University of Freiburg
Faculty of Applied Sciences
Department of Computer Science
Junior Research Group Humanoid Robots

Master's Thesis

Recognition of Human Gestures using Monocular Vision

Author:	Tobias Axenbeck
Submitted on:	25.01.2008
First Referee:	Dr Sven Behnke
Second Referee:	Prof Dr Wolfram Burgard
Supervisor:	Dr Maren Bennewitz

Abstract

Robots coexisting with humans in everyday environments should be able to interact with them in an intuitive way. This requires that the robots are able to recognize typical gestures performed by humans such as head shaking/nodding, waving, or pointing gestures. In this thesis, we present a system that is able to spot and recognize complex gestures from monocular image sequences. To estimate their position and to represent people, we detect and track their faces and hands using classifiers trained with AdaBoost. WE use few expressive features extracted out of this compact representation as input to hidden Markov models (HMMs). We segment example gestures into distinct phases and train HMMs for each phase separately. To construct a HMM composed of the individual-phase HMMs, we define regular grammars. Once a specific phase is recognized, we estimate the parameter of gestures such as the pointing target.

As we demonstrate in our experiments, our method is able to robustly locate and track hands, which is a difficult task since they can take a large number of substantially different shapes. Furthermore, we show that our system is able to reliably spot and recognize gestures in real time. Additional experiments illustrate that parameters of gestures can be accurately estimated.

dull day...

Samuel Beckett, 1936 in Hamburg

Zusammenfassung

Roboter, welche mit Menschen in Alltagsumgebungen koexistieren, sollten in der Lage sein, mit ihnen auf intuitive Art und Weise zu interagieren. Dies setzt voraus, dass der Roboter fähig ist, typische menschliche Gesten wie Kopfschütteln, -nicken, Winken oder Zeigegesten zu erkennen. In dieser Arbeit stellen wir ein System vor, welches imstande ist, komplexe Gesten in monoskopischen Bildsequenzen zu entdecken und zu erkennen. Um die Position von Menschen im Bild zu schätzen, detektieren und verfolgen wir ihre Haende und Gesichter unter Verwendung von Klassifikatoren, welche mit Adaboost trainiert wurden. Aus dieser kompakten Darstellung menschlicher (Arm)-Bewegungen extrahieren wir eine kleine, ausdrucksstarke Merkmalsmenge, welche als Eingabe für Hidden Markov Modelle (HMMs) dient. Komplexe Gesten unterteilen wir in verschiedene Phasen und trainieren HMMs für jede einzelne Phase. Um das finale Erkennungs-HMM zu kontruieren, definieren wir reguläre Grammatiken. Sobald eine spezifische Phase erkannt ist, schätzen wir eventuelle Parameter der Geste, wie zum Beispiel das Zeigeziel.

Wie wir in unseren Experimenten zeigen werden, ist unsere Methode in der Lage, Hände robust zu lokalisieren und zu verfolgen, was schwierig ist, da Hände unzählige verschiedene Formen annehmen können. Des weiteren zeigen wir, dass unser System in der Lage ist Gesten verlässlich und in Echtzeit wahrzunehmen und zu erkennen. Zusätzliche Experimente veranschaulichen, dass Parameter von Gesten akkurat geschätzt werden können.

dull day…

Samuel Beckett, 1936 in Hamburg

Acknowledgments

I would like to thank all those who supported me. Especially, I wish to thank Dr Maren Bennewitz who supervised this work. She introduced me to scientific working and was always willing to answer my every question. Special thanks also to Dr Sven Behnke for making this work possible and for all his advice and suggestions. I would further like to thank Dr Cyrill Stachniss for his implementation of the Hungarian algorithm and his help in creating the great filled-curve figures with gnuplot. My thanks to Theodora Vatahska for her work on head pose estimation. Of course I must not forget my patient "models" Eva Axenbeck, Benjamin Riedinger, Ann-Kristin Hintertupfingen, Peer Fechner, and Maria Kopp. They all spared neither trouble nor expense to make it to the recording sessions. Many thanks to them. For proofreading this work, I'm grateful to Felix Faber, Hauke Strasdat and Maria Kopp.

Contents

1 Introduction

Gestures play a vital role within interpersonal communication where we use them to support or even replace speech. It is therefore of utmost interest to be able to perceive and interpret gestures performed by the conversational partner. In the beginnings of research in gesture recognition, only cumbersome special devices such as bodysuits, markers, or data gloves could be used to obtain the needed data. Nowadays, the development of visual gesture recognition systems is often part of the effort to achieve more efficient, interactive and intelligent interfacing with computers.

This coincides with the ambitions within the *Human Computer Interaction* (HCI) field. HCI investigates modes of interfacing beyond traditional devices such as keyboards, mice, or displays, to further improve the usability and efficiency of modern computers. The aim is to provide the same intuitiveness and naturalness as in interpersonal interactions. These are primarily governed by the auditory and visual modality. Whilst the auditory modality is used to communicate information directly, additional important information can be deduced by the visual perception of our conversational partner. This most notably includes information conveyed by the use of gestures.

Humanoid robots constitute an excellent testbed for the investigation of such intuitive communication strategies as they inherently provide the possibility to embody human interaction techniques. It is this very reason that motivated the construction of our multi-modal communication robot Fritz, shown in Figure 1.1. For a thorough description of our robot refer to [Bennewitz et al. 2007]. In order to successfully perceive and interact with the environment, the robot relies on the input from auditory and visual sensors. Apart from speech synthesis and recognition, it uses natural modalities such as eye gaze, facial expressions, and gestures to generate human-like behaviour and

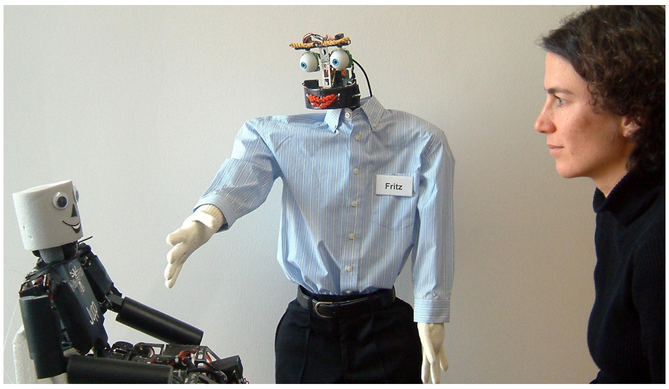


Fig. 1.1: Our communication robot Fritz drawing the attention of a user to an exhibit by pointing at it.

to interact with people in an intuitive way.

Through the animated mouth and eyebrows, Fritz can change its facial expression to show emotions, such as anger or joy. With head and arms it performs gestures such as waving, nodding, and draws the attention to objects of interest by pointing and looking toward these.

Currently, there is an asymmetry between Fritz' ability of generating gestures, and to sense them visually, in that it cannot recognize and interpret them. Up to now, its visual perception ability is limited to the detection of faces. In this thesis, we will present a system which provides the ability to recognize common human gestures such as waving, pointing, or nodding.

In the following, a brief excursion on the nature of human gestures is given as well as a containment of gestures we want to be able to recognize within the gesture space.

1.1 On Gestures

Gestures can exist alone or involve external objects. Empty-handed, we wave, gesticulate, show emblems such as "thumbs up" or may even make use of more formal sign languages. With objects, we have a broad range of gestures that are almost universal, including pointing at objects, touching, or moving objects, deforming object shape, or handing objects to others. This suggests that gestures can be classified according to their functional roles. In [Crowley & Coutaz 1995] three such roles are distinguished:

Semiotic gestures are used to communicate meaningful information based on convention and resulting of common cultural knowledge.

Ergodic gestures correspond to the creation and manipulation of objects.

Epistemic gestures are used to learn from the environment through tactile or haptic exploration.

In this thesis we are primarily interested in how gestures can be used to communicate with a robot, so we will be mostly concerned with empty-handed semiotic gestures. These can further be categorized according to their functionality. Rime & Schiaratura [1991] propose the following gesture taxonomy:

Symbolic gestures are gestures that, within each culture, have come to have a single meaning. An emblem such as the victory sign is one such example, however sign language gestures also fall into this category.

Deictic gestures are the types of gestures most often seen in HCI and are the gestures of pointing, or otherwise directing the listeners attention to specific events or objects in the environment.

Iconic gestures are these gestures which are used to convey information about the size, shape or orientation of the object in question.

Finally, we observe that gestures can be either static, when one assumes a certain pose or configuration, or dynamic, defined by movements. Emblems and sign languages are primarily of the former kind whereas the others involve motion, mainly of the arms. According to [McNeill 1992], this motion can be separated into three phases: *preparation*, *stroke* (or *hold*), and *retraction*. The preparation and retraction elements consist of moving the arms to and from the rest position, to and from the start and end of the stroke. Semantically relevant, however, is only the stroke phase. It is therefore of special importance to catch this phase precisely, as possible parameters such as the pointing target can only be estimated during this phase.

1.2 Goals and Contributions

The objective of this thesis is to provide the ability to recognize complex arm and head gestures. A system is developed which tackles the problem in a sequence of steps as illustrated in Figure 1.2. It first performs appearance-based face and hand detection and associates them with previous detections, thus keeping track of people. The obtained motion trajectories are then analyzed for the occurrence of meaningful patterns.

The characteristics of both the robotic platform and the application scenario are considered, which imposes the following requirements and constraints:

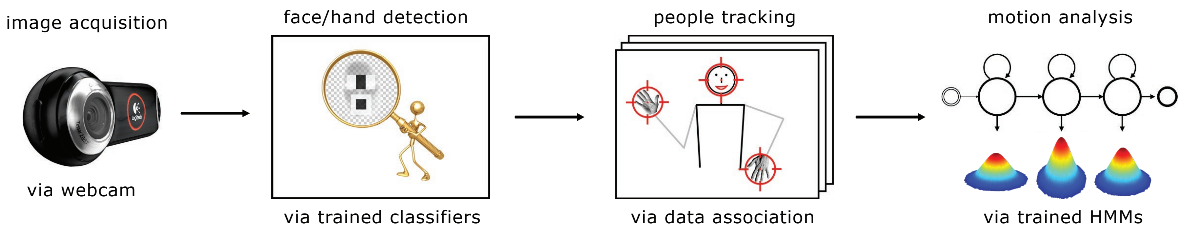


Fig. 1.2: The sequence of steps performed by our gesture recognition system. After image acquisition from standard webcams, we use previously trained face and hand classifiers to detect them in each individual image. These observations are associated with previous ones and thus tracking is performed. The continuous observation sequence is transformed into descriptive features and fed into previously trained HMM to classify it accordingly either as a specific gesture or as non-gesture.

- Mono-vision: Only a single low-cost camera (standard webcam) is used. This implies monocular images, limited resolution and frame rates.
- Real-time ability: Despite the limited processing power, the responsiveness of the system is crucial, i.e., gestures should be recognized in an online fashion with low latency. This restricts the use of time-consuming techniques.
- Non-static background: As the robot is able to move itself, the resulting camera motion prohibits the use of simple segmentation or motion detection techniques such as image differencing.
- Cluttered background: this complicates the segmentation step and leads to ambiguities as well as noisy measurements that have to be dealt with accordingly.

The system presented in this thesis is able to deal with these problems. In particular, we combine the object detection framework of Viola & Jones [2001] which works on grayscale images with the skin color cue to support and speed up the hand detection process. Hand classifiers which are able to distinguish both hands from non-hands, and right hands from left hands, are constructed. We further propose to track hands indirectly using Kalman filters and the Hungarian method for solving the data association problem. Here, we take into account the uncertainty in the hand detection process. To robustly establish a compact human body model with face and hands, we consider spatial dependencies as well as the laterality of the detected hands. From the obtained 2D motion trajectories of face and hands, a set of expressive features is selected and hidden Markov models (HMMs) are used to model gesture-segments according to each phase. The trained individual-phase HMMs are aligned into a single compound HMM according to a regular grammar, for both mono- and bi-manual gestures as well as for head gestures. The feature stream is continuously interpreted for the occurrence of the individual phases using Viterbi path alignment. In practical experiments, we demonstrate that we can robustly spot and recognize gestures in real-time. Furthermore, we show that we can estimate the parameters of deictic and iconic gestures accurately during the hold phase of the respective gesture.

1.3 Thesis Outline

The remainder of this thesis is organized in accordance with the individual steps performed by our recognition system, as illustrated in Figure 1.2.

The next chapter reviews previous publications related to our work. In Chapter 3, the detection of faces and in particular of hands is described, as well as training the appropriate classifier. Chapter 4 deals with the tracking step, that is, how correspondences between successive frames are determined and how human body models are established. Chapter 5 introduces briefly the head pose estimation system which resulted from a previous master's thesis as well as extensions which were necessary for our work. Feature extraction, the modeling of gestures with HMMs, their training and, finally, their application to recognize of gestures is presented in Chapter 6. At last, we present experimental results demonstrating the applicability of our system in various real-world scenarios in Chapter 7 and conclude with a brief summary and discussion of future work in Chapter 8.

2
Related Work

Many researchers have investigated the problem of gesture recognition. Each approach tackles the problem domain differently and focuses on different aspects. In this chapter, we give an overview over existing techniques and draw comparisons to our work.

One of the earliest work addressing gesture recognition is due to [Yamato, Ohya, & Ishii 1992]. In this approach, discrete HMMs and a sequence of vector-quantized (VQ) labels are used to recognize six classes of tennis strokes. The labels are obtained by processing a monocular image sequence applying background subtraction to extract the moving objects (which are assumed to be humans), and binarization of the moving objects to generate blobs. These blobs correspond to the poses of the human. The features are the number of object pixels which are then vector-quantized, such that the image sequence becomes a sequence of VQ-labels. This sequence is subsequently processed by the discrete HMMs.

Campbell et al. [1996] explicitly locate head and hands via a commercial stereo-vision system which also performs tracking over time and provides the 3D trajectories. These are used to recognize 18 different Tai Chi gestures by means of HMMs. Focus is placed on the evaluation of different feature combinations fed to a fixed HMM, e.g., using absolute position data, hand positions relative to the head, velocities, and angles. They found that relative position data significantly outperformed absolute position data and incorporating velocities yields the best results with a recognition rate of over 90%.

In [Rigoll, Kosmala, & Eickeler 1998] real-time gesture recognition of 24 isolated gestures is presented using very low resolution gray-scale monocular images. For data reduction differences of successive images are computed which extracts the moving body parts. These regions are described using image moments and other statistics to form a 7-dimensional feature vector that is fed into either a linear left-to-right or a cyclic HMM, depending on the nature of the gesture in question. Recognition rates over 90% are obtained at 15 fps.

An interesting extension to HMMs was introduced by Wilson & Bobick [1999]; Wilson [2000]. To extract the information carried by iconic and deictic gestures the corresponding parameter is explicitly integrated into a *Parametric* HMM (PHMM) using a modified version of the Baum-Welch Expectation-Maximization algorithm (see Chapter 6). Using this PHMM, they are able to deduce the parameter with high accuracy and achieve higher recognition rates than with standard HMMs. The 3D position data of head and hands was assumed to be available (obtained using a stereo-vision tracking system). As the parameter of gestures affects all states of the PHMM, it can only be derived after processing the full observation sequence, thus no online recognition is possible.

The work by Nickel, Seemann, & Stiefelhagen [2004] focuses on pointing gesture recognition. They use a stereo camera and heuristics to locate human heads. From these regions, histograms are computed to locate other skin-colored parts as candidates for hands. Tracking hand candidates is performed using three probabilistic scores that take into account 1) skin-ishness (*observation score* $P(O_t)$, 2) the likelihood of the currently assumed posture (*posture score* $P(s_t)$) and 3) the movement between two successive frames (*transition score* $P(s_t|s_{t-1})$. The highest ranking hypotheses are then assigned to the head as left and right hand. For recognition of a pointing gesture, three individual HMMs are trained, one for the preparation, one for the hold (stroke), and one for the retraction phase respectively. Rather than using the tracked head and hand positions directly, hand coordinates are transformed into a cylindric coordinate system with the head being the origin. The radius r, the angle Θ, and the velocity of the y coordinate constitute the features continuously fed into the HMMs. As soon as the hold phase is recognized, the pointing target is estimated using the 3D positions of head and hands. Around 90% for both pointing gesture recognition and target identification is obtained. In contrast to this system, ours is not restricted to one single gesture. Instead, we are able to recognize a set of different arm gestures. Furthermore, they apply a time-consuming analysis to estimate the end of a gesture phase.

Just, Bernier, & Marcel [2004] consider the problem of recognizing mono- and bimanual gestures given 3D trajectories of blobs using Input/Output HMMs (IOHMMs). Their feature vector consists of 12 features and, accordingly, a high number of training sequences is needed. IOHMMs provide no means of recognizing parametric gestures.

Brethes et al. [2004] present a face and hand tracking system for gesture recognition in real-time combining shape and color cues. With a previously trained skin color model corresponding region segmentation is done using the I1I2I3 color space with high accuracy. Face detection is performed using the face detection method by Viola & Jones. Additionally, face recognition is implemented using PCA and Harris interest points with a reported recognition rate of 90%. Head and hands are tracked using the Condensation algorithm. Reweighing the particles is based on matching the shape with pre-defined templates such that the most likely hand configuration hypothesis is known too. In this way four different hand poses can be distinguished which is used to trigger certain commands. Clearly, the approach has its limitations as only very few and distinct hand templates are considered and only the static part of a gesture is analysed.

An approach to tackle visual recognition of typical office activities is presented in [Montero & Sucar 2004], together with a thorough evaluation of different HMM configurations and feature combinations. Using a ceiling-mounted camera, only single-hand detection and tracking is conducted based on back-projection applying a given color histogram and simple heuristics to distinguish heads from hands. The obtained 2D trajectory is transformed into different feature sets as in [Yoon et al. 1999] and vector-quantized into a varying number of symbols. They found magnitude and orientation in polar coordinates together with 64 discrete symbols and a 10-states HMM to perform best with a recognition rate of 97%.

The problem of whole body gesture recognition is addressed in [Lee 2006]. Using depth images against simple (white) background a 40 DOF human limb model is reconstructed as a linear combination of stored prototypes. The obtained angles of 13 selected joints form the feature vectors which, after clustering with Gaussian mixture models (GMMs), are the input to the HMMs. A similar approach is presented by Yang, Park, & Lee [2006]. They use angular relations between a dozen body parts in 3D as features which are clustered using GMMs.

Although HMMs are the most successful and therefore most often applied technique to model the spatio-temporal pattern of gestures a few other means have been applied. As a gesture can be modeled as a sequence of states in a spatio-temporal configuration space, *finite state machines* (FSM) can be employed to recognize them. In [Hong, Huang, & Turk 2000], each gesture is defined to be an ordered sequence of states, using spatial clustering and temporal alignment. The spatial information is first learned from a number of training images of the gestures. This information is used to build FSMs corresponding to each gesture. Also minimum and maximum time-spans need to be defined. No recognition rates are reported.

Also *Neural Networks* (NN) have been applied to gesture recognition, as in [Yang & Ahuja 1999], to recognize American Sign Language (ASL) gestures. Multiscale motion segmentation is performed by means of region matching between two consecutive frames. Subsequently, only those regions containing skin-color are considered. These are then tried to fit into either an ellipse or rectangle to distinguish basic hand shapes (open hand (ellipse) vs fist (rectangle)) and head (large ellipse) from hands. A Time Delay NN with two hidden layers is employed to classify the motion of hand regions as a particular gesture (sign) according to ASL. Using this approach, a recognition rate of 93% can be achieved.

In [Richarz et al. 2006] NNs are used for pointing gesture recognition. Gabor-filtered input images are fed into a cascade of NNs to estimate the corresponding pointing direction. To achieve this, first face detection using the object detection framework by Viola & Jones [2001] is employed. Centered around the face, a region of interest (ROI) is constructed which is assumed to contain the upper body plus arms. The region is scaled to a fixed size, Gabor-filtered and input into the first stage of the NN cascade which outputs the pointing side (left/right). The ROI is adjusted accordingly and serves as input to the second stage which performs radius and angle estimation of the target. A 50% rate of correct radius and angle was achieved which still outperforms humans. In a later work ([Martin, Steege, & Gross 2007], several improvements have been conducted: background subtraction, Gabor jets of distinctive points in the ROI to achieve real-time ability. Other neural function approximators were evaluated (Neural Gas, Self-Organizing Map, Local Linear Map), yet the already used NN was found to perform best.

In contrast to the approaches discussed above, using our techniques, we can reliably recognize a set of typical complex, dynamic arm and head gestures and estimate parameters of gestures from monocular images. We do not assume a known or static background nor do we apply simple heuristics to detect head and hands. Instead, we follow an appearance-based approach Viola & Jones [2001] and train classifiers that can robustly distinguish the objects of interest from the background.

Employing such a pattern recognition techniques rather than relying on heuristics for face or hand detection is a great step toward more reliable and robust detection of humans. Often being costly, recent advancement allow for real-time processing of such techniques and consequently are being applied, as in our work. The object detection method by Viola & Jones [2001], originally applied in the face detection domain, has been adopted to hand detection by only few research groups.

Kolsch & Turk [2004] concentrate on few distinctive hand shapes which are frequency-analyzed for good class separation ability. This is motivated by the fact that training such a classifier is very time-consuming (several days on a decent computer) and training a general hand classifier is impossible as there are infinitely many possible hand configurations. For each of the most promising hand shapes an individual classifier was trained, both with the standard feature set and with an additional, more expressive rectangular feature (see Section 3.2), although it turned out that it did not improve the detection rate. The best achieved detection rate was as high as 92% with a false positive rate of $1.01 \cdot 10^{-8}$.

Extending the feature set is also the focus of the work by Barczak [2005], where they provide a method to rotate the rectangular features by an arbitrary angle, thus achieving rotational invariance. To yield significant better results, however, large minimum search areas are needed, which limits the usability of this approach.

In [Chen, Georganas, & Petriu 2007] for four hand postures (fist, little finger, index and middle finger, open hand) a separate hand classifier is trained on samples against white background and a detection rate of over 90% is achieved. A context-free grammar is then used to describe a sequence of hand postures as gesture and thus continuously classify the posture stream as such. A more unconstrained approach is pursued by Ong & Bowden [2004]. In their work the authors train a two-layer classifier tree for hand shape detection where a database of hand images is clustered into sets of similar hands according to a distance metric based on shape context. The first layer is trained with all images for general hand detection whilst branches in the second layer are trained on a single hand shape class. A very high detection rate is achieved, however, no false detection rates are given. Also the hand images used for training and testing are both against simple and similar background.

In contrast to these methods, our system is able to detect and track hands with arbitrary shapes, even under difficult background and lighting conditions. Since the classifiers work on grayscale images, we use a skin color cue to support and speed-up the process of hand detection.

Finally, there are also a few surveys which review further work in this area. Somewhat dated but very exhaustive is [Joseph J. LaViola 1999], a very recent survey can be found in [Mitra & Acharya 2007].

3 Detecting Faces and Hands

Both face and hand detection belong to the general problem of (visual) pattern recognition and have been subject to research within the computer vision community long since. In particular, *face detection* assumes an outstanding position. It does not only serve as a first-rate cue to human detection in general but also is a prerequisite for many subsequent and closely related problems such as face recognition, face authentication or facial expression recognition, to name a few. Hence, a vast body of literature on this topic exists, a recent survey of which is given by Yang, Kriegman, & Ahuja [2002]. Challenges of face detections include the more intrinsic ones, e.g., the high variability and non-rigidity of faces at large and the even higher variability of facial features such as beards or glasses in particular. Extrinsic factors such as imaging conditions (lighting and camera characteristics) or image orientation further increase the complexity. According to Yang, Kriegman, & Ahuja [2002], approaches to nevertheless address this complexity can be categorized as follows:

Knowledge-based Establish rules of how a face is formed, mostly involving distinctive features such as eyes and nose and their relation to each other.

Feature invariant Denotes methods which try to find facial features that behave robustly toward changes in lighting, pose etc., such as pupils or skin color.

Template matching Refers to methods which correlate an average face pattern with the input image.

Appearance based Methods that learn a face pattern from appropriate training images, mainly using machine learning techniques such as Neural Networks, Naive Bayes classifier, or Hidden Markov models.

To the latter category belongs the method by Viola & Jones [2001], which has received much attention being one of the fastest approaches so far. Given the limited hardware resources of a mobile robot and the wanted real-time ability, computational complexity is one of our main issues pervading this work. Hence, we opted for the object detection framework by Viola & Jones [2001], by that taking advantage of an already trained face detector most conveniently accompanying the OpenCV library[1]. Primarily applied to face detection, we will use this approach to train a hand detector.

To support the hand detection, an image preprocessing step is performed incorporating information drawn from the face detection result. Using the obtained *face bounding box* we can estimate both how far a person is away from the camera and the dimensions of the remaining body. More importantly, however, by simple color analysis of the image detail containing the face, assumptions about the hands' color can be made. This information is used to identify candidate hand regions in the image. This is further of interest as the object detection method we use is only working on graylevel images.

The preprocessing step is presented in the following section. In Section 3.2, the object detection method by Viola & Jones [2001] is described in detail. Section 3.3 then covers constructing and training the hand detectors. Section 3.4 concludes by briefly explaining the actual classification step.

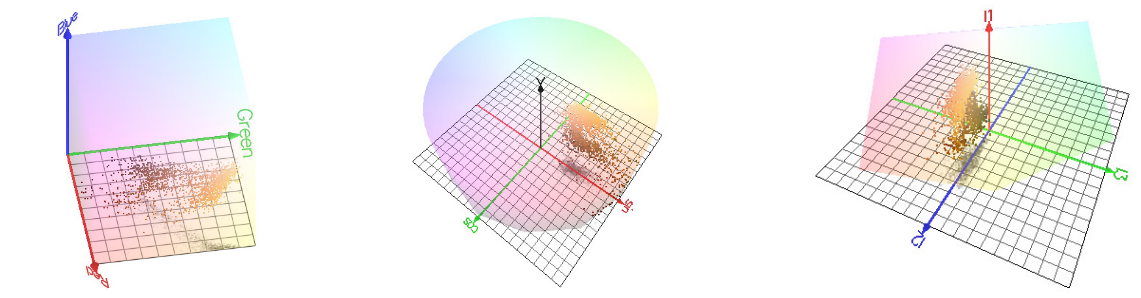
3.1 Image Preprocessing

Starting point of the image preprocessing step is the face detection and the assumption that both the color of the detected face, that is, its areas of skin, and the color of the corresponding hands are similar.

The general idea is to analyze the image patch, given by the face bounding box, to derive the color of the areas of skin, i.e. *skin color*, and classify the pixel colors of the remaining image accordingly. Only within connected regions of an appropriate minimum size identified as skin-colored we will perform the subsequent hand detection.

Skin color as a cue for human hands and, more importantly, face detection has been utilized frequently. Advantages are that color in general is fast to process, skin color in particular is robust toward diverse variations, for example of shape, which made it a prominent feature for the feature-based face detection methods. An elaboration on the usage of skin color is given by Vezhnevets, Sazonov, & Andreeva [2003]. In their paper,

[1]http://sf.net/projects/opencvlibrary



(a) (b) (c)

Fig. 3.1: Three color spaces evaluated in this work, each with the respective cluster of a skin color sample. (a) depicts the RGB color space as a cube and the occupied skin color cluster. One observes its large size and incompactness. In (b) the Hue-Saturation-Value (HSV) color space in cylindric view is shown. It is easiliy seen that the cluster size is smaller and more compact. (c) The I1I2I3 color space with a likewise compact and small cluster.

the authors identify two main problems that arise when using skin color as described: 1) What color space to operate in and 2) how to model the skin color distribution. The following two subsections cover these questions.

3.1.1 Color Spaces

Image data often is represented in the RGB color space, a model composed of the three additive primary colors *red*, *green* and *blue* which originates from computer display applications. For color based image analysis, however, it is considered inappropriate, mainly because of its mixing of luminance and chrominance information. Besides, skin color cannot be represented accurately for it does not cluster nicely in this space, as depicted in Figure 3.1(a). In general, skin color exhibits an immanent variability, though more in intensity than in tone, and thus occupies a more or less large cluster in the respective color spaces.

A variety of color spaces exist, most of which are better suited for representing skin color as RGB and conversion between them is nearly always possible. One obvious criterion for the color space of choice is that it separates luminance and chrominance information to make a color more robust toward illumination variations. A straightforward way to achieve this is to normalize the RGB color space as follows:

$$\begin{bmatrix} r \\ g \\ b \end{bmatrix} = \begin{bmatrix} \frac{R}{R+G+B} \\ \frac{G}{R+G+B} \\ \frac{B}{R+G+B} \end{bmatrix} \tag{3.1}$$

The resulting so-called *pure colors* show much less dependence on the brightness of the original RGB colors, as desired. Further, as $r+g+b=1$, b can be omitted which saves storage space. With its transformation simplicity this has led to a wide use of this color space amongst researchers [Vezhnevets, Sazonov, & Andreeva 2003]. Note that, unlike with most others color spaces, reverse conversion is not possible as the intensity information drops away.

Another criterion for a color space of choice is the compactness of the skin color cluster and how well it separates from other colors. An exhaustive study regarding these questions can be found in [Martinkauppi, Soriano, & Laaksonen 2001]. In this thesis, the commonly used HSV and the specially designed I1I2I3 color space have been evaluated and are briefly described in the following.

HSV Color Space

The Hue-Saturation-Value (HSV) color space describes colors as points in a cylinder whose central axis corresponds to *value*, i.e., brightness, which ranges from black at the bottom to white at the top. Angles around the axis correspond to hue, which defines the dominant color (such as red, green, purple and yellow). The distance from the axis corresponds to saturation, i.e., measures the colorfulness. The HSV color space relates more to the way humans perceive colors, that is, to the questions: What color? How intense? How light or dark?

More formally, a conversion from RGB to HSV can be given as:

$$\begin{bmatrix} H \\ S \\ V \end{bmatrix} = \begin{bmatrix} \arccos \frac{\frac{1}{2}((R-G)+(R-B))}{\sqrt{((R-G)^2+(R-B)(G-B))}} \\ 1-3\frac{\min(R,G,B)}{R+G+B} \\ \frac{1}{3}(R+G+B) \end{bmatrix} \begin{cases} 0^\circ \leq H \leq 360^\circ \\ 0 \leq S \leq 1 \\ 0 \leq S \leq 1 \end{cases} \tag{3.2}$$

In terms of skin color one can observe that it stably consists of reddish color tones and the most variant parts are contained within the V plane. This can straightforwardly be used to roughly isolate possible skin colored regions, see Figure 3.2(b).

The I1I2I3 Color Space

The I1I2I3 color space has first been presented by Ohta, Kanade, & Sakai [1980] and was specially designed to fulfill the following desired properties of a color space used for color segmentation: Separation of luminance and chrominance information, simplicity of transformation from and to other color spaces, no jump discontinuity as HSV (at

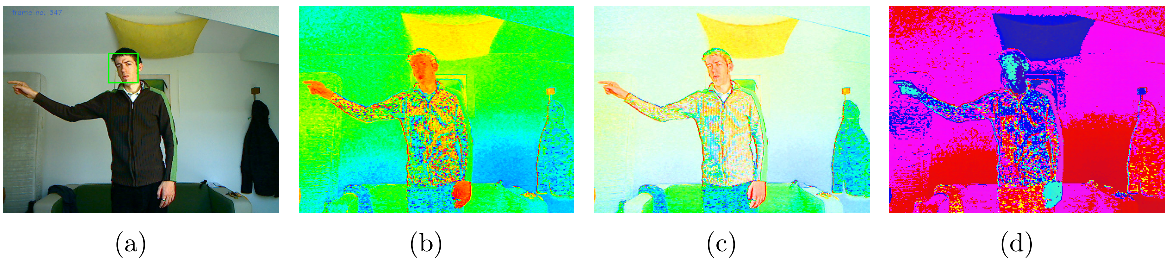


Fig. 3.2: Various chrominance channels. (a) is the source image with drawn in face bounding box. (b) shows the respective *hue* plane, with *saturation* and *value* set to unity. Note the redness of the skin regions. In (c) we see the *hue-saturation* plane, with value set to unity. One observes that the skin color appears rather independent from brightness (∼value). (d) shows the *I2I3* plane with *I1* set to unity. Once again the skin color is easy to distinguish from other colors.

360°) and other color spaces. Conversation from RGB is given as:

$$\begin{bmatrix} I1 \\ I2 \\ I3 \end{bmatrix} = \begin{bmatrix} \frac{1}{3}(R+G+B) \\ \frac{1}{2}(R-B) \\ \frac{1}{4}(2G-R-B) \end{bmatrix} \tag{3.3}$$

As can be seen, $I1$ corresponds to the gray-value axis of the RGB color space and thus contains the intensity information. $I2$ and $I3$ contain the chrominance information and can therefore be used similarly to the HS planes to robustly locate a certain color.

We finally opted for the HSV color space for reasons that will become clear in the following, where we describe how (skin) color distributions can be modeled.

3.1.2 Skin Color Models

Before we can classify the image pixels into foreground and background, we need a model which describes the colors of interest. In the simplest case the model will be a set of rules, i.e., it explicitly defines lower and upper bounds of skin color in the respective color space. The advantage of such a method lies in its simplicity thus allowing for fast classification of unknown pixels. Settling adequate bounds, however, can be a tricky task.

More sophisticated models are distinguished into parametric and non-parametric respectively. Parametric models often have the form of a single Gaussian distribution or a mixture of them. Obviously, their goodness depends on the color distribution they have to model which, in turn, much depends on the chosen color space. Main advantages are their mathematical foundation, their ability to generalize to some extent, and their compact representation. Non-parametric models avoid the dependence

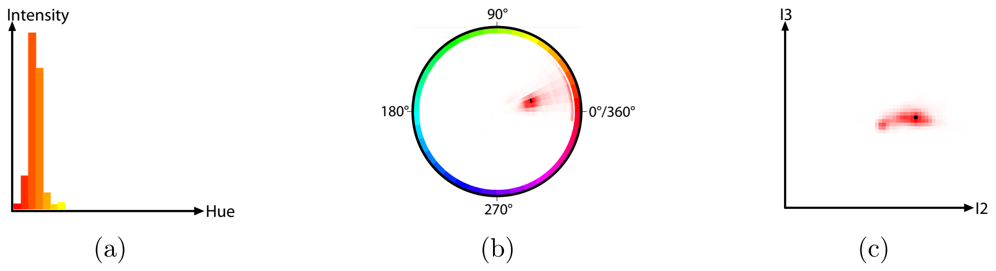


Fig. 3.3: Histograms of the face bounding box patch seen in Figure 3.2(a). Depicted in (a) is the 1D hue histogram with 36 bins. The length of the bars correspond to the number of entries. (b) shows the 2D hue-saturation histogram with 50x50 bins in a polar . The position of the bins correspond to the hue value (angle) and saturation value. The shades of red correspond to the respective hit count. Marked black is the bin with the highest hit count. In (c) finally, the histogram of the I2I3 planes is depicted. Shades of red and the black spot have the same meaning as in (b).

on the color distribution as no explict model is derived. Various realizations of such models exist, for example training an artificial neural network to classify pixels [Brown, Craw, & Lewthwaite 2001], called SOM (Self-Organizing Map).

A more prominent representative for these types of models is the *histogram*. A histogram partitions the color space into equally spaced *bins* thus corresponding to a certain range of color value components. Depending on how many color planes are used, one speaks of either 1D, 2D, or 3D histograms. Each bin stores the frequency of occurrence of pixel values falling into the respective range, as illustrated in Figure 3.3. Usually, one scales these bin values such that the maximum value equals 1, thus transforming the histogram into a probability map, or *Skin Probability Map* as Brand & Mason [2000] termed it. A lookup for an unknown color c:

$$P_{skin}(c) = \frac{hist[c]}{\max hist[c]} \tag{3.4}$$

will return the likelihood P_{skin} of being skin-ish according to this histogram.

In this work we use histograms to model skin color as they are fast to obtain, are independent of the skin color distribution, and can be applied offhand to derive skin probabilities. More important, however, is that in our setting a prior trained skin model is neither necessary nor used as the current skin color is attained from the face. This makes the ability to generalize and complex parameter estimations unnecessary.

3.1.3 Locating Skin Regions

Prior to calculating skin probabilities the histogram has to be prepared. As the face bounding box often includes some background and parts of the face which are not skin colored, it is important to ignore these regions to make sure that only areas of skin contribute to the histogram.

Looking at Figure 3.2, one observes that skin-color has distinct values in the chrominance planes, red-ish in the hue plane, blue-ish in the I2I3 planes. So for instance, it should be safe to demand from the hue of a proper skin color c to satisfy $-45° < hue(c) < 90°$. This directly leads us to reject all pixels having a different hue before calculating the histogram. Similarly, other bounds can be derived for the other planes.

To further refine the histogram, we first smooth the distribution with a Gaussian filter. This diminishes the negative side effects of the binning and reduces the risk of holes in the outcome. The bin with the highest value is determined and a region growing algorithm is applied which isolates the skin color "mountain" from possible others around which may be caused by background pixels. In Figure 3.3 the hue-, hue-saturation-, and I2I2-histogram respectively are shown.

Finally, the *skin probability image* can be obtained by an operation that has been termed histogram *backprojection* by Swain & Ballard [1991]. It refers to the already mentioned simple operation using the scaled histogram as lookup table. Each pixel of the source image is then replaced by the value of the corresponding bin. In a further step, the morphological operations *close* and *open* are applied on the skin probability image to close holes within larger regions and to remove small regions. To homogenize the remaining regions, we smooth the image with a Gaussian filter. As the final step the image is binarized by thresholding it. For an illustration of these steps see Figure 3.4.

Thresholding is one of the earliest and simplest techniques used for image segmentation, see [Sezgin & Sankur 2004]. However, it is not clear in advance what a good threshold value is. Setting the threshold too high will introduce large holes in connected regions or even remove them completely. Setting it too low will possibly introduce new regions which are not skin colored but roughly similar, for example yellow, brown or red objects. Hence, the choice of an adequate threshold parameter is crucial and many methods exist. In this work a combination of two thresholds is applied. The skin probability image is binarized twice using both a high and a low threshold value. If a region resulting from the low threshold binarization contains at least one region resulting from the high threshold binarization both these regions are merged and kept, otherwise the region is dismissed. By this way, holes in actually connected regions are diminished as well as non-skin colored regions.

Figure 3.4(f) shows the final result. Via connected component labeling each individual blob is obtained which will then be the *region of interest* for the successive hand detection.

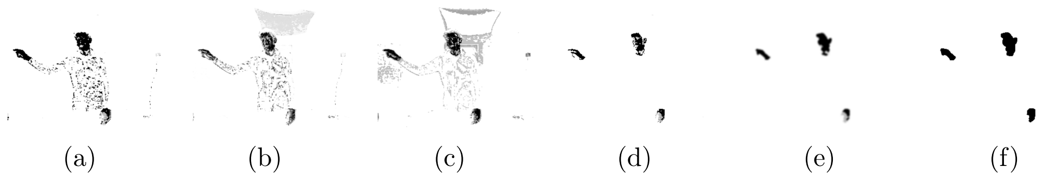


Fig. 3.4: Backprojections of Fig. 3.2(a) (inverted for better readability). In (a) a 1D hue histogram was used, a 2D hue-saturation histogram in (b) and a 2D I2I3 histogram in (c). Observe that not only skin colored parts are left back. This is mainly due to very dark or very bright pixels which often have no defined color. (d) shows (a) after excluding such pixels. In (e) the morphological operations *close* and *open* have been applied to remove small regions and close holes in larger regions. (f) The final binarized image.

As a result, we found that with the I2I3 histogram in fact the best accuracy was obtained, followed by the 2D HSV histogram. We found further, however, that regardless of the chosen color space, the hands can be of different color than the face. This is especially observable under bad lighting conditions. For this reason, we opted for the 1D HSV histogram, which is less exact. To diminish this problem, we start updating the histogram with color information from the hand regions once they have been detected by the hand classifier. By this, we adapt the segmentation process to the current color of the hands. To account for the fact that hands are rather elliptic than rectangular, the region containing the hand is weighted with the Epanechnikov kernel [Epanechnikov 1969], given as

$$k_E(r) = \begin{cases} \frac{3}{4}(1-r^2) & \text{for } r \leq 1 \\ 0 & \text{otherwise,} \end{cases} \tag{3.5}$$

where r denotes the the distance from the center of the region. Using the weighted hand region to update the histogram ensures that pixels near the center which most likely contain hand pixels count the most as opposed to pixels which are farther away.

This completes the section on preprocessing which serves the purpose to constrain the image to regions where we expect hands to be found. The means to detect these will be the object detection framework by Viola & Jones [2001], presented in the following.

3.2 Rapid Object Detection

The general modus operandi when detecting objects is to move a search window in small steps and with increasing dimensions over the image and to process the detail defined by the search window. The patch can be used directly as an input to, for instance, a neural network or one could try to extract some meaningful *features*. Often this is to prefer, as dimensions get reduced. Additionally, features usually compensate small variations of the object better, as they in general depend on larger regions than only one pixel.

In all cases, however, the described process can be quite time consuming as each detection step will be called thousands of times and thus every effort has to be undertaken to minimize the time needed for such a detection step. The object detection framework for graylevel images proposed by Viola & Jones [2001] achieves this by introducing three concepts:

1) The first concept relates to the utilization of simple rectangular features in conjunction with so-called *integral images* which allow for extremely fast computation of the corresponding feature values.

2) The second concept is a way to select of possibly hundreds of thousands of features only those which are the most discriminative. An adaptive version of the *Boosting* algorithm is used here to build efficient and strong classifiers with only a small number of features.

3) The third contribution is the arrangement of a set of such classifiers of increasing complexity into a *cascade*. This allows for fast rejection of image regions that do not contain the object.

These three concepts will be described in the following sections in more detail.

3.2.1 Feature Computation

As mentioned, simple rectangular features are used to describe the object in question. Each feature is divided into two, three, or four regions, as shown in figure 3.5(a) - 3.5(o). Each such feature can be arbitrarily scaled and translated within the search window and the *feature value* is then computed as the sum of pixel intensities covered by the white area(s) subtracted by the sum of pixel values under the red area(s). Due to their simplicity, only horizontal, vertical, and to some extent diagonal structures can be captured well but their sheer number[2] and the already mentioned possibility of fast feature value computation well compensate for this somewhat restricted flexibility.

[2]In a 24x24 search window one can construct over 160000 such features, see [Lienhart & Maydt 2002] how to compute the exact number.

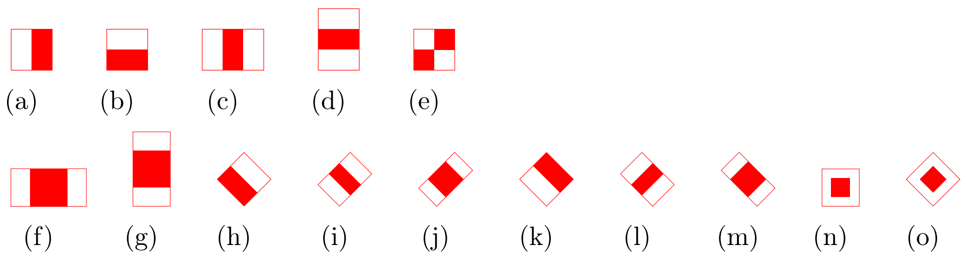


Fig. 3.5: Rectangular features used in this work. Top row: The original features used by Viola & Jones [2001]. (a) and (b) adapt to edges, (c) and (d) are line features, whereas (e) catches diagonal structures. Bottom row: Extended feature set by Lienhart & Maydt [2002]. The main contribution is the introduction of 45° rotated features (h) - (m) and (o) which increases the overall expressive power. The feature value is computed as the sum of pixels covered by the red part subtracted by the sum of pixels under the white parts.

The Integral Image

To compute the feature value the pixel intensities have to be summed up and subtracted accordingly. For fast computation of these sums, Viola & Jones [2001] introduced the integral image, which, once computed, allows for pixel summation of arbitrarily sized rectangles in constant time. Having the same dimensions as the original image each pixel entry $ii(x, y)$ equals the sum of all pixel contained in the rectangle spanned by the image origin $(0, 0)$ and (x, y) (see Figure 3.6):

$$ii(x,y) = \sum_{x' \leq x} \sum_{y' \leq y} i(x', y') \tag{3.6}$$

With an appropriate recurrence or dynamic programming scheme the integral image can be computed in one single pass. The pixel sum s of an arbitrary rectangle (x, y, w, h), w and h being the width and the height respectively, can now be computed by the four pixel sums $a = ii(x-1, y-1)$, $a = ii(x+w-1, y-1)$, $a = ii(x-1, y+h-1)$, $a = ii(x+w-1, y+h-1)$:

$$s = d - b - c + a \tag{3.7}$$

that is, in constant time, as desired (see Figure 3.6).

3.2.2 Boosting Classifiers

Boosting, first introduced in [Schapire 1990], belongs to the so-called ensemble methods which follow the idea to combine several *weak learners* to a single *strong learner*. Weak learners are simple classifiers which only have to correlate slightly with the true classification though better than a random guess would do. The classification of the

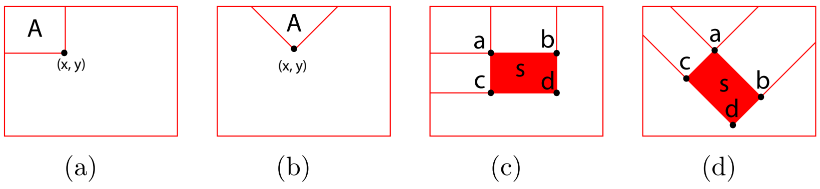


Fig. 3.6: (a) The value of the integral image at (x, y) equals the sum of the pixel values of the original image covered by A. (b) For rotated features a slightly different representation is necessary. (c) and (d) The pixel sum within rectangle s can be computed as follows: Subtracting from the large rectangle d, rectangles b, and c, and adding rectangle a, as its area got subtracted twice, i.e., $s = d - b - c + a$

final strong classifier is a simple majority vote of its weak classifiers and it is proven that, by taking an appropriately high number of weak classifiers the combined classification then correlates arbitrarily well with the true classification.

The usefulness of such an algorithm in the case at hand is twofold: On the one hand, it provides a method to combine weak classifiers based on simple rectangular features to a single strong classifier. On the other hand, it allows to reduce the set of possible features, which, as mentioned, is very large, by selecting only the most discriminative features.

While the original version of the boosting algorithm conducts a fixed number of rounds to create the final strong classifier, AdaBoost - presented in [Freund & Schapire 1995] - which is used here, adapts to the training error of the weak classifiers until a desired small error rate is reached. It is presented in the following section.

AdaBoost

Algorithm 1 ADABOOST

Input: a training set $(x_1, y_1), \ldots, (x_N, y_N)$ where $y_i - 1$ for negative and $y_i - 1$ for positive samples

Output: the final strong classifier $H(x) = sgn\left[\sum_{t=1}^{T} \alpha_t h_t(x)\right]$

```
1: $w_i^1 = \frac{1}{N}$ for $i = 1 \ldots N$                         ▷ initialize weights
2: for $t \leftarrow 1, T$ do
3:     call WEAKLEARNER which returns $h_t(x)$ which minimizes $\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} w_i^t$
4:     $\alpha_t = \frac{1}{2} ln \left(\frac{1-\epsilon_t}{\epsilon_t}\right)$                 ▷ compute classifier weight
5:     $w_i^{t+1} = w_i^t exp(-y_i \alpha_t h_t(x_i))$                  ▷ update weights
6:     $w_i^{t+1} = \frac{w_i^{t+1}}{\sum_i w_i^{t+1}}$                        ▷ normalize weights
```

The AdaBoost algorithm (see Algorithm 1) receives as input a sequence of n labeled training samples $(x_1, y_1), \dots, (x_n, y_n)$. x_i denotes the image patch, y_i the class label with $y_i = 1$ for positive samples and $y_i = -1$ for negative samples.

AdaBoost proceeds in a series of rounds $t = 1, \dots, T$ in the following fashion. At each round t the WeakLearner generates a weak classifier $h(x) \in \{-1, +1\}$ which best separates the positive and the negative examples with respect to the current weights w^t. These weight are initially uniformly distributed. The choice of the best hypothesis is governed by the error function

$$\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} w_i^t \tag{3.8}$$

which has to be minimized. A classifier weight α_t is computed reciprocally proportional to the error. Then *reweighing* is conducted according to the performance of the current weak classifier h_t. The goal of the update rule is to decrease the weight of correctly classified examples and increase the weight of misclassified examples. In this way, in the next round the algorithm is forced to concentrate on the hard examples which were poorly predicted in former rounds.

The last step of the boosting algorithm is to calculate the final strong classifier as a linear combination of all weak classifiers.

$$H(x) = sgn\left[\sum_{t=1}^{T} \alpha_t h_t(x)\right] \tag{3.9}$$

The parameters α^t are used as weighting factors to measure the importance of each weak hypothesis h^t. The weight of a hypothesis h^t gets larger as its error ϵ^t gets smaller. In this way, AdaBoost adapts to the accuracies of the generated weak classifiers.

What is left to clarify is how the procedure WeakClassifier actually constructs and learns a weak classifier out of the features. To keep it simple (and therefore fast), a weak classifier $h_i(x)$ only consists of a single feature f_i and a threshold θ_i such that classification is carried out according to following equation:

$$h_i(x) = \begin{cases} 1 & \text{if } p_i f_i(x) < p_i \theta_i \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

The parity p_i changes the sign of the inequality, meaning that if $p_i = 1$ positive examples are below the threshold and if $p_i = -1$ above. In order to calculate the optimal threshold θ_i and the parity p_i for a given feature f_i the corresponding values for N

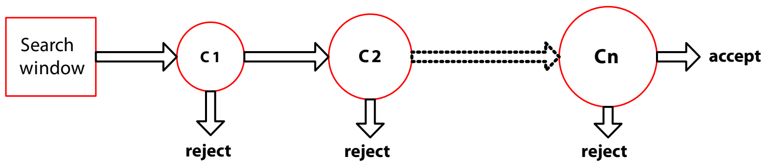


Fig. 3.7: The classifier cascade consists of n individual, increasingly complex classifiers. Only if all of them accept the input window the final classifier returns positive. The first few classifiers are trained in such a way that they reliably reject most of the negative (= background) samples.

training samples are sorted ascendingly: $f_i(x_1) < \ldots < f_i(x_N)$. The threshold is positioned between two successive values and the resulting classification error according to the weights and the class of the sample (either positive or negative) is computed. Starting from top of the list, i.e., position the threshold between $f_i(x_1)$ and $< f_i(x_2)$ all possible classification errors can be computed in a single pass. This is done twice setting $p = 1$ and $p = -1$ respectively. The combination of p and the threshold leading to the minimum classification is then returned.

3.2.3 Cascading Classifier

To further decrease computation time and even increase detection performance the concept of a *cascading classifier* is introduced. A cascade is a hierarchical construct similar to a linear decision tree, containing (strong) classifiers of increasing complexity (see Figure 3.7). If the currently evaluated sub-window passes one stage it is forwarded to the next stage, otherwise rejected immediately. Only if all stages are passed successfully the input is classified as positive. While the first few stages are designed to roughly distinguish between object and non-object the subsequent are trained to focus on the differences between object and object-similar background. This allows for very simple classifiers in the beginning, i.e., containing only a few weak classifiers whilst in the final stages the complexity increases more and more, as more features are needed to describe the subtle differences between object and object-similar background. In the example classifier of Viola & Jones [2001] the first stage contains only two features, the final one as many as 200.

Clearly, this achieves the desired reduction of computation time as for a proper background region, which will be the vast majority, only the early stages will be needed to reject it, thus requiring the computation of only a few dozens of feature values. In contrast, a monolithic approach would need to evaluate all contained weak classifiers, of which there can be thousands. To see that, in return, detection performance increases, recall that all subsequent stages have to deal only with what passed the previous stage, that is, they can use rather specialized features on this preselected input, which might fail on completely unexpected, i.e., arbitrary input.

In order to obtain classifiers with the desired behaviour two conditions have to be met: On the positive training data a minimum detection rate d_i is demanded as well as a maximum false positive rate f_i. For N such classifiers in a cascade the overall detection rate D and overall false alarm rate F can be computed as

$$D = \prod_{i=1}^{N} d_i, \quad F = \prod_{i=1}^{N} f_i \tag{3.11}$$

With the typical values $d_i = 0.995$ and $f_i = 0.5$ and 20 stages a sufficiently high $D \approx 0.9$ and sufficiently low $F \approx 9.5 \cdot 10^{-7}$ is achieved.

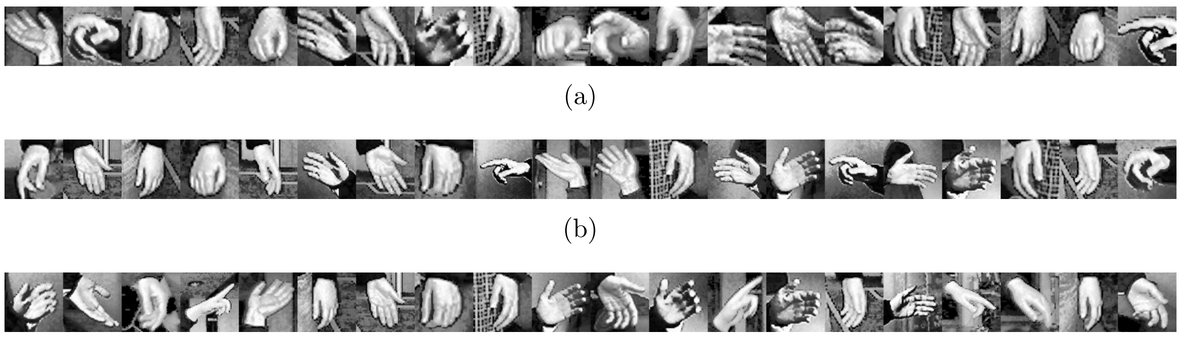
Training such a stage classifier c_i requires slight modifications to the original AdaBoost algorithm as it tries to minimize the overall classification error instead of taking into account certain (high) detection rates and appropriate (low) false alarm rates. A simple trick is applied to make this possible: a threshold b, with $b > 0$, is added to Equation 3.9. By increasing b, more samples, both negative and positive, pass the classifier, thus the detection rate is regulated. By performing this step the false alarm rate also increases. If it exceeds the required maximal false alarm rate f_i new weak classifiers have to be added until both conditions are met.

3.3 Constructing the Hand Detector

It is obvious that hand detection is immanently harder than for example face detection. Whereas faces are usually upright, hands can appear arbitrarily rotated, in-plane as well as out-of-plane. In addition, they can assume the most various shapes as they are highly articulated with more than 20 degrees of freedom. In fact, without context information there certainly are hand configurations which would hardly be recognized even by humans. Context information such as an arm which further is connected to the body of a person allows to nevertheless classify this uncommon hand configuration correctly.

We construct two classifiers: a *generic* hand classifier which detects hands and rejects non-hands, and a *specific* hand classifier which is able to discriminate right from left hands. For the latter, we trained a classifier with positive samples containing hands of a particular (in our case right) *laterality* only. As hands are symmetric, it can be used to detect left hands as well. We connect both classifiers in a row such that the specific classifier is only applied in case the generic classifier detected a hand at all.

In the following section we present several strategies which have been employed to obtain a suitable collection of training samples.



(a)

(b)

(c)

Fig. 3.8: Image patches containing hands. In (a) we see hands that have been cropped using the given size of the face bounding box such that we observe smaller and bigger hands. (b) No fixed size is chosen but according to the actual dimension of the hand. In (c) right hands are seen half of which are flipped left hands from the set of samples shown in (b).

3.3.1 Collecting Training Data

The main idea is to constrain both positive and background samples in such a way that they correlate to the detection scenario. The same way the detection process is restricted to skin-colored regions, only these regions are considered at all during training data acquisition. At the first sight this seems to make little sense as the most proper skin color in a grayscale image differs not much from, for instance, a proper green. However, the intuition is that skin-colored objects might share other properties, for example similar structure, which in case of faces certainly is true.

Another cue that is being incorporated affects the size of positive samples. We observe that there is a correlation between the dimensions of hand and corresponding face. Rarely larger than the face, the hand will mostly fit easily into the face bounding box. By cutting out an image patch containing the hand with the same size of the face bounding box context information is included which might be useful. See Figure 3.8(a) for a collection of positive samples.

To the specific classifiers' training data, image patches containing the opposing hand are added as negative samples. As we cannot know which negative samples are in fact considered during the AdaBoost rounds as this depends on the error weights, another version of the specific classifier is trained. For this *tuned* version, we limited the training data to consist only of hand image patches, that is, right hand image patches as positive samples and left hand image patches as negatives. This should also relate more to the detection scenario where the specific classifier is only applied to regions that are likely to contain hands.

To summarize, we collect the training data as follows:

- apply the face detector to the input image
- extract skin-colored regions as described in Section 3.1
- if skin-colored, mark hands as either left or right
- crop and scale the positive samples to equal sizes, here 24x24 pixels
- for the specific classifier, flip image to add the opposing hands to the respective negative and positive sample pool
- crop all other skin-colored regions as negative samples

In a first attempt we selected only a few hundred positive samples from a single person, yet additionally investigated an idea to improve the positional accuracy of the classifier. More exactly, by providing many samples where the hand is precisely centered and less where it is shifted a few pixels, we want to force the classifier to prefer the former. For that purpose, each positive sample was duplicated according to following Gaussian distribution matrix

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

where the rectangle defining the image patch to be cropped was shifted accordingly. As a side effect, this enlarges the set of positive samples by the factor 16.

In a second step we repeated the above to obtain more samples, both negative and positive (around 5000), with other people, different background and lighting conditions and applied the already trained classifiers to include *false positives* into the collection of negative samples in order to further improve the detection accuracy.

For reasons of comparison, we also trained classifiers without the imposed restrictions on the training data set. More exactly, the size of the positive sample patch is not dependent on the face bounding box size but corresponds to the actual hand dimensions. For a selection of such positive samples see Figure 3.8(b) and 3.8(c). As background samples we chose images containing skin regions, but not containing hands. To our surprise, we experimentally found the generic classifier trained this way to outperform the restricted ones. For the comparison and a discussion see Chapter 7.

3.4 Detecting Hands

We detect hands according to Figure 3.9. Starting from the face detection result, image preprocessing is performed as described in Section 3.1. Additionally, the size of the

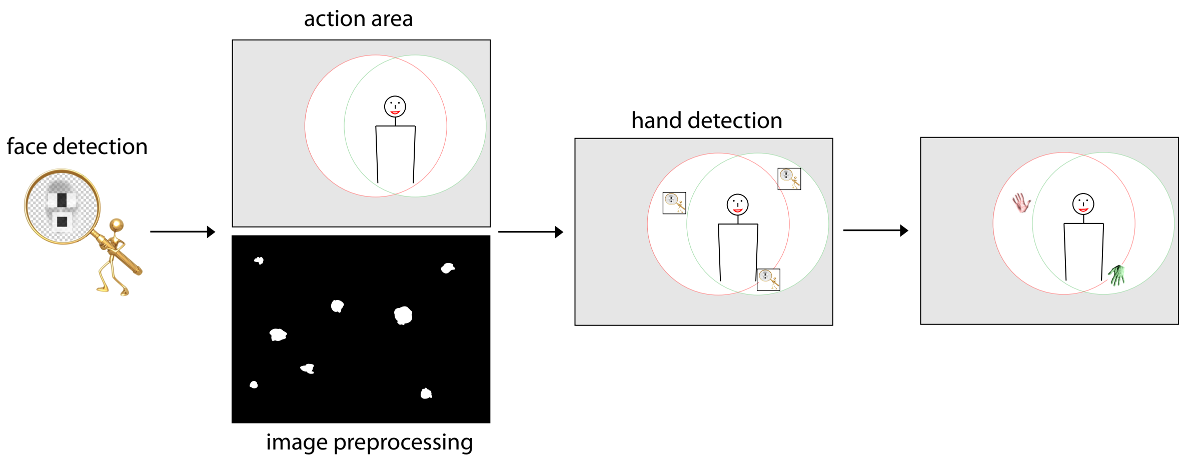


Fig. 3.9: The detection process. First faces are detected. From the face bounding box derive action area and perform image preprocessing. Then apply hand detectors at each skin region within action area. As a result we get hands which are either classified as right (in the image green) or left (red) or unknown.

face bounding box is used for an estimation of an *action area* of the hands. The action area is simply the union of the two circles centered at the estimated shoulder joints with the outstretched arms as radius. The arm length is also determined from the face bounding box size.

For each of the skin-colored regions within this action area the hand detectors are applied with a search window size set to the corresponding face bounding box size. First, the generic hand detector is applied. In case there is a hand, the search window is usually accepted at multiple locations and multiple scalings which are close to each other. This is understood, since the features used, especially the larger, are invariant to slight shifts and variations in size. The number of all accepted search windows can therefore serve as a certainty measure. In case it exceeds a threshold ϵ, the search windows are clustered based on a distance metric. In each of the resulting mean rectangles the right hand classifier is applied twice, once in the original image and once in the flipped. Four cases are possible:

1) No success in both images $\Rightarrow$ return generic hand.

2) Success in original image $\Rightarrow$ return right hand.

3) Success in flipped image $\Rightarrow$ return left hand.

4) Success in both images $\Rightarrow$ compare the numbers of accepted search windows and return the hand corresponding to the higher number.

This concludes the detection process. Of course, false detections of both the generic and the specific classifier are possible as well as missing detections. We therefore maintain a probabilistic belief over a hand existence and laterality. This, amongst others, is the subject of the next chapter.

4 Tracking People

Having located faces and hands in the current image, the next step is to determine correspondence with previously detected faces and hands. In doing so, we track faces and hands over time and thus obtain the *trajectory* in space. Additionally, we have to ascertain the belonging of hands to a particular face, thus constituting our compact human model.

Correspondence can be settled by different means and different similarity measurements. In the domain of face recognition one often applies principal component analysis (PCA) to re-identify known faces. However, such approaches are too time-consuming for our purposes so that we rely on a distance-based measure. This allows for fast frame rates which in turn makes it robust as the average displacement in each frame is sufficiently small. To find the globally best assignment given some associated costs, we make use of the Hungarian method which is described in Section 4.1.

To account for the possibility that a face or, in particular, hand does not get detected in every frame, other means to keep track of it have to be applied. In this work, we employ a Kalman filter to predict the next positions and to filter noisy measurements, combined with a fast color-based tracking algorithm.

Finally, yet another assignment problem has to be solved. This time, we have to decide for each hand if it could possibly belong to a certain face thus constituting a person. Up to two hands can be assigned to each person and again this is a global problem as each hand can only get assigned once. Again, we use the Hungarian method to find the best assignment.

4.1 Hungarian Method

The Hungarian method has first been devised by Kuhn [1955] as a means to solve the so-called *assignment problem*. The assignment problem is a combinatorial optimization problem which consists of finding matching pairs for the elements of two sets by considering associated costs that have to be minimized (alternatively, benefits that have to be maximized). It corresponds to the graph theoretic problem of finding a maximum-weight matching in a weighted bipartite graph. A common exemplification is to consider a number of agents and jobs. Each agent can perform exactly one job each causing different costs and the goal is to dispatch the jobs to the agents such that minimal costs arise.

In a brute-force approach this problem can be solved considering $n!$ combinations, n being the number of agents or jobs; with the Hungarian method the time complexity is reduced to $O(n^3)$. To achieve this, the costs for every possible assignment are arranged into a cost matrix C,

$$C = \begin{bmatrix} c_{0,0} & \cdots & c_{0,n} \\ \vdots & \ddots & \vdots \\ c_{n,0} & \cdots & c_{n,n} \end{bmatrix} \tag{4.1}$$

where c_{ij} denotes the costs caused by the assignment of the jth job to the ith agent. If C is not a square matrix (more jobs than agents or vice-versa), C is augmented with dummy jobs or agents respectively usually with zero costs or some threshold costs. The actual algorithm is listed in Algorithm 2. Obviously, the key challenge is to define an appropriate cost function, which largely depends on the case at hand and thus will be described in the corresponding sections.

4.2 Kalman Filter

The Kalman filter was developed by Kalman [1960] and is used to estimate the state of a linear dynamic system based on possibly incomplete and noisy measurements. Meanwhile, there exist variants for non-linear systems such as the Extended Kalman filter (EKF) or Unscented Kalman filter (UKF). Welch & Bishop [1995] provide a good introduction into Kalman filters and all its variants. In this work, we only used the basic Kalman filter.

The Kalman filter algorithm can be described as a two-step process. Firstly, there is a prediction step that makes use of the learned system dynamics to anticipate the next state of the system. Secondly, an update step occurs that takes a new observed measurement and uses it to correct the system dynamics and system state. In the

Algorithm 2 HungarianMethod

Input: a cost matrix C where c_{ij} denotes costs of assigning job i to worker j

Output: modified cost matrix C, with zeros representing optimal assignment

```
 1: for $r \leftarrow 0, Rows$ do                          ▷ for each row
 2:     $c_{r,j} \leftarrow c_{r,j} - min(c_{r,0:n})$      ▷ subtract row minimum from each row entry
 3: for $c \leftarrow 0, Cols$ do                          ▷ for each column
 4:     $c_{i,c} \leftarrow c_{i,c} - min(c_{0:n,c})$      ▷ subtract col minimum from each col entry
 5:
 6: cover all zeros with the minimum number of lines
 7: if $num\_lines == n$ then
 8:     find assignment combination with zero costs
 9: else
10:     min = $min\_unc(c_{0:n,0:n})$                      ▷ find minimum uncovered entry
11:     $c_{i,j} \leftarrow c_{i,j} -$ min                 ▷ subtract from all uncovered entries
12:     $dc_{i,j} \leftarrow dc_{i,j} +$ min               ▷ add to all doubly covered entries
13:     goto line 5
```

domain of object tracking the state of a system usually comprises position, velocity and possibly acceleration of the object, whereas only object locations are measured.

Algorithm 3 lists the steps performed by the Kalman filter. During the prediction step the prior state $\overline{x_t}$ is estimated according to Line 2. Here, A denotes the state transition matrix and relates to a motion model in the object tracking domain. B relates the option control input (e.g., motor control of an mobile robot) with the estimated state.

The estimated state deviates from the actual state by a certain estimation error which is given by the error covariance matrix Σ_{t-1}. It is estimated in Line 3 according to the transition matrix plus additional Gaussian process noise Q_t.

In the correction step the estimation $\overline{x_t}$ is improved by means of the measurement z_t. This measurement is assumed to relate to the state according to following equation

$$z_t = Hx_t + v_t \ . \tag{4.2}$$

Here, H is a matrix that captures the relation between state and measurement. In the simplest case, that is, all of the system state can be measured, it will be the identity matrix. v_t denotes the measurement noise and is assumed to be zero mean Gaussian noise with covariance R_t: $v_t \sim \mathrm{N}(0, R_t)$.

The algorithm then proceeds by computing the Kalman Gain K_t according to Line 6. K_t is finally used to calculate the posterior state x_t and the error covariance Σ_t.

Algorithm 3 KalmanFilter

```
Input: state $x_{t-1}$ and associated error covariance matrix $\Sigma_{t-1}$, $u_t$ current control, $z_t$
    new measurement
Output: new state $x_t$ and associated error covariance matrix $\Sigma_t$
1: Prediction:
2: $\overline{x_t} = A_t x_{t-1} + B_t u_t$
3: $\overline{\Sigma_t} = A_t \Sigma_{t-1} A_t^T + Q_t$
4:
5: Correction:
6: $K_t = \overline{\Sigma_t} H_t^T (H_t \overline{\Sigma_t} H_t^T + R_t)^{-1}$
7: $x_t = \overline{x_t} + K_t (z_t - H_t \overline{x_t})$
8: $\Sigma_t = (I - K_t H_t) \overline{\Sigma_t}$
```

In the object tracking domain, the choice of the system state and the corresponding transition matrix is of practical interest. In the simplest case we assume the object to move at constant velocity. Then, the system state $x = [x, y, v_x, v_y]$, with the state transition matrix

$$A = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.3}$$

We further considered a constant acceleration motion model, thus having a system state $x = [x, y, v_x, v_y, a_x, a_y]$ and accordingly

$$A = \begin{bmatrix} 1 & 0 & \Delta t & 0 & \frac{1}{2}\Delta t^2 & 0 \\ 0 & 1 & 0 & \Delta t & 0 & \frac{1}{2}\Delta t^2 \\ 0 & 0 & 1 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.4}$$

Besides, the correct initialization of error and noise matrices is essential as these will not get updated by the Kalman filter process.

4.3 Tracking Faces and Hands

Tracking hands is a more challenging problem as we will point out. Thus, additional cues are required for reliable tracking results. Hence, in the following first the method to keep track of faces is described. Extensