B., Fishkin, K., Gujar, A., Mochon, C., and Want, R. (1998). Squeeze me, hold me, tilt me! An exploration of manipulative user interfaces. *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1:17–24.

Haykin, S. (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR Upper Saddle River, NJ, USA.

Holzinger, A. (2005). Usability engineering methods for software developers. *Communications of the ACM*, 48(1):71–74.

Intel (2009). An introduction to neural networks with an application to games. `http://software.intel.com/en-us/articles/an-introduction-to-neural-networks-with-an-application-to-games/`. [Online; accessed 02-April-2010].

Jolliffe, I. (2002). *Principal component analysis*. Springer verlag. ISBN-13: 978-0387954424.

Joos, M., Roetting, M., and Velichkovsky, B. (2003). Spezielle Verfahren I: Die Bewegungen des menschlichen Auges: Fakten, Methoden, innovative Anwendungen. *Psycholinguistik–Ein internationales Handbuch*, 1:142–168.

Kent, J. (1983). Information gain and a general measure of correlation. *Biometrika*, 70(1):163.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International Joint Conference on Artificial Intelligence*, volume 14, pages 1137–1145. LAWRENCE ERLBAUM ASSOCIATES LTD.

Larsen, R. and Marx, M. (2001). *An introduction to mathematical statistics and its applications*. Prentice Hall. ISBN-13: 978-0132018135.

Liu, X. and Chen, T. (2003). Video-based face recognition using adaptive hidden markov models. *Proc. of IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 1:340–345.

McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5(4):115–133.

Michie, D., Spiegelhalter, D., Taylor, C., and Campbell, J. (1994). *Machine learning, neural and statistical classification*. Ellis Horwood Ltd. ISBN-13: 978-0131063600.

Mitchell, M. (1998). *An introduction to genetic algorithms*. The MIT press. ISBN-13: 978-0262631853.

Nolker, C. and Ritter, H. (1998). Illumination independent recognition of deictic arm postures. *Proc. 24th annual conference of the IEEE Industrial Electronics Society*, 4:2006–2011.

Oxford English Dictionary, O. (2009). *Definition of gesture*. Oxford University Press. ISBN 0199563837.

Pattanaik, S., Ferwerda, J., Fairchild, M., and Greenberg, D. (1998). A multiscale model of adaptation and spatial vision for realistic image display. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 287–298. ACM.

Prechelt, L. (1998). Early stopping-but when? *Neural Networks: Tricks of the trade*, 1:553–553.

Rigoll, G., Kosmala, A., and Eickeler, S. (1997). High performance real-time gesture recognition using hidden markov models. *Gesture and sign language in human-computer interaction*, 1:69–80.

Rodgers, J. and Nicewander, W. (1988). Thirteen ways to look at the correlation coefficient. *American Statistician*, 1:59–66.

Roetting, M. (1999). Typen und Parameter von Augenbewegungen. *Blickbewegungen in der Mensch-Maschine-Systemtechnik*, 1:1–19.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386–408.

Roweis, S. and Saul, L. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323.

Rumelhart, D., Hinton, G., and Williams, R. (2002). Learning representations by back-propagating errors. *Cognitive modeling*, 1:213.

Schou, T. and Gardner, H. J. (2007). A wii remote, a game engine, five sensor bars and a virtual reality theatre. *OZCHI '07: Proceedings of the 19th Australasian conference on Computer-Human Interaction*, 1:231–234.

Schrepp, M. (2006). On the efficiency of keyboard navigation in web sites. *Universal Access in the Information Society*, 5:180–188.

Sears, A., Feng, J., Oseitutu, K., and Karat, C. (2003). Hands-free, speech-based navigation during dictation: Difficulties, consequences, and solutions. *Human-Computer Interaction*, 18(3):229–257.

Smith, D. and Alexander, R. (1988). *Fumbling the future: How Xerox invented, then ignored, the first personal computer*. William Morrow & Co., Inc. New York, NY, USA. ISBN-13: 978-1583482667.

Smith, T. and Guild, J. (1931). The CIE colorimetric standards and their use. *Transactions of the Optical Society*, 33:73–134.

Sobel, I. and Feldman, G. (1968). A 3x3 isotropic gradient operator for image processing. presented at a talk at the Stanford Artificial Project in 1968, unpublished but often cited, orig. In *Pattern Classification and Scene Analysis, Duda, R. and Hart, P., John Wiley and Sons*, volume 73.

Søgaard, H. and Olsen, H. (2003). Determination of crop rows by image analysis without segmentation. *Computers and electronics in agriculture*, 38(2):141–158.

Stark, M., Kohler, M., and Zyklop, P. (1995). Video based gesture recognition for human computer interaction. Technical report, University of Dortmund, Germany.

Stone, G. (1986). An analysis of the delta rule and the learning of statistical associations. *Mit Press Computational Models Of Cognition And Perception Series*, 1:444–459.

Swain, M. and Ballard, D. (1990). Indexing via color histograms. In *Computer Vision, 1990. Proceedings, Third International Conference on*, pages 390–393.

Takahashi, T. and Kishino, F. (1991). Hand gesture coding based on experiments using a hand gesture interface device. *ACM SIGCHI Bulletin*, 23(2):67–74.

Tkalcic, M. and Tasic, J. (2003). Colour spaces: perceptual, historical and applicational background. In *Eurocon*.

Turk, M. (2002). Gesture recognition. *Handbook of virtual environments: Design, implementation, and applications*, 1:223–237.

Tuttle, R. (1986). Computer input devices. *Software in healthcare*, 4(2):50.

Valensi, G. (1961). System of television in colors. US Patent 2,982,811.

Van Buskirk, R. and LaLomia, M. (1995). A comparison of speech and mouse/keyboard gui navigation. In *Conference on Human Factors in Computing Systems: Conference companion on Human factors in computing systems*.

Weeks, A. (1996). *Fundamentals of electronic image processing*. Wiley-IEEE Press. ISBN : 9780470544709.

Werbos, P. (1974). *Beyond regression: New tools for prediction and analysis in the behavioral sciences*. Harvard University.

WikipediaUser:Chrislb (2010). Artificial neural network. `http://commons.wikimedia.org/wiki/File:ArtificialNeuronModel_english.png`. [Online; accessed 02-April-2010].

WikipediaUser:SimpsonsContributer (2010). Demonstration of the canny edge detect operator. `http://en.wikipedia.org/wiki/File:Valve_original_%281%29.PNG`. [Online; accessed 02-April-2010].

Zhang, Z. and Zha, H. (2004). Principal manifolds and nonlinear dimension reduction via local tangent space alignment. *SIAM Journal of Scientific Computing*, 26(1):313–338.