Universiteit van Amsterdam

Master Thesis

Sonic Gesture

Author:
Gijs Molenaar

Supervisors:
M.Sc. Ivo Everts
Dr. Theo Gevers

October 6, 2010

Abstract

A realtime method for estimating hand poses in video using a normal RGB camera is presented. The method uses a skin color distribution for pixel based limb localization and HOG features for classification. The performance of HOG is compared with SURF where HOG yielded superior performance. As classifiers k-NN and SVM with the RBF and χ^2 kernels are evaluated, where SVM in combination with the χ^2 kernel performed best. The evaluation is done on a new constructed dataset consisting of 72 recordings of 20 people performing 28 hand symbols based upon the Curwen Solfege method. A prototype implementation that translates estimated hand poses into Open Sound Control parameters is described, which can be used to make music. This to demonstrate the practical usability of the method in a real-time situation.

Contents

Acknowledgements

Until not so long ago I never dared dreaming to do this; writing the acknowledgments of my Master Thesis. I'm really proud I came this far and I'm really content with the subject and results of my research. It took a total of 9 months to finish the software, gather a dataset, read papers and write this thesis. I never worked on a project this long.

The Master at the University of Amsterdam was the most exciting but also most challenging time of my life. I would never have made it, without the help and inspiration I got from people in my surroundings.

First of all I want to thank Ivo Everts, who gave up a lot of his time to supervise, motivate and inspire me. Countless cigarets where smoked during our weekly meetings, and I hope this will keep happen sometimes in the future.

I want to thank Maarten van Someren for helping me to bridge the gap between the Bachelor of Arts and the Master of Science. It was a difficult time and I really had to catch up with a lot of math and the scientific methodology back then.

I want to thank my parents for supporting me financially and motivating me for so long, it would have been very difficult to do this Master without that help.

I want to thank Anne Schuth, Arjan Nusselder and André van der Vlies for reviewing this thesis and giving valuable feedback.

I want to thanks my girlfriend Anouk Hövels - even though I think she has no clue what this thesis is about - has always supported me, motivated me and helped me where she could.

I want to thank Anne Schuth, Arjan Nusselder, me, Ivo Everts, Jasper Stroomer, Peter, Hanne Nijhuis, Jasper 2, Ork de Rooi, Roberto Valenti, Xiron, Gosia, Hamdi, Michael, Sil Westerveld, Victoria, Bas van der Vlies, Koen, Chu and Stratis for participating in my dataset and I'm sorry if I didn't remember your surname.

I want to thank Intel for making OpenCV open source, it is a great library for computer vision and it will be even better in the near future.

and finally, I want to thank everybody who I forgot, everybody that gave me good idea's and tips for Sonic Gesture.

Chapter 1

Introduction

1.1 Background

Human-Computer Interaction

Human-Computer Interaction (HCI), the communication between humans and computers is limited in transmission rate. The amount of information per second is still low nowadays, and innovation is progressing slowly. Since the introduction of the mouse in the seventies very few revolutionary new and still affordable peripherals that aid the communication with a computer where introduced. The most notable innovations in HCI are the Wii - which is mainly used for gaming - and (multi) touch screen, which is becoming more popular for integration smartphones.

New and revolutionary ideas are required to create new peripherals that improve productivity, but are still intuitive and thus easy to learn to use. To get inspiration for improvement, one can look around and study already existing communication methods, for example the communication between humans.

When two people are in a room and do not have an auditive or visual limitation, they will probably communicate by speech. But there is much

Figure 1.1: The first mouse

more going on than only producing and interpreting words. The intonation, speed and other small variations in the voice add a lot more information to the words. Also the facial expression and body language give more space for expression. Some people like to 'talk with their hands' while telling a story, something that adds more expression to the transmitted information.

Sign Language

A deaf person cannot interpret spoken words, at least not by listening. He or she is highly depended on visual information. Sign languages have been emerged or invented to aid this visual communication. In these languages two elements play an important role: the face and the hands. These body parts give the most expressive power. The face because it is very good for expressing feelings and emotions, and the hands because they are very deformable. This makes the hands very interesting and useful, since there are countless combinations of finger poses and orientations.

Speech synthesis [Hunt and Black, 1996], speech recognition [Rabiner and Juang, 1993], facial expression recognition [Cohen et al., 2003] and sign language interpretation [Cooper and Bowden, 2007] are all subject of extensive research. Breakthroughs in these areas are important for revolutionary new interfaces which can tremendously improve the speed of interaction and usability for a user.

This thesis aims at looking into the details of using computer vision to interpret sign language and especially to use sign language to actually control the computer - to use your hands for non-intrusive interaction. To show the advantages of having such a system it would be interesting to find an application for it and demonstrate that this can be actually useful. The application found was sound generation and manipulation, or in other words; making music.

Translating sign language into music is a very interesting concept, not only because it sounds poetic but it can really demonstrate the power of such a system. Making live music is a complex process where real-time interaction between a artist and his tools is important. Also, for an audience it is much more compelling to look at a artist who is physically more active than only clicking a mouse.

Hand Poses

Sign language does not really have distinct signs for musical concepts. Usually, when music is translated into sign language only the vocals are translated. Facial expression and references to emotions are used to indicate the

Figure 1.2: Guidonian Hand

'mood' of the music and the speed and expressiveness of the gestures indicate the intensity of the music, but these kind of gesture are not really usable for HCI because they are too subjective. A more formal method is required. Also, previous studies have shown that the segmentation of the consecutive gestures is a difficult task [Buehler et al., 2009, Bowden et al., 2004].

It is more feasible to have a limited set of hand poses that corresponds to a set of commands and parameters. Any set of hand poses would do for this system, as long as the individual poses do not look too similar. Since the idea is to translate the interpreted hand poses into sound, it would be interesting to use a set of poses that has an already existing relationship to sound.

In Medieval music, the Guidonian hand was a mnemonic device used to assist singers in learning to sight sing; singing according to visual perceived information like sheet music or hand gestures, typically not seen before. The idea of the Guidonian hand is that each portion of one hand represents a specific note within the hexachord system, which spans nearly three octaves. The other hand is used to point to the correct hand portion. Figure 1.2 shows a hands with the tonal positions.

Despite the fact that this system has a large set of symbols - 22 to be exact - this system is not usable since the individual poses are very much alike. Discriminating between the different positions on the hand will be problematic.

A more recent method using hand poses in relation to music are the Curwen solfege hand signs [Choksy and Lois, 1999]. This method was introduced in the 19th century by John Curwen, who also is the founder of the famous

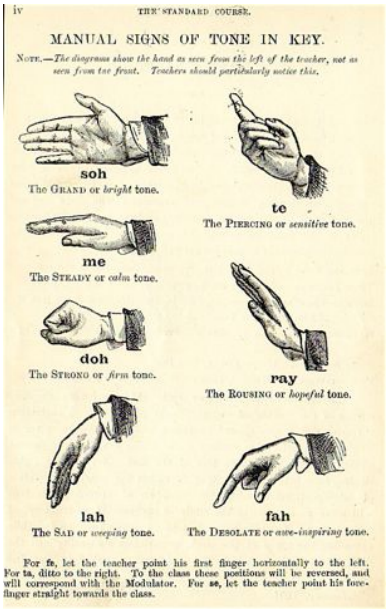


Figure 1.3: Depiction of Curwen's Solfege hand signs from 1904

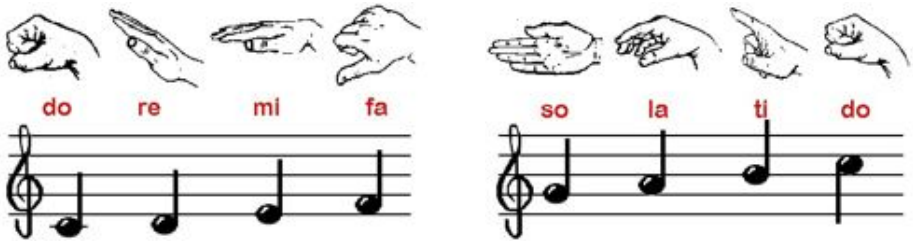


Figure 1.4: The curwen hand symbols and the corresponding musical notes

tonic sol-fa. The tonic sol-fa is better known as *'do re mi fa sol la ti'*.

Figure 1.3 is a scan from a teaching book from 1904 where the 6 tonal hand poses are shown. These 6 poses correspond to the 6 notes in the musical major scale. These hand poses are much more suitable for our system, since the individual hand symbols are distinct. Also the hand poses can be easily performed by both hands individually next to the body or in front of the body. Figure 1.4 shows the same hand poses with the corresponding musical tones. It is less common known that there are also names for the chromatic increase and decrease, flat (♭) or sharp (♯) tones. These names are *'di, ri, fi si li'*. Also these tonal names have their own corresponding Curwen hand poses.

1.2 The Goal

The goal of this thesis is to describe the design and evaluation of a system for hand pose recognition. The system needs to be fast and user friendly; it should be able to run on current consumer hardware. Overall the system should satisfy the following requirements:

- Localize and interpret the hand poses of a person in a video stream
- Do this with real time performance; a minimum of 10 frames per second (FPS). 10 FPS is enough for the human perception to perceive consecutive images as a movie.
- Processor power requirements should be moderate; it needs to work on a normal consumer desktop or laptop
- Use a normal inexpensive RGB camera or webcam, without infrared
- Non-intrusive, no gloves or skin mounted electromechanical sensors
- No calibration or initialization is required at startup or during the usage
- Minimal to no configuration parameters are desired.

To realize these requirements some restrictions on the system's setting are required:

- There should be only one person at a time in the image
- The person is wearing clothing with long sleeves, no arm skin is visible
- The lightning conditions should be 'good enough', enough light should illuminate the user
- There should be no skin like colors in the image like pictures and posters with faces.

The system can be used as a controller for a computer, with a focus on subtle detail for continuous variables, which give the sense of more direct control which is important for real time interaction. Akin to how a graphical designer prefers a graphics tablet over a mouse for drawing, the design of the system aims to accomplish a similar sensation. The output of the system can be easily configured by a user to map to his or her preferred set of commands. For example, one hand can control the pitch of a sound, while the other controls an accompanying beat while the position of this hand controls a sound manipulating variable.

It is important to say that the scope of this thesis is limited to extracting the hand poses from video, and will not cover the study of the mapping of

Figure 1.5: Microsoft Kinect

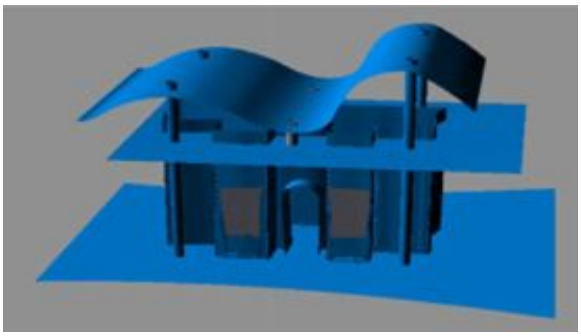


Figure 1.6: 3D model of train station made with aid of hand gestures

hand poses into sound although a prototype system is discussed in chapter chapter 5.

1.3 Related Work

At the moment of writing this thesis Microsoft is finishing the development of a new commercial product called 'Kinect'. Kinect is claimed to provide full-body 3D motion capture. To accomplish this, Kinect uses a range camera, which interprets 3D scene information from a continuously-projected infrared pattern. For this setting an infrared projector and a range camera is required. It is interesting to see that again (this is also the case for the Wii) a new revolutionary input device is created for a gaming console.

[Yi et al., 2009] proposed a method for computer aided 3d architecture design using hand gestures, see Figure 1.6 for the a model constructed with this software.

A lot of work has been done in the field of hand pose recognition. [Erol et al., 2007, Mitra and Acharya, 2007] give a good overview of what has been done and what techniques have been used. [Stenger, 2006] describes a similar

Figure 1.7: 3D hand pose estimation using a colored glove

approach as in this paper, but uses template matching and k-NN. Also a similar approach is [chih Wang and chih Wang, 2007], but SIFT features are used and only 3 hand poses are detected. Both papers don't have a dynamic skin model. From the same author is [Wang and Popović, 2009], where a colored glove is used to perform full 3d hand pose estimation which seem to yield promising results. The disadvantage of this method is the intrusive requirement of wearing a specific color glove, see Figure 1.7.

Other studies require extra non-consumer hardware; [Mo and Neumann, 2006] proposes a method for pose estimation from low-resolution depth images from a laser camera. [Xiong et al., 2006] describes a method for 3D hand path tracking in a meeting setting with multiple camera's. A 3d model fitting approach is described in [Athitsos and Sclaroff, 2003, de La Gorce and Paragios, 2010]. 3D model fitting can yield more accurate results, but the search space is much bigger and it will be very difficult to gain real-time performance.

An alternative approach to hand localization is to capture the complete body pose. This could be a more robust method, but is at the moment also computationally expensive. [Ferrari et al., 2008] introduces a method to extract a body pose from monoscopic video. [Van den Bergh et al., 2009] proposes a method for 3D pose recognition using multiple camera's. [Poppe, 2007] discusses how 3D model fitting can be used for pose estimation. [Moeslund et al., 2006] gives an overview of all relevant body pose estimation research until 2006.

Interesting research is going on in analyzing real sign language, something Sonic Gesture can't do, since it is limited to poses. [Buehler et al., 2009] discusses a method to learn sign language with news broadcasts that are subtitled and translated into sign language. The complexity of segmentation in sign language is discussed in [Bowden et al., 2004]. An other approach for segmentation is discussed in [Cooper and Bowden, 2007]. [Starner et al.,

1998, Vogler and Metaxas, 1999] use Hidden Markov Models to model sign language.

Chapter 2

Body Part Detection

To perform hand pose estimation of a person in a image, hands need to be localized first. Hands are very deformable - they can have very different appearances and different orientations. Also hand size, hand shape and skin color can differ tremendously per person. This makes localization a difficult task.

One way to localize the hands is by searching for skin like colors in an image given a pre-calculated color profile. A generic skin color model based on average skin color has been constructed for this purpose [Jones and Rehg, 1999]. The problem with this method is that it fails when the skin color is not 'pure', for example when it is colored by a light source. Also, since the profile is very generic, it contains colors for many different skin colors. This introduces more false positives. It would therefore be preferred to to obtain the skin color of a person in a image from that image itself. This can be done by extracting the skin color distribution from the face, which is much easier to detect than a hand.

This chapter describes how a skin distribution is constructed and how this is used to find the skin pixels in the image. These skin pixels are clustered and labeled as left hand, right hand and head.

2.1 Skin Segmentation

Face localization

Finding faces in an image is a rather well solved problem. A face is easy to detect, since it is almost non-deformable. Different faces are quite similar looking at the most prominent properties, e.g. edges. People do not tilt their head often or not more than a few degrees, which makes it even easier to

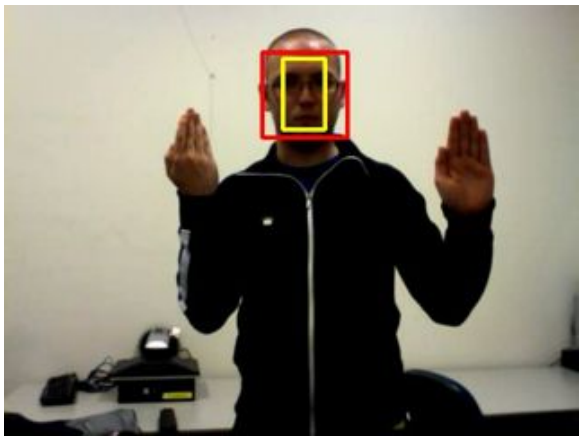


Figure 2.1: Face detection

detect. Face detection can be done in a robust way using a *haar classifier*, a boosted rejection cascade that is trained with Haarlike wavelet [Viola and Jones, 2001, 2004]. This method does not rely on color information. The classifier is run over the image on different scales and positions with a score above a certain threshold are classified as face position, see Figure 2.1 for an example. The red rectangle is the returned face location and size. This classifier is trained with traindata which also includes a small area of background, so this is also returned. Since we are only interested in the skin pixels a smaller sub-region is used for further processing. This sub square can be adjusted to minimize the influence of facial hair.

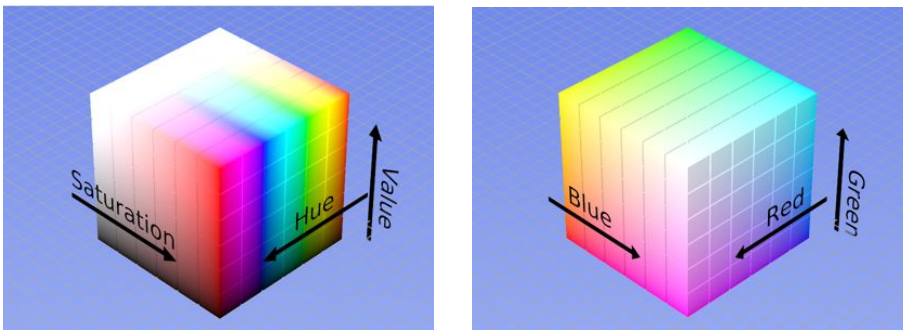
Unfortunately, the face detection is a relative expensive operation. Fortunately, as a face does not move fast in a video sequence, a number of frames can be skipped which will free more computational time for other operations.

With the face location and size a skin color distribution of the user can be constructed.

Color Space Conversion

Usually, pixel values of a image are stored in the Red, Green Blue (RGB) color space where colors are represented as combinations of these primary colors. The RGB representation of colors is not suitable for modeling skin color, because it represents not only color, but also luminance which is not a reliable measure for segmenting skin pixels [Cai and Goshtasby, 1999].

When a subject is illuminated by a light source with uniform hue distribution, the light does not change the hue or saturation of the subject. The only thing that will change is the luminance, which is changing because of the



(a) HSV color space **(b)** RGB color space

Figure 2.2: The RGB and HSV color space

light source's intensity, distance or (self casted) shadow. Since we want to extract hand pixels independent of the illumination intensity, we can ignore this channel and only use the hue and saturation.

The luminance can be removed from the color space by transforming the color distribution to a chromatic color space. There are multiple chromatic color spaces, for example LAB, HSV and normalized RGB. No significant improvements are measured with a specific choice of these color spaces, so the HSV color space is used in the rest of this paper [Vezhnevets et al., 2003, Bradski, 1998].

HSV stands for Hue (color), Saturation (how concentrated the color is) and Value (brightness). In practice a light source never has a uniform distribution and *will* change the color of the subject. Fortunately, since we build a color distribution from the image itself, all skin pixels will change in the same way.

The RGB color space is transformed into the HSV color space using the following equations:

$$V \leftarrow \max(R, G, B) \tag{2.1}$$

$$S \leftarrow \begin{cases} \frac{V - \min(R,G,B)}{V} & \text{if } V \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

$$H \leftarrow \begin{cases} \frac{60(G-B)}{S} & \text{if } V = R \\ \frac{120+60(B-R)}{S} & \text{if } V = G \\ \frac{240+60(R-G)}{S} & \text{if } V = B \end{cases} \tag{2.3}$$

Note that for computation reasons this a simplification of the real HSV color space. The real HSV colorspace is circular in the Hue dimension.

(a) hue channel **(b)** saturation channel **(c)** value channel

Figure 2.3: The HSV channels

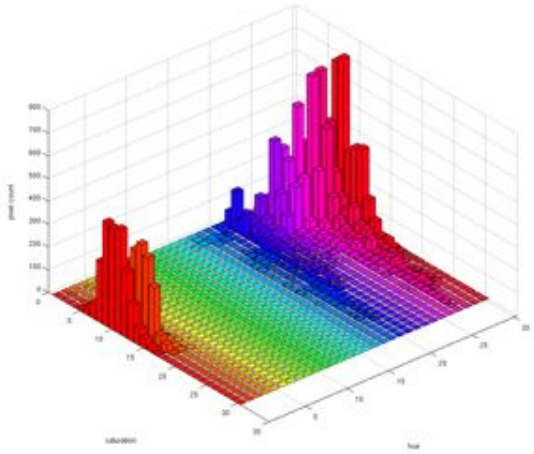


Figure 2.4: A histogram of face pixels

The input 3-channel RGB image is transformed into a 3 channel HSV image, after which the value channel is discarded. Using the detected face region of the hue and saturation channel a histogram can be constructed which will represent a statistical skin distribution.

Skin Color Distribution

The skin color distribution is represented in a 2 dimensional histogram, where the first dimension is Hue and the second is Saturation. The histogram is filled with the values from the detected face region. The histogram is normalized - all values are divided by the sum of all bins - to properly approximate the color density. This makes the histogram independent of the original image size, and each histogram bin value represents a 'skin probability'. Figure 2.4 is the histogram of a caucasian's skin color.

Manual experiments show that tweaking the number of bins does not have

Figure 2.5: Backprojection

much effect, as long as the number of bins is not too high or not to low. Since for the storage of the values a 8 bit integer is used, a number of bins higher than 256 does not make sense. In all experiments mentioned in this paper a number of bins of 30 is used for both the hue and saturation.

Back Projection

A back projection is the combination of an image and a distribution. The result is a new single channel image. All pixels in the input image are iterated and the corresponding bins are looked up in the histogram. The pixel in the same position in the new image is replaced with the value from the histogram. If the histogram is a skin color histogram, the resulting image will have high values for pixels that are skin-like pixels, and low values for other pixels. The result of the process can be seen in Figure 2.5. Since the probabilities are very low the contrast of this image is enhanced so the maximum pixel value becomes pure white.

Smoothing

Back projection can be quite noisy, which is caused by the rounding of values in the in the histogram and noise introduced by the camera. Thresholding this image will result in skin pixel groups with rough edges and a lot of holes, see Figure 2.6. The noise can be reduced by smoothing the image. This way, the pixel value is replaced by the old value weighted with the surrounding pixel values. Using a gaussian kernel with a high variance gives a good result, see Figure 2.7.

Figure 2.6: Thresholded image without blur preprocessing

Figure 2.7: Blurred image

Threshold

A mechanism is required to label each pixel as skin or non-skin. The easiest way to accomplish this is by defining a threshold. All pixel values below a certain threshold are replaced with false/non-skin, all above this threshold

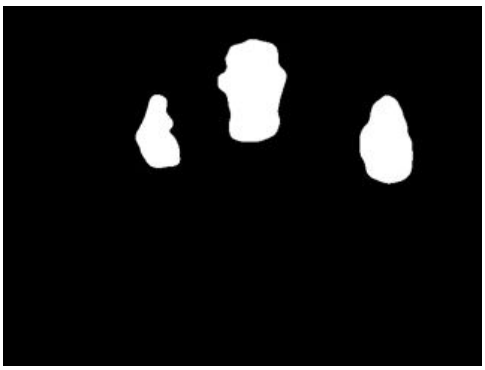


Figure 2.8: Thresholded image with blur preprocessing

Figure 2.9: Morphologically closed

will be replaced with true/skin. This results in a binary image with labels for (non) skin pixels. This introduces one parameter - the threshold for going from the probabilistic domain to the binary domain.

This parameter could be adjusted automatically, by comparing the percentage of skin pixels in the image and comparing this to a believed proper percentage. This percentage should be made dependent on the size of the detected face. If the percentage is too low, then the threshold should be lowered and raised if too high.

An alternative method of going from the probabilistic domain to the binary domain is adaptive thresholding. Here the threshold is determined per pixel by the values of surrounding pixels. This method has one parameter; the neighborhood blocksize n is used to determine the threshold. The threshold per pixel is calculated by the equation

$$\mathrm{dst}(x,y) \leftarrow \begin{cases} 1 & \text{if } \mathrm{src}(x,y) > T(x,y) \\ 0 & \text{otherwise} \end{cases}, \tag{2.4}$$

where $T(x,y)$ is the mean of the n by n pixel neighborhood.

Morphological Operations

An alternative to smoothing out rough edges and holes are (combinations of morphological) operations. A morphologic closing operation of A by B is obtained by the dilation of A by B, followed by erosion of the resulting structure by B. The result of performing this operation is that small holes in the binary image are removed, and edges are smoothed as seen in Figure 2.9. The effect is not really significant, since the gaussian smoothing already removes a lot of noise.

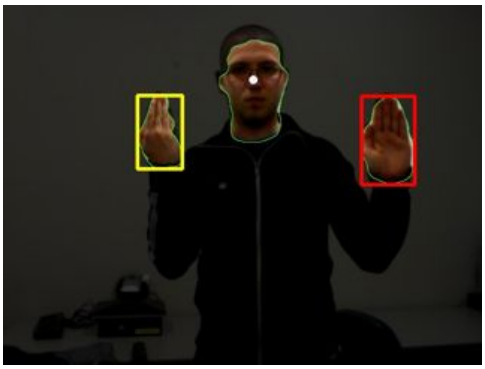


Figure 2.10: Labeled blobs with hand windows

2.2 Object Localization

Pixel Grouping, Contour Extraction

To be able to say something useful about groups of pixels, one needs to know which pixels belong together. The human eye perceives neighboring pixels as a whole, but this is less obvious for a computer. Grouping pixels is called clustering. In this case clustering is performed by grouping pixels together that touch horizontally and vertically. Each cluster of pixels is called a blob.

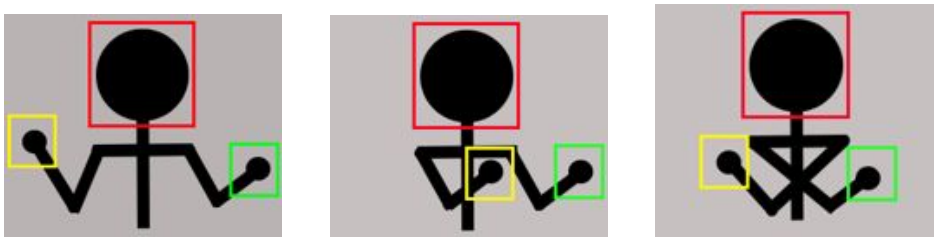
To handle the blobs in a time efficient way, it is a good idea to represent the blob in a more compact form; the contours. The contour is a minimal list of pixels that contain all other pixels in the blob, also called a convex hull. In this way interesting problems can be solved more efficiently like determining if a certain pixel coordinate is inside a certain blob, the maximum or minimal horizontal or vertical position and hole removal.

Since it is unusual to have holes in blobs that represent skin regions, these can be discarded. This is done with the algorithm described in [Suzuki and Abe, 1985], where the contours of the blobs are extracted and all contours except the outer most contour are removed. This removes holes and islands in holes.

From the contours of the group, a square region of interest is defined by the outer borders. This window is called the hand window from now on.

Blob Labeling

To interpret a blob the label of that blob is required. A blob can be a hand, a head or noise mislabeled as a body part. This noise can reduced by



(a) (b) (c)

Figure 2.11: 3 scenarios for labeling of blobs

assuming a body part has a minimal size. If the surface of a blob is smaller than a certain value this blob can be discarded.

The threshold for noise is calculated by

$$T = (\frac{h}{c})^2, \tag{2.5}$$

where h is the height of the face in pixels and c is a parameter to control the effect of the threshold. A small experiments shows that a value of 3 for c removes a lot of noise blobs, but still is small enough to leave most hand blobs intact.

Multiple cues can be used to determine which blob is which body part. The most certain and important one is the face center, which is known as we detected the face. If a blob contains this point, this blob is the face.

To determine which blob is which hand the relative x position to the head can be used. The hands are sorted by x position and labeled accordingly. The result of the heuristics in different scenarios is shown in Figure 2.11. Subfigure (a) shows the usual position, where the left hand is on the left of the face and the right hand on the right. The labels are labeled accordingly. In some cases both hands are on one side of the face (b), here the relative position is used to label the hand. If only one hand is detected the relative position to the head is determined, left to the head is labeled left and the same for right. This is a robust method, except for the scenario where the hand positions are swapped in the x position. Fortunately this is a unnatural body pose, so this case is just simply ignored. If such a scenario occurs the labels of the hands are mixed.

Blob label stabilization

A hand does not move very fast in an image - usually it will not move from the left side to the right side in one frame. If this is detected this is probably a measurement error caused by noise or pour labeling. In this case the history of previous positions of a blob must be incorporated. This can be done with a Kalman filter Welch and Bishop [1995]. A Kalman filter is an easy and fast method for smoothing out the current position with the previous positions. The result will be a more stable estimation of the hand position. A second advantage of the Kalman Filter is the ability to actually predict the position of the hand in the next frame. This can become useful when there is no new hand detected. The hand position can then be estimated with a different method that will the the Kalman prediction.

For every hand a Kalman filter is initialized. The measurement that need to be smoothed is the hand window. The hand window has a x and y position and a width and hight. Each hand also has a speed in the x and y direction but we do not measure that - we let the Kalman filter represent, calculate and use that internally.

A hand window represented in a measurement vector as

$$m_k = \begin{pmatrix} x_k \\ y_k \\ w_k \\ h_k \end{pmatrix}, \tag{2.6}$$

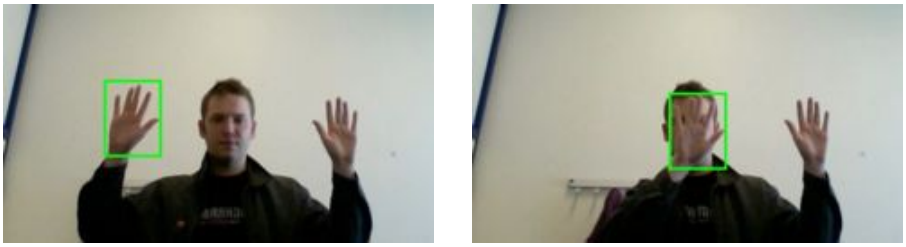
where x_k is the horizontal position, y_k is the vertical position, w_k is the width and h_k is the height of the hand window.

The Transition matrix is defined as follows

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2.7}$$

This is just an identity matrix, except for the ones on the right top in the matrix. These represent the combination of the current position and the internal state of the speed.

With these matrices and a standard Kalman filter the position and size of the hands can be predicted. The predicted values are used for an other hand localization method in case no hand is found en the next frame.



(a) detected hand **(b)** result of template search

Figure 2.12: Example of template search

Self Occlusion by Body Parts

Failing to detect the hand in the current frame is caused by one of three disturbances. First of all the hand can be out of the image, or occluded by an obstacle. The second case is where the hand detection phase just fails and couldn't localize the body part. The third case is self occlusion, where the hand is very close or occluding the face or the other hand. The skin segmentation will segment this as one big blob. Disturbance one and two are fatal and hand localization is impossible, but for the third case there is a last resort. Here template search is used to track the specific hand. A cut out image of the hand from the previous frame is used for this template search. Template search is a very simple and fast method, as long as the search area is small. A sliding window with the same size as the cutout image is sliding over a small surrounding area of the location predicted by the Kalman filter. The window with the lowest squared sum difference to the previous cutout image is set as the new location. If the original cutout is touching the border of the image or the squared sum difference is too high it is assumed the hand is not visible anymore, and is flagged accordingly. This method works if the shape and size of the hand do not change too much from the last.

Figure 2.12 shows a test setting of a template search. In the left image a hand was found using the skin color distribution. In the second frame the hand cannot be found, because it is occluding the face and using the skin distribution approach the hand will be labeled as face. Using a template search the hand can still be tracked.

2.3 Discussion

This method is quite robust, but can fail if the conditions listed in section 1.2 are not met. See Figure 2.13 for an example of failed segmentation. This

Figure 2.13: Example of failed segmentation

Figure 2.14: Adaptive threshold with pixel neighborhood of 51

figure is a still from one of the movies in the dataset used in the experiments. The test subject has a skin color profile that is similar to parts of the background. Not much can be done to solve this problem, except the threshold can be manually adjusted. Still, this will introduce more false negatives and the segmentation will still be poor.

Thresholding can be adjusted automatically by adaptive thresholding. The downside of the method is that it requires a large neighborhood value to work in a reasonable stable way, which becomes too computationally expensive. Also, the background will be clustered as a blob also, which introduces an extra blob removal step. Figure 2.14 shows the example output of a adaptive threshold with a neighborhood of 51.

Chapter 3

Pose Recognition

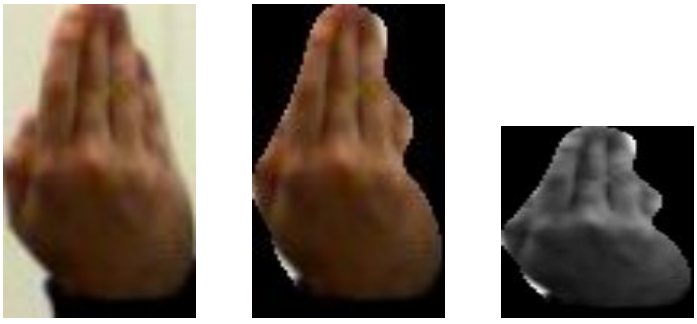
Knowing the location of the hands in a image does not tell much about the hand poses. A method needs to be constructed to discriminate the different hand poses.

This chapter describes how the hand poses are detected in the hand windows. Features are extracted from the hand window and represent the hand pose in a more compact way. These features are compared by a classifier with previously extracted features of hand poses for which the labels are known. The classifier will determine which known hand pose(s) resembles the at that moment examined hand.

3.1 Feature Extraction

The hand window contains some background pixels, because a hand will never fill a perfect square (Figure 3.1a). These pixels are unwanted since they contain arbitrary values that introduce noise into our process. In section 2.1 a binary mask for skin pixels is constructed. The inverse skin mask can be used to remove the background (Figure 3.1b). Sometimes the blob is too small to say something useful about the hand pose. One way to artificially increase the blob size is to do a morphological dilate operation on the blob. This will increase the size of the hand cutout and probably add skin pixels, but this will also introduce more noisy (non-skin) pixels.

To compare two or more hands with each other a method is required to calculate the similarity. A very simple method is to compare each pixel location with each other and sum all the differences. But this method is not feasible; it is not going to work on images that are not the same size or orientation. Also it is computationally expensive for larger images - for an image of 100 by 100 pixels there are already 10.000 dimensions. Also



(a) Hand window **(b)** Hand cutout **(c)** Resized hand, used for feature extraction

Figure 3.1: Preparing the hand window for feature extraction

important is that this method is not very forgiving for small variations such as scaling and rotation.

Descriptors

To reduce the number of dimensions of an image a descriptor can be used. A descriptor describes an image in, if properly used, less dimensions than the image itself. Also it can introduce some useful properties like scale and rotation invariance. In this paper 2 descriptors are discussed and compared, Histogram of Oriented Gradient [Dalal et al., 2006] (HOG) and Speeded Up Robust Features [Bay et al., 2006] (SURF).

The HOG descriptor has been successfully applied and studied in human detection [Dalal et al., 2006, Watanabe et al., 2009]. The HOG descriptors method uses a dense grid of uniformly spaced cells, where for each cell the gradients are calculated in 9 orientations. The values for each orientation for each cell are stored in a histogram that represent the image. Figure 3.2 visualizes a simplification of the HOG calculation.

The SURF descriptor is based on sums of approximated 2D Haar wavelet responses and makes use of integral images. SURF approximates the speed of Scale-invariant feature transform (SIFT) and is claimed to more robust against several image transformations [Murillo et al., 2007, Valgren and Lilienthal, 2010]. SIFT is an other descriptor that is often used. SIFT is similar to HOG as in they both build a histogram of gradients of the key-points. SIFT and SURF differ from HOG in that they incorporate a method for calculating usable key points, HOG uses a dense grid of key points.

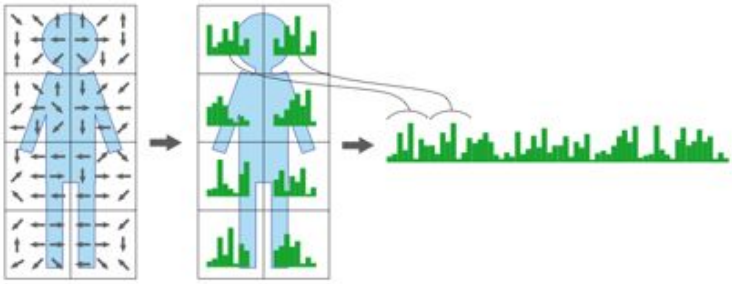


Figure 3.2: Visual representation of HOG calculation

3.1.1 Principal Component Analysis

An other approach for dimension reduction is Principal Component Analysis (PCA). PCA is a method that reduces data dimensionality by performing a covariance analysis between vectors Jolliffe [2002]. The original space will be mapped into a new space based on the variance in the original data. PCA transforms a number of correlated variables into a (possibly smaller) number of uncorrelated variables called principal components. The first principal component vector accounts for as much of the variability in the data as possible, and each consecutive component vector accounts for as much of the remaining variability as possible.

PCA may improve the performance of a classifier both in classification and time performance. PCA can also be used in combination with the previously described descriptors.

3.2 Classification

Using a set of training examples each marked as belonging to a category, a classifier builds a model that predicts whether a new example falls into one category or the other. There are many classifiers which can be used in different ways. For computer vision two of these are interesting because of simplicity, or superiority; k Nearest Neighbors and Support Vector Machines.

k Nearest Neighbors

The k-Nearest Neighbors algorithm (k-NN) is a method for classifying objects based on euclidean distance between training examples in the feature space. k-NN is one of the the simplest of all machine learning algorithms. It

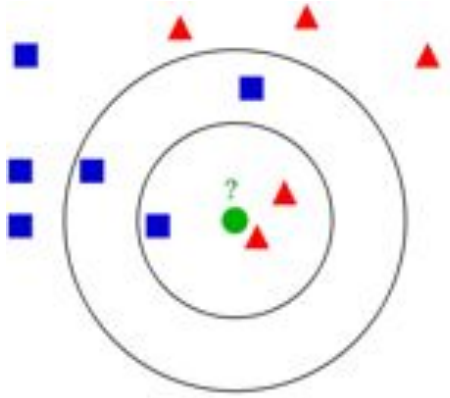


Figure 3.3: Classification using K Nearest Neighbors

is a type of 'lazy learning' where all computation is deferred until classification. This results in a fast training phase, but a computationally expensive classification. Also the memory footprint is large since all data points are stored.

The training samples are labeled multidimensional vectors. The training phase of the algorithm consists only of storing the vectors and class labels of the training samples. In the classification phase, k is a user-defined constant, and an unlabeled vector is classified by assigning the label which is most frequent among the k training samples nearest to that query point.

Figure 3.3 is an example of k-NN classification. The test sample (the circle) should be classified either to the squares class or to the triangles class. If $k = 2$ the circle is classified to the triangles class because there are 2 triangles and 1 square inside the inner circle. If $k = 3$ it is classified as 'square'.

Support Vector Machines

Support Vector Machines (SVMs) are a set of related supervised learning methods used for classification. It is a binary classifier, but can be extended to be a multi class classifier by combining multiple binary classifiers.

SVM uses a subset of the train data (the support vectors) that lay on the borders of a category cluster to calculate a Decision Boundary (DB). This DB is constructed by maximizing the distance between the support vectors of two classes. The decision boundary, a linear function, is later used to classify new data points.

Often, a set of data points in a space is not linearly separable. For this reason the original space is mapped to a higher dimensional space making the separation easier. This is done by a kernel function.

The training phase consists of computing a kernel to make the space linearly separable. Then a decision boundary is calculated. Training a SVM classifier can be computationally expensive, but classification is very fast and memory efficient. The training phase has one parameter, which is the error cost c. c controls the trade off between allowing training errors and forcing rigid margins. It is a multiplication of the error - the difference between a predicted value and true value. This allows some flexibility in separating the categories and, if properly set, prevents over and under fitting.

The function for the DB for a test sample x has the following form:

$$g(\mathbf{x}) = \sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) - b, \tag{3.1}$$

where $K(x_i, x)$ is a kernel function for the training sample x_i and the test sample x and y_i the class label of x_i with value $+1$ or -1. α_i is the weight of the training sample x_i and b is a threshold parameter which both are learned in the training phase.

There are various kernels, but [Hsu et al., 2003] states the Radial Basis Function is a good kernel to start with because it generally gives good results. This kernel is defined by:

$$K(S_i, S_j) = e^{(-\gamma||S_i - S_j||^2)}, \gamma > 0, \tag{3.2}$$

where S_i and S_j are two feature vectors. It has one parameter γ which needs adjusting to fit the dataset. It controls the scale of the kernel. An other interesting kernel is based upon the χ^2 distance[Zhang et al., 2007]. An advantage is that it does not have a parameter that needs optimization. This distance is incorporated into SVM by the usage of Gaussian kernels[Chapelle et al., 1999]:

$$K(S_i, S_j) = exp(-\frac{1}{A} D(S_i, S_j)), \tag{3.3}$$

where $D(S_i, S_j)$ is defined as

$$D(S_1, S_2) = \frac{1}{2} \sum_{i=1}^{m} \frac{(u_i - w_i)^2}{u_i + w_i}, \tag{3.4}$$

with $S_1 = \{u_1, ..., u_m\}$ $S_2 = \{w_1, ..., w_m\}$.

Figure 3.4 is an example of a SVM classification setting. The space is already linearly separable, so no space transformation is required.

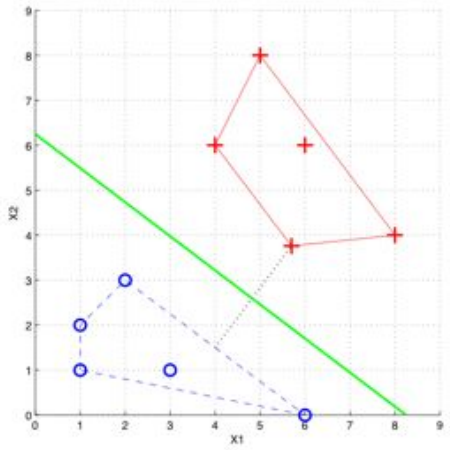


Figure 3.4: Classification using Support Vector machines

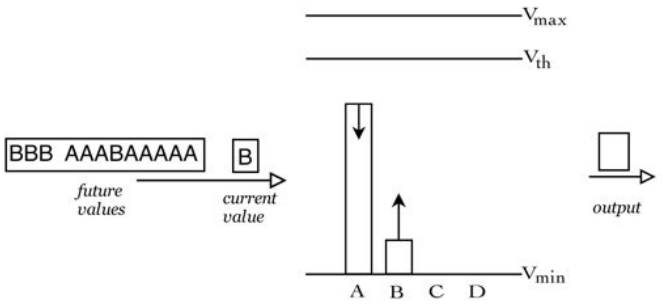
The Stabilizer

Since consecutive frames in a video sequence have a spatial-temporal relationship their labels have this property as well. Because of noise, misclassification can happen. These false matches can be filtered out by smoothing the labels on a time scale.

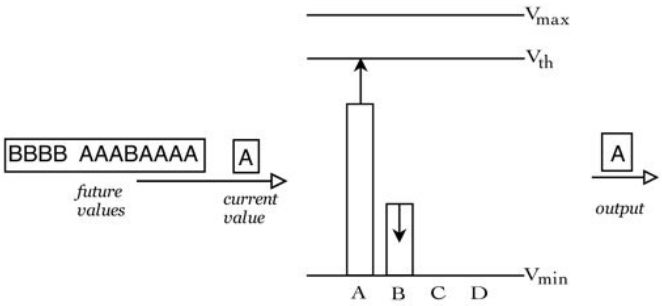
This smoothing is done with a simple self invented method called 'the stabilizer'.

A visual representation is given in Figure 3.5. The stabilizer is initialized with n numbers of bins which is equal to the number of labels. There are 2 parameters, V_{max} which controls the maximum and V_{th} which controls the threshold.

For every new label that is given by the classifier all bins are decremented with 1, except for the bin with the currently classified label which is incremented. A bin is incremented until it reaches V_{max}. When at any moment one of the bins value rises above V_{th}, the stabilizer will output the label of that bin. The result is a more stable and smoothed stream of labels, where single noisy labels are filtered out. For a sequence of frames with a correctly detected pose there is a delay between the first frame and the stabilizer will output the correct label. This delay is controlled by V_{th} which is measured in frame count. A higher value will reduce more noise, but will give a bigger delay. There is also a delay after a sequence which is controlled by n_{max}. With a framerate between 10 and 25 frames a second, $V_{max} = 15$ and $V_{th} = 10$ give good results reducing noise and still being very responsive.



(a) A



(b) B

Figure 3.5: Two examples of a stabilizer in action

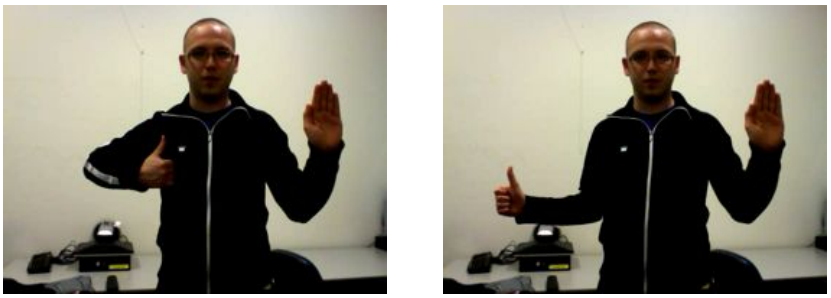
Chapter 4

Experiments

4.1 The dataset

A dataset was constructed where people perform the Curwen hand poses in different settings. Usually the 12 Curwen solfege hand symbols are performed in front of the torso. To increase the number of hand poses to be recognized and increase the usability, all 12 Curwen are also performed mirrored next to the body of the recorded subject, see Figure 4.1. Additionally four extra hand symbols have been added that are not part of the Curwen sequence. These last four symbols are performed next to the head. This results in a total of 28 hand poses.

In total there are 74 movies containing 20 different people performing the complete sequence of 28 hand symbols. At first, people were recorded performing the complete sequence 5 times, but this was taking too much time and people became impatient. After we switched to 3 movies per person. For each pose in each movie a frame was manually labeled where the per-



(a) in front of torso **(b)** next of torso

Figure 4.1: The Curwen hand poses

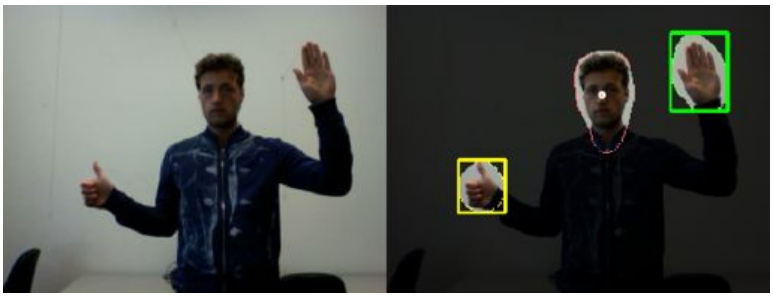


Figure 4.2: Still of movie ivo5 with simple background

Figure 4.3: Still of movie gosia3 with complex background

son was performing the pose correctly. The movies where recorded with a resolution of 532x400 and a frame rate of 10 fps. The test subjects were recorded while looking at a computer screen and asked to mimic the examples as in Figure A.2. 12 test subjects were recorded with a simple (almost empty and smooth) background (Figure 4.2), 3 were recorded with a complex background (Figure 4.3) and 6 were recorded with the same complex background but also with a poster with skin like colors.

The nationality of the test subjects is diverse. Half of the set is Dutch and two are from Asia. The rest is from eastern and southern Europe.

Figure 4.4: Still of movie sil1 with complex background with skin like poster

Test Subject	Movies	Background
Anne	5	Simple
Arjan	5	Simple
Gijs	5	Simple
Ivo	5	Simple
Jasper 1	5	Simple
Peter	5	Simple
Hanne	5	Simple
Jasper 2	3	Simple
Ork	3	Simple
Roberto	3	Simple
Xirong	3	Simple
Gosia	3	Complex
Hamdi	3	Complex
Michael	3	Complex
Sil	3	Complex + poster
Victoria	3	Complex + poster
Bas	3	Complex + poster
Koen	3	Complex + poster
Chu	3	Complex + poster
Stratis	3	Complex + poster

Table 4.1: Dataset details

4.2 Evaluation

4.2.1 Part I - Evaluating Classifiers

In this section different classifier with different parameters are evaluated on subsets of the dataset.

Method

All hand windows for each symbol in each movie are extracted and the features are extracted and stored. This data in then imported in Matlab, where the experiments are performed. For the SVM classifier the libsvm[1] package is used. To perform PCA the prtools[2] package is used. For nearest neighbors the KNN implementation in the biolearning package of matlab itself is used. The evaluation of the classifiers is split in three runs per classifier:

[1] http://www.csie.ntu.edu.tw/ cjlin/libsvm/
[2] http://www.prtools.org/

K-fold per movie using simple dataset only Only the movies with a simple background are used. For each run one movie is used for testing and all other movies are used for training. For each recorded person in the test set there are also 2 or 4 recordings of him or her used in the train set. This setting mimics the real life situation where the system is pre-trained with other people *and* the user, with a simple background.

K-fold per person All movies or used, but per person the movies are used for testing and all other movies for training. Both complex and simple backgrounds are used. This setting mimics the real life situation where the system is pre-trained with only other people than the user of the system, and the background is not necessarily simple.

Simple as train set, complex as test set The classifier is trained with all movies with the simple background, and as a test set the movies with the complex background are used. This mimics the real life situation where the training is with recordings in optimal conditions, but the system is used in a complex environment like a living room. Also the system is not trained by the user. One time the complete test is run with HOG features, and one time with SURF so we can compare the performance. The SURF features are only used in this experiment, since this one takes the least time, running the others experiments again with SURF would take a couple of days.

Test setting

For the number of neighbors for k-NN a small number of tests were run, and a value between 3 and 10 for N yielded similar performance. Since a lower number of k can give better performance $k = 3$ was used for all experiments.

When PCA was performed on the dataset the smallest eigenvectors were removed such that 95% of the original variance was remaining. On the simple dataset this results in a reduction from 3780 to 594 dimensions. For the complex set combined with the simple set the dimensionality is reduced to 688 dimensions, for complex only 283 and for the complete dataset 749.

Two kernels are evaluated, RBF and χ^2. The c for both kernels and γ for RBF values where found by an extensive grid search on the 'per person' test set with only the simple movies. The grid search was performed on a small Matlab cluster in the ranges $[2^{-3}, 2^{15}]$ for c and $[2^{-15}, 2^{3}]$ for γ. The most optimal values for γ is 2^{-5}. With this γ changing the value for c didn't have much effect on the performance, so a value of 64 was taken.

(a) Re **(b)** Ri

Figure 4.5: Hand poses for *Re1* and *Ri1*

Classifier	PCA	Full scale	Major scale
k-NN ($k = 3$)	no	84.27%	90.21%
k-NN ($k = 3$)	yes	83.67%	89.70%
SVM RBF ($c = 2^6$ $\gamma = 2^{-5}$)	no	86.02%	90.88%
SVM RBF ($c = 2^6$ $\gamma = 2^{-5}$)	yes	86.85%	**91.85%**
SVM χ^2	no	**87.92%**	91.58%
SVM χ^2	yes	76.93%	83.83%

Table 4.2: k-fold per film using simple dataset only,

For calculating the HOG features the same parameters as [Watanabe et al., 2009] are used, except that the image is not resized to 64 by 128 pixels, but to 128 by 128 pixels.

To be able to compare the HOG and SURF, the SURF descriptor is configured to work with a fixed set of interest points set up as a dense grid - the same set used for HOG. This results in a 13440 dimensional feature vector. Also using the interest point detection is a expensive operation, calculating this on a small image takes about 500 ms which makes it unusable for real-time operation.

The Curwen hand symbols that are closely related in a musical scale way are quite similar, for example *Re* and *Ri* (Figure 4.5). It is expected that a lot of misclassifications will be caused by this similarity. To investigate the impact, 2 different scores are calculated, one for the major scale and one for the full scale. The full scale threats every class as a single class, for the major scale the notes in the major scale and their corresponding similar notes are joined into one class. *Do* is combined with *Di*, *Re* with *Ri*, *Fa* with *Fi*, *Sol* with *Si* and *La* with *Li*.

Results

K-fold per movie using simple dataset only Table 4.2 shows the result for this experiments. k-NN performs worse in all cases. Using PCA with k-NN makes the results even worse. SVM with the RBF kernel performs

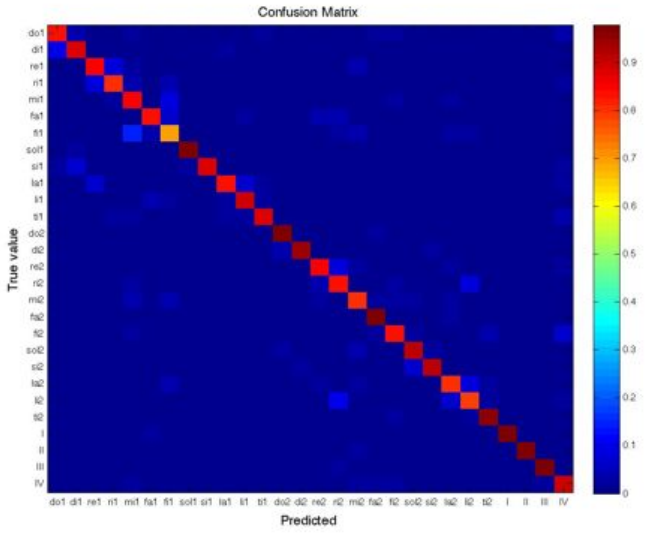


Figure 4.6: Confusion matrix for the n-fold per movie experiment

(a) Fi1 **(b)** Sol1 **(c)** Do2 **(d)** Fa2

(e) Ti2 **(f)** I **(g)** II **(h)** III

Figure 4.7: *Sol1, Do2, Fa2, Ti2, I, II* and *III*

better on the Major scale, but the χ^2 kernel performs better on the Full scale.

Figure 4.6 is the confusion matrix of the best results for the full scale, a SVM classifier with a precomputed kernel using the χ^2 distance. Most notable is the poor performance for the *fi1* hand pose, see Figure 4.7a. This is probably due to the lack of skin surface and unique features. On the opposite *sol1, do2, fa2, ti2, I, II, III* and *IV*, perform near perfect. These hand poses, shown in Figure 4.7, expose a lot of surface and gradients, and also possess unique features.

Classifier	PCA	Full scale	Major scale
k-NN ($k = 3$)	no	66.98%	76.32%
k-NN ($k = 3$)	yes	65.31%	76.00%
SVM RBF ($c = 2^6$ $\gamma = 2^{-5}$)	yes	**73.38%**	81.44%
SVM χ^2	no	71.83%	80.07%
SVM χ^2	yes	72.43%	**82.82%**

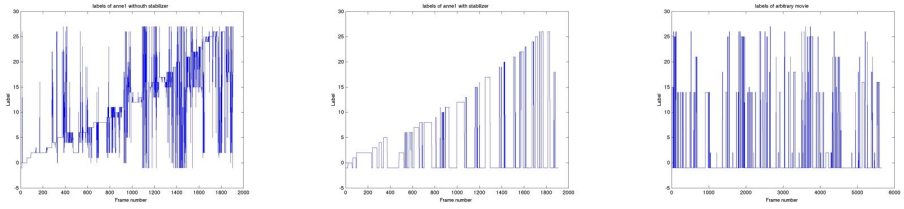
Table 4.3: k-fold per person

Descriptors	Classifier	pca	Full scale	Major scale
Hog	k-NN ($k = 3$)	no	58.57%	72.78%
	k-NN ($k = 3$)	yes	57.62%	72.17%
	SVM RBF ($c = 2^6$ $\gamma = 2^{-5}$)	yes	62.14%	73.39%
	SVM χ^2	no	**63.81%**	**74.71%**
	SVM χ^2	yes	58.57%	71.98%
SURF	k-NN ($k = 3$)	no	46.19%	60.17%
	k-NN ($k = 3$)	yes	44.05%	56.65%
	SVM RBF ($c = 2^6$ $\gamma = 2^{-5}$)	yes	44.29%	57.18%
	SVM χ^2	no	37.93%	47.91%
	SVM χ^2	yes	53.33%	65.13

Table 4.4: simple as trainset, complex as testset

K-fold per person In Table 4.3 the results of this experiment are shown. Also here, SVM has superior performance compared to k-NN. SVM with the RBF kernel performs better on the Full Scale, while the χ^2 kernel performs better on the Major scale. Again, PCA improves the performance for SVM but not for k-NN. It is interesting to see that the overall performance is worse than the previous experiment. This is due to the fact that the train set doesn't contain extracted hands from the test subject during k-fold. It shows that Sonic Gesture works better when it is additionally trained by the end user.

Simple as train set, complex as test set The results of this experiment are presented in Table 4.4. Again, SVM performed better than k-NN, but now the χ^2 kernel performs better than the RBF kernel in both cases. But overall the difference is minimal. The abnormality here is that PCA did not improve the results. The performance of the previous experiment is better than this experiment. Also for this experiment the train data doesn't contain hands from the same person in the test set. This experiment also shows that the a complex background has a negative influence on the classification performance.



(a) Without stabilizer **(b)** With stabilizer **(c)** Arbitrary movie with stabilizer

Figure 4.8: Plots of labels per frame

Even more interesting is the fact that the SURF features perform very bad compared to HOG features. Performing PCA on the SURF features improves performance dramatically. The horrible performance of SURF is probably caused by the improper usage of the keypoints.

4.2.2 Part II - Evaluating Sonic Gesture

Method

To evaluate Sonic Gesture in the time domain, the sequential output must be evaluated. Sonic gesture outputs for every frame the estimated label for each hand. Constructing a ground truth for each frame is a difficult, time consuming and subjective process. The material is also 'polluted' with wrong hand poses caused by misunderstanding or misinterpretation. In some cases there are also more people in the image than only the test subject. To evade these problems a different evaluation method is used, where the consecutive changes in labels is evaluated. Since the goal for each recorded movie in the dataset was to record a person performing the 28 hand poses sequentially, the ground truth is a ordered list of all labels.

Figure 4.8 visualized the sequence of labels in three cases, (a) when no stabilizer is used, a erratic pattern is the result. When the stabilizer is used (b) a more smooth incremental label pattern is visible. (c) shows the label sequence of an arbitrary movie. Scoring these patterns would be useful for comparison an tweaking. This can be done by calculating the Damerau-Levenshtein distance to the ground truth.

the output of Sonic Gesture for all movies is stored. This results in a long list of labels - one label for each frame - for each move. In each list all repeating labels are replaced with one instance of that label. This reduces the length of each list significantly. Now this list can be used to compute the Damerau-Levenshtein Distance.

The Damerau-Levenshtein Distance algorithm is shown in Algorithm 1. The algorithm calculates the number of deletions, insertions, substitutions and transpositions required to transform one string into an other. The maximum number of operations is equal to the length of the shortest string. Because of the ground truth's length of 28 the minimum will always be 28 or less. This way the distance can be safely normalized and divided by 28, so all distances can be easily compared.

Algorithm 1 DamerauLevenshteinDistance(str1, str2)

Require: a string str1
Require: a string str2
Ensure: The distance between str1 and str2

for $i = 1$ to lenStr1 **do**
  $\mathbf{D}[i, 0] \Leftarrow i$
end for
for $j = 1$ to lenStr2 **do**
  $\mathbf{D}[0, j] \Leftarrow j$
end for
for $i = 1$ to lenStr1 **do**
  for $j = 1$ to lenStr2 **do**
    if $\text{str1}_i = \text{str2}_j$ **then**
      cost $\Leftarrow 0$
    else
      cost $\Leftarrow 1$
    end if
    $x \leftarrow \mathbf{D}_{i-1,j} + 1$
    $y \leftarrow \mathbf{D}_{i,j-1} + 1$
    $z \leftarrow \mathbf{D}_{i-1,j-1}+$ cost
    $\mathbf{D}_{i,j} \Leftarrow \min(x, y, z)$
    if $(\text{str1}_i = \text{str2}_{j-1}) \wedge (\text{str1}_{i-1} = \text{str2}_j)$ **then**
      $\mathbf{D}_{i,j} \Leftarrow \min(\mathbf{D}_{i,j}, \mathbf{D}_{i-2,j-2}+$ cost $)$
    end if
  end for
end for
return $\mathbf{D}_{lenStr1,lenStr2}$

Results

Table 4.5 contains all distances, grouped per dataset. The numbers in table heading are the movie number of that column. The average column is the *average* of all movies per person. The average per person column lists the same distances, but without using the stabilizer discussed in section 3.2. As

Dataset	Name	1	2	3	4	5	Average	Average without stabilizer
Simple	Anne	0.43	0.32	0.25	0.29	0.14	0.28	0.88
	Arjan	0.46	0.25	0.25	0.07	0.29	0.26	0.83
	Gijs	0.14	0.18	0.21	0.21	0.14	0.18	0.87
	Ivo	0.07	0.11	0.07	0.11	0.04	0.08	0.86
	Jasper 1	0.68	0.46	0.21	0.21	0.43	0.40	0.85
	Peter	0.25	0.29	0.25	0.32	0.18	0.26	0.86
	Hanne	0.32	0.00	0.04	0.25	0.07	0.14	0.91
	Jasper 2	0.32	0.18	0.18			0.23	0.90
	Ork	0.21	0.18	0.14			0.18	0.86
	Roberto	0.68	0.29	0.46			0.47	0.83
	Xirong	0.61	0.50	0.36			0.49	0.88
Complex	Gosia	0.57	0.32	0.21			0.37	0.88
	Hamdi	0.50	0.43	0.54			0.49	0.87
	Michael	0.21	0.29	0.46			0.32	0.90
	Sil	0.50	0.61	0.46			0.52	0.83
	Victoria	0.32	0.21	0.18			0.24	0.88
	Chu	0.29	0.21	0.07			0.19	0.89
Complex	Koen	0.39	0.32	0.39			0.38	0.88
with	Bas	0.54	0.43	0.32			0.44	0.90
poster	Stratis	0.43	0.11	0.18			0.24	0.86

Table 4.5: Damerau-Levenshtein distance of all movies

you see this has a dramatic impact on the performance in this experiment. No clear patterns are visible, there is a small difference between the average distance for the simple set (0.27), and the complex set 0.35. This score tells more about how well the test subject performed the Curwen hand sequence. This score could be used to compare Sonic Gesture to other systems when the same dataset is used.

4.2.3 Part III - The Search for Spock

Method

To show that Sonic Gesture can also work with non artificial video material a third experiment was constructed. For this experiment a collection of real fragments of movies and pictures of people performing the 'Vulcan Salute' was gathered. The Vulcan Salute is greeting hand gesture and was popularised by the half-Vulcan character Mr. Spock on the Star Trek television

Figure 4.9: Mr. Spock giving the Vulcan Salute

series in end of the 1960s. Figure 4.9 is a movie fragment of Mr. Spock giving the hand salute. The original dataset used for the previous experiments was extended with a 29th hand pose, the vulcan salute.

Results

Figure 4.10 shows eight examples of successful detection of the vulcan salute. The images are cutout screenshots of the implementation of Sonic Gesture. The upper left window is the input, the upper right window is the segmented and labeled visualisation. A green square indicates a right hand and yellow left. The bottom squares show the estimated hand poses, the left square for the left hand and the right square for the right hand. For the visualisation of the detected Vulcan Salute the picture of Mr. Spock is used.

Figure 4.11 shows two screenshots of failed segmentation. Sonic Gesture fails on the first salute - performed by Condolisa Rise - because the skin segmentation fails. The skin colour of her face has more in common with the wall than with her hand. The second failure case is performed by Simon Pegg, here the segmentation is correct, but classification is just wrong. More training data could fix this problem.

Figure 4.10: Successful detection of Vulcan salute

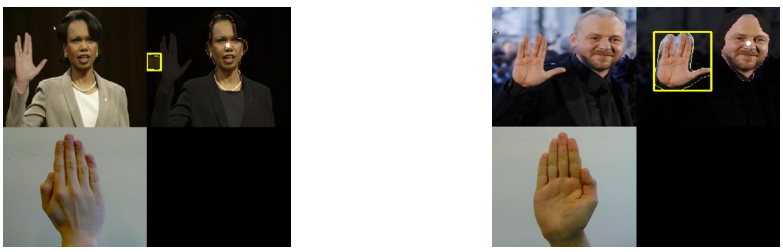


Figure 4.11: Failed detection of Vulcan salute

Chapter 5

Sonic Gesture

5.1 Implementation

Figure 5.1: Screenshot of Sonic Gesture

Sonic Gesture has been implemented in C++. Almost all computer vision algorithms used are part of OpenCV, an open source library made for this kind of software. For the graphical interface QT is used. Sonic Gesture has been release as Open Source software and is released under the Apache license. The software can be downloaded, modified and distributed freely from the website[1].

[1] http://code.google.com/p/sonic-gesture

Figure 5.1 is a screenshot of the main screen of sonic gesture. The program can capture directly from a webcam or it can read movies with recorded material. It has 2 modes, the first is the 'finder' mode where hand poses in the video stream are detected and classified. The second mode is the 'capture' mode, which is used to label movies. When labeling a movie in this mode a text file is created with frame positions of the labels. This can later be used to extract the correct frame and extract training data for the classifier. This mode has been used a lot during the gathering of the dataset.

When the finder mode is active, Sonic Gesture will translate the labeled hand poses, the hand positions and the size of the hand into Open Sound Control (OSC). OSC is a protocol which is specifically designed for transferring audio information over a network. For those who are familiar with digital music, it is very similar to MIDI. A program that can interpreted OSC can then be configured to listen on a specific port and process these OSC command.

Sonic Gesture will submit:

- For each hand the beginning and the end of detected sequence of hand specific hand pose. This is an integer. This can be used to trigger a action, like playing a specific note.
- The position of each hand in x and y position. This is a float and can be used to manipulate a parameter.
- The size of the hand. This is also a float and can also be used to manipulate a parameter.

There is no strict defined relationship between a hand pose and a sound or parameter, the user of Sonic Gesture can define his or her own specific mapping with apreferred audio application.

5.2 Time Performance

A lot of effort has been put into getting Sonic Gesture as fast as possible. Initially Sonic Gesture was written in Python and used the Python API of OpenCV. Soon it became clear that Python was to limited to do high performance graphic processing so a switch to C++ was made.

The performance of Sonic Gesture depends on how fast the testing systems CPU power is, how fast the camera can capture frames. Using Intel Performance Primitives (IPP), a proprietary library used to speed up intensive calculations on Intel hardware, may help also. On a Macbook Pro, 2.4 GHz intel core 2 duo with 4 GB of memory using the build-in iSight as camera, processing one frame takes 65 ms on average. This is with the full dataset of 2072 datapoints with 3780 dimensions using the KNN classifier. KNN

Figure 5.2: Sonic Gesture linked to Ableton with OSCulator

What	relative time	comment
kNN	47%	3780 features, 2016 samples
Color space convertion	8.4%	
Image resizing	7.6%	
Face detection	3.7%	19% if every frame
HOG features	2.6%	

Table 5.1: Performance timing of Sonic Gesture

works fine with low numbers of datapoint, but with high numbers it starts to slow down. Still it is quite fast; around 25 ms on average. SVM will probably perform faster with a high number of datapoints but the current implementation of SVM in OpenCV is problematic.

An other expensive operation is the face detection algorithm, when tweaked it takes around 13 ms to locale a face in a image. Since a face position is not required constantly this is done only every 10th frame, so valuable computation time is saved. An other surprisingly expensive operation is the resizing of a image. Resizing an image to a small size is a crucial part of the pipe line, because some operations on a big image will take too much time. But resizing a image using interpolation is a expensive operation. Where interpolation is not required, for example for rendering on the screen, it is disabled and thus saving computing time.

calculating SURF features of a hand train image takes 4ms and results in 11 features on average.

Chapter 6

Conclusions

A method is proposed, implemented and evaluated for performing real-time computer vision based hand pose extraction with a single RGB camera and could run on current consumer level computing power. The implementation shows that the system is usable in a real-time human computer interaction setting, e.g. used for creating and manipulating sound. It is assumed that in the input video stream only one person is visible, this person is wearing clothing with long sleeves, and no skin-like colors are in the image.

The implementation uses a histogram based probabilistic skin color distribution which is based upon the users face which is found by a haar classifier. This is a robust way for face detection and the detection speed is reasonable if performed every 10th frame. The face detection will give unpredictable results when multiple faces are visible in the input image.

The skin color distribution is an effective and fast way to localize body parts in the image, but it is required that no skin-like colors are in the input image. Going from the probabilistic domain to the labeled binary domain can be controlled with a threshold parameter. This parameter can be automatically adjusted, but a carefully chosen fixed value is enough for most cases.

Using heuristics for blob labeling is a robust method, but fails when the hands cross their position in the x direction. Keeping the labels of a blob fixed after the first detection could solve this problem, but introduce many new problems like what to do when the blobs are gone for sequence of frames and decide at what moment a blob should get which label.

Using HOG features for image representation gave very good results, much better than SURF features. This is probably caused by the improper use of SURF, where the use of the keypoint detection phase seems essential. In the experiments HOG and SURF where both used with a dense grid of keypoints.

k-NN can be used as a classifier, but becomes slow when a lot of training samples (+2000) are used. SVM is a better alternative, which yields better performance in classification time an accuracy. As a kernel for SVM the χ^2 and RBF kernel accomplish similar results, but the RBF kernel requires the optimization of one extra parameter γ. This extra parameter requires more work during the training phase, so the χ^2 kernel is preferred.

Even though the accuracy of the classification system was high in some benchmarks, the choice of the Curwen hand poses as 'language' wasn't a good one. A smaller set with hand poses that are less similar and have a large visible surface would perform better.

The experiment results show that the system is not person independent. Incorporating training data gathered from the eventual user will improve the classification performance. It is not yet known if the performance will increase with more training data. Also the best results are yielded when a simple background is used.

6.1 Future Work

The performance of the system could be evaluated when configured with less hand poses that are less similar. Also the current dataset is too small to analyze the impact of more or less train data. A bigger dataset could be constructed and the impact on person independence could be studied.

Less scientific but also interesting is the things that could be done with the implementation of Sonic Gesture. It currently uses the k-NN classifier, but the experiments show that a SVM classifier with the χ^2 kernel perform much better. Currently it doesn't use automatic threshold adjusting, something that can be implemented also. The capture mode is currently not very user friendly, something that would help gathering train data when improved.

Bibliography

V. Athitsos and S. Sclaroff. Estimating 3d hand pose from a cluttered image. volume 2, pages II – 432–9 vol.2, jun. 2003.

Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *In ECCV*, pages 404–417. 2006.

Richard Bowden, David Windridge, Timor Kadir, Andrew Zisserman, and Michael Brady. A linguistic feature vector for the visual interpretation of sign language. *Computer Vision - ECCV 2004*, pages 390–401, 2004.

Gary R. Bradski. Computer vision face tracking for use in a perceptual user interface, 1998.

P. Buehler, A. Zisserman, and M. Everingham. Learning sign language by watching tv (using weakly aligned subtitles). *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 0:2961–2968, 2009.

J. Cai and A. Goshtasby. Detecting human faces in color images. *Image and Vision Computing*, 18(1):63 – 75, 1999. ISSN 0262-8856.

Olivier Chapelle, Patrick Haffner, and Vladimir Vapnik. Svms for histogram-based image classification, 1999.

Chieh chih Wang and Ko chih Wang. Hand posture recognition using adaboost with sift for human robot interaction, 2007.

Choksy and Lois. *The Kodály Method I.* Prentice Hall, Englewood Cliffs, 1999. ISBN 0139491651.

Ira Cohen, Nicu Sebe, Ashutosh Garg, Lawrence S. Chen, and Thomas S. Huang. Facial expression recognition from video sequences: temporal and static modeling. *Computer Vision and Image Understanding*, 91(1-2):160 – 187, 2003. ISSN 1077-3142. Special Issue on Face Recognition.

Helen Cooper and Richard Bowden. Large lexicon detection of sign language. In *ICCV, Workshop Human Comp. Inter*, 2007.

Navneet Dalal, Bill Triggs, and Cordelia Schmid. Human detection using oriented histograms of flow and appearance. *Computer Vision ECCV 2006*, pages 428–441, 2006.

Martin de La Gorce and Nikos Paragios. A variational approach to monocular hand-pose estimation. *Computer Vision and Image Understanding*, 114(3):363 – 372, 2010. ISSN 1077-3142.

Ali Erol, George Bebis, Mircea Nicolescu, Richard Boyle, and Xander Twombly. Vision-based hand pose estimation: A review. *Comput. Vis. Image Underst.*, 108(1-2):52–73, 2007. ISSN 1077-3142.

Vittorio Ferrari, Manuel Marín-Jiménez, and Andrew Zisserman. 2d human pose estimation in tv shows. *Lecture Notes in Computer Science*, 5064, 2008.

C. W. Hsu, C. C. Chang, and C. J. Lin. A practical guide to support vector classification. Technical report, Taipei, 2003.

A.J. Hunt and A.W. Black. Unit selection in a concatenative speech synthesis system using a large speech database. volume 1, pages 373 –376 vol. 1, may. 1996.

I. Jolliffe. Principal component analysis. 2002.

Michael J. Jones and James M. Rehg. Statistical color models with application to skin detection. In *International Journal of Computer Vision*, pages 274–280, 1999.

S. Mitra and T. Acharya. Gesture recognition: A survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 37(3):311 –324, may. 2007. ISSN 1094-6977.

Zhenyao Mo and Ulrich Neumann. Real-time hand pose recognition using low-resolution depth images. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1499–1505, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2597-0.

Thomas B. Moeslund, Adrian Hilton, and Volker Krüger. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding*, 104(2-3):90 – 126, 2006. ISSN 1077-3142. Special Issue on Modeling People: Vision-based understanding of a person's shape, appearance, movement and behaviour.

A.C. Murillo, J. J. Guerrero, and C. Sagues. Surf features for efficient robot localization with omnidirectional images. pages 3901 –3907, apr. 2007.

Ronald Poppe. Vision-based human motion analysis: An overview. *Com-*

puter Vision and Image Understanding, 108(1-2):4 – 18, 2007. ISSN 1077-3142. Special Issue on Vision for Human-Computer Interaction.

L. Rabiner and B.H. Juang. *Fundamentals of speech recognition*. Prentice hall Englewood Cliffs, New Jersey, 1993.

T. Starner, A. J. Weaver, and Pentland. Real-time american sign language recognition using desk and wearable computer based video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1371–1375, 1998.

Björn Stenger. Template-based hand pose recognition using multiple cues. In *In Asian Conference on Computer Vision*, pages 551–560, 2006.

S. Suzuki and K. Abe. Topological structural analysis of digitized binary images by border following. *CVGIP*, 30(1):32–46, April 1985.

Christoffer Valgren and Achim J. Lilienthal. Sift, surf & seasons: Appearance-based long-term localization in outdoor environments. *Robotics and Autonomous Systems*, 58(2):149 – 156, 2010. ISSN 0921-8890. Selected papers from the 2007 European Conference on Mobile Robots (ECMR '07).

Michael Van den Bergh, Esther Koller-Meier, and Luc Van Gool. Real-time body pose recognition using 2d or 3d haarlets. *International Journal of Computer Vision*, 83:72–84, 2009. ISSN 0920-5691. 10.1007/s11263-009-0218-0.

Vladimir Vezhnevets, Vassili Sazonov, and Alla Andreeva. A survey on pixel-based skin color detection techniques. In *Proceedings of the GraphiCon 2003*, pages 85–92, 2003.

Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. pages 511–518, 2001.

Paul Viola and Michael Jones. Robust real-time face detection. *Int. J. Comput. Vision*, 57(2):137–154, 2004. ISSN 0920-5691.

C. Vogler and D. Metaxas. Parallel hidden markov models for american sign language recognition. volume 1, pages 116 –122 vol.1, 1999.

Robert Y. Wang and Jovan Popović. Real-time hand-tracking with a color glove. *ACM Trans. Graph.*, 28(3):1–8, 2009. ISSN 0730-0301.

Tomoki Watanabe, Satoshi Ito, and Kentaro Yokoi. Co-occurrence histograms of oriented gradients for pedestrian detection. *Advances in Image and Video Technology*, pages 37–47, 2009.

G. Welch and G. Bishop. An introduction to the Kalman filter. *University of North Carolina at Chapel Hill, Chapel Hill, NC*, 1995.

Yingen Xiong, Bing Fang, and Francis Quek. Extraction of hand gestures with adaptive skin color models and its applications to meeting analysis. pages 647 –651, dec. 2006.

Xiao Yi, Shengfeng Qin, and Jinsheng Kang. Generating 3d architectural models based on hand motion and gesture. *Comput. Ind.*, 60(9):677–685, 2009. ISSN 0166-3615.

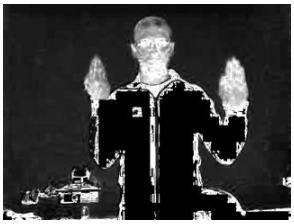
J. Zhang, S. Lazebnik, and C. Schmid. Local features and kernels for classification of texture and object categories: a comprehensive study. *International Journal of Computer Vision*, 73:2007, 2007.

Appendix A

Images

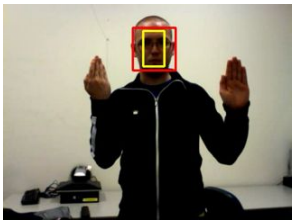
(a) input frame

(b) hue channel

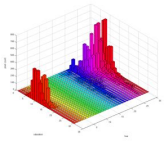


(c) saturation channel

(d) value channel



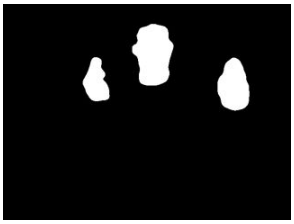
(e) face detection



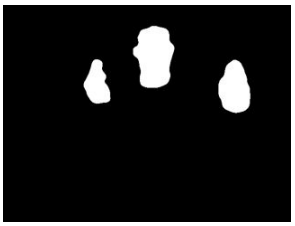
(f) Face color histogram

(g) Backprojection

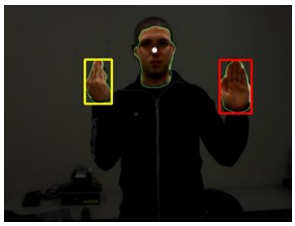
(h) Gaussian smooth



(i) Thresholded binary image

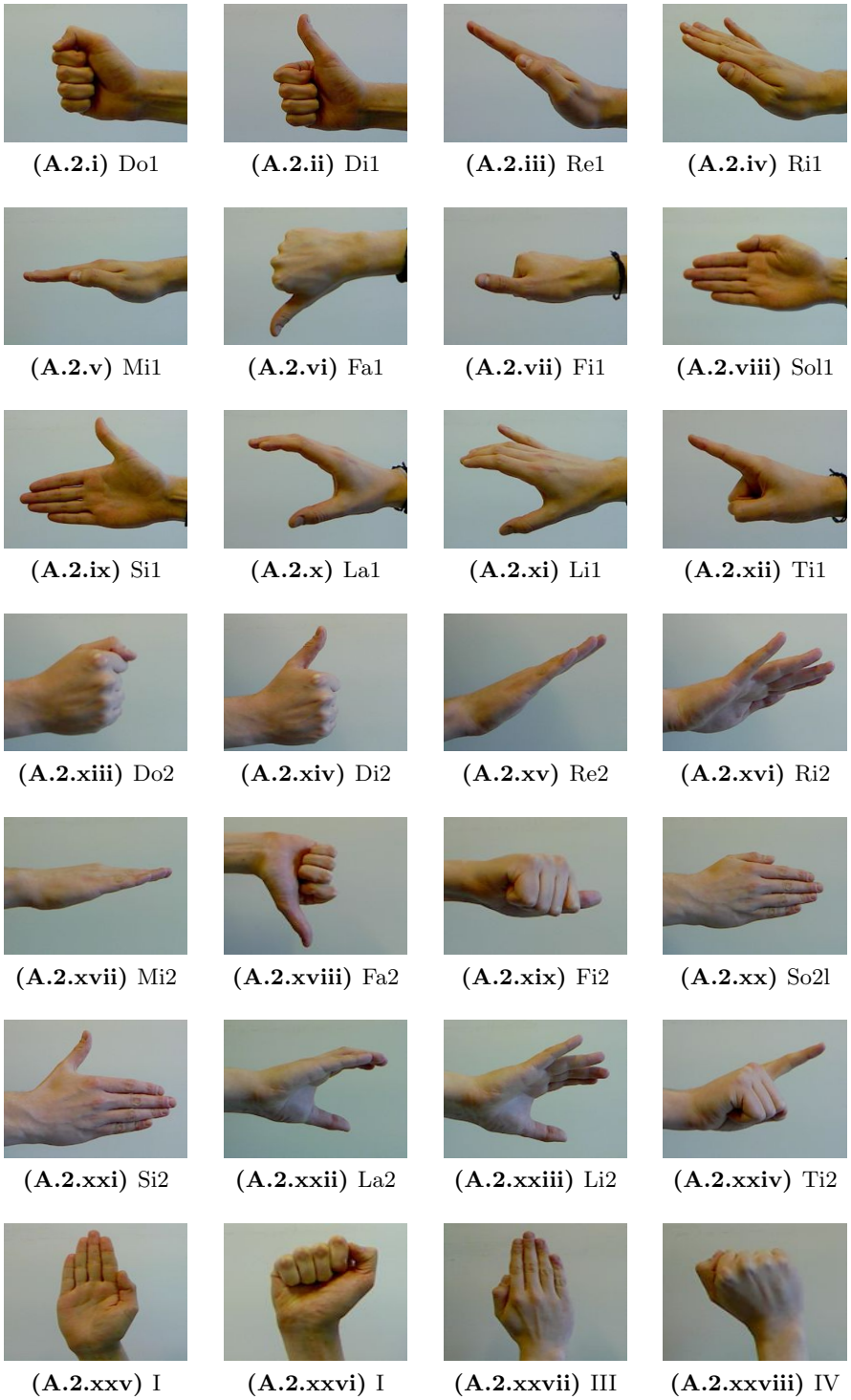


(j) Morpholoical close



(k) Blob labeling

Figure A.1: The complete hand pose detection pipeline



(A.2.i) Do1 **(A.2.ii)** Di1 **(A.2.iii)** Re1 **(A.2.iv)** Ri1

(A.2.v) Mi1 **(A.2.vi)** Fa1 **(A.2.vii)** Fi1 **(A.2.viii)** Sol1

(A.2.ix) Si1 **(A.2.x)** La1 **(A.2.xi)** Li1 **(A.2.xii)** Ti1

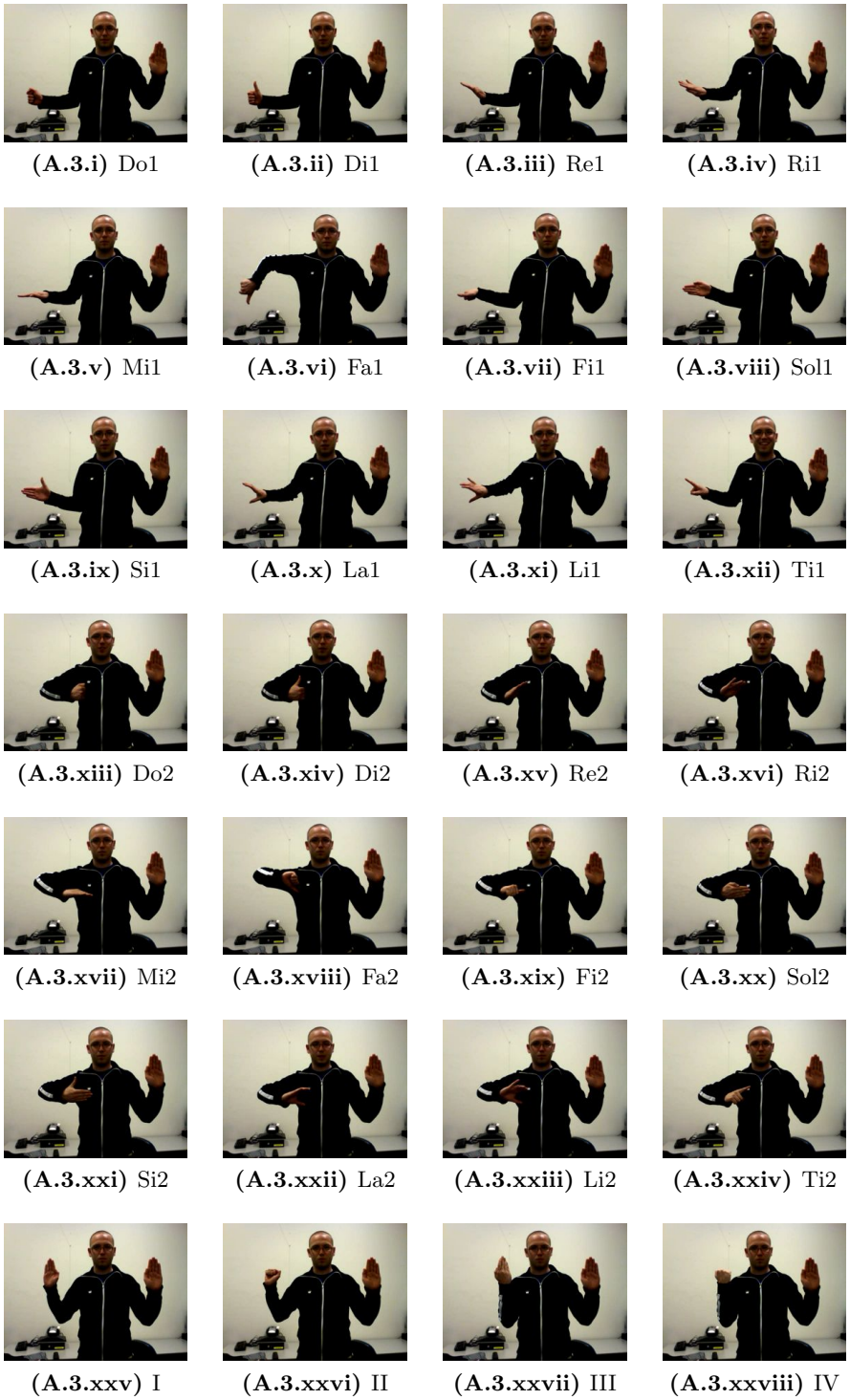
(A.2.xiii) Do2 **(A.2.xiv)** Di2 **(A.2.xv)** Re2 **(A.2.xvi)** Ri2

(A.2.xvii) Mi2 **(A.2.xviii)** Fa2 **(A.2.xix)** Fi2 **(A.2.xx)** So2l

(A.2.xxi) Si2 **(A.2.xxii)** La2 **(A.2.xxiii)** Li2 **(A.2.xxiv)** Ti2

(A.2.xxv) I **(A.2.xxvi)** I **(A.2.xxvii)** III **(A.2.xxviii)** IV

Figure A.2: The example hand poses as shown to the test subjects



(A.3.i) Do1 **(A.3.ii)** Di1 **(A.3.iii)** Re1 **(A.3.iv)** Ri1

(A.3.v) Mi1 **(A.3.vi)** Fa1 **(A.3.vii)** Fi1 **(A.3.viii)** Sol1

(A.3.ix) Si1 **(A.3.x)** La1 **(A.3.xi)** Li1 **(A.3.xii)** Ti1

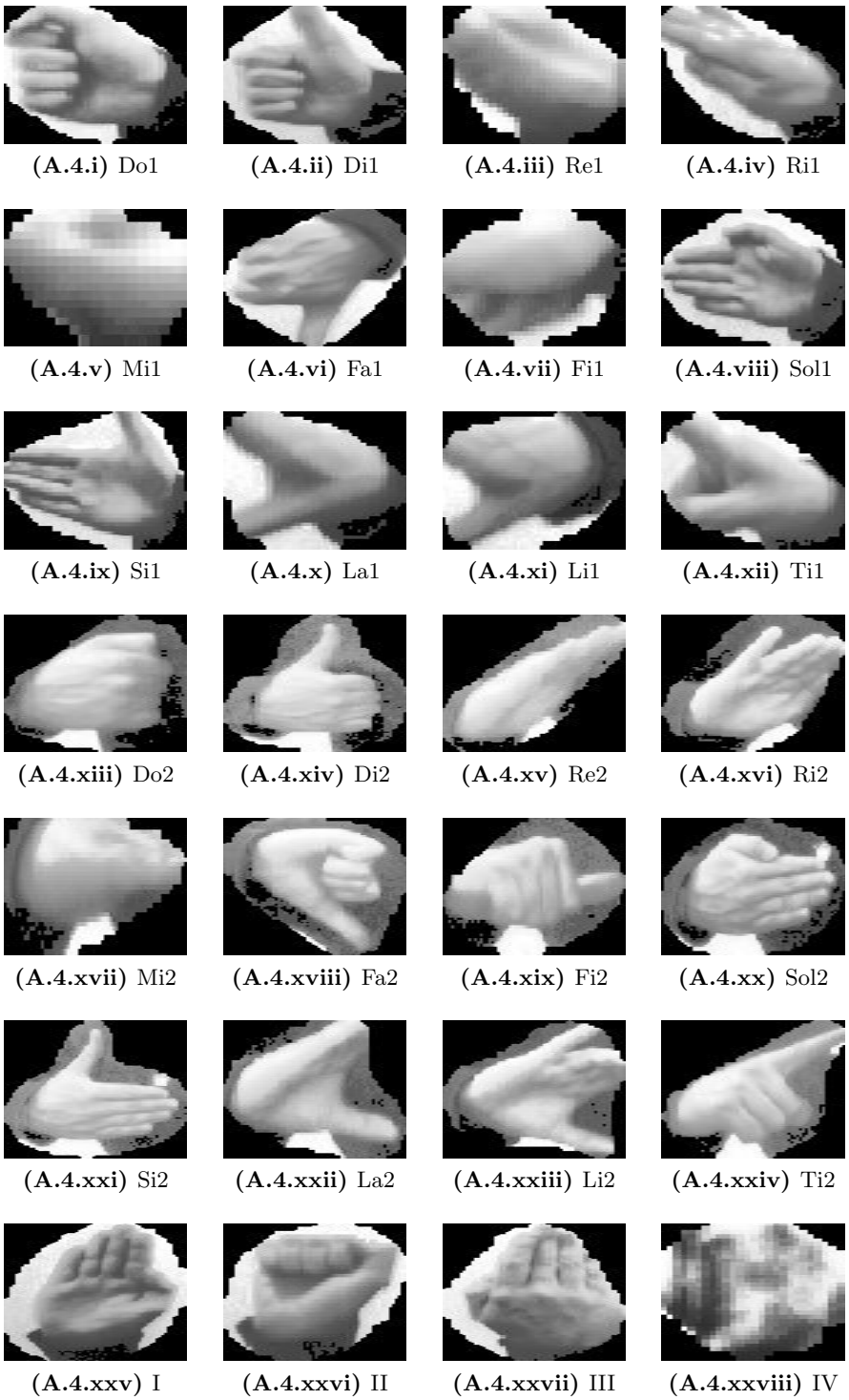
(A.3.xiii) Do2 **(A.3.xiv)** Di2 **(A.3.xv)** Re2 **(A.3.xvi)** Ri2

(A.3.xvii) Mi2 **(A.3.xviii)** Fa2 **(A.3.xix)** Fi2 **(A.3.xx)** Sol2

(A.3.xxi) Si2 **(A.3.xxii)** La2 **(A.3.xxiii)** Li2 **(A.3.xxiv)** Ti2

(A.3.xxv) I **(A.3.xxvi)** II **(A.3.xxvii)** III **(A.3.xxviii)** IV

Figure A.3: The labeled frames of a movie from the dataset



(A.4.i) Do1 **(A.4.ii)** Di1 **(A.4.iii)** Re1 **(A.4.iv)** Ri1

(A.4.v) Mi1 **(A.4.vi)** Fa1 **(A.4.vii)** Fi1 **(A.4.viii)** Sol1

(A.4.ix) Si1 **(A.4.x)** La1 **(A.4.xi)** Li1 **(A.4.xii)** Ti1

(A.4.xiii) Do2 **(A.4.xiv)** Di2 **(A.4.xv)** Re2 **(A.4.xvi)** Ri2

(A.4.xvii) Mi2 **(A.4.xviii)** Fa2 **(A.4.xix)** Fi2 **(A.4.xx)** Sol2

(A.4.xxi) Si2 **(A.4.xxii)** La2 **(A.4.xxiii)** Li2 **(A.4.xxiv)** Ti2

(A.4.xxv) I **(A.4.xxvi)** II **(A.4.xxvii)** III **(A.4.xxviii)** IV

Figure A.4: The hand windows, used for feature extraction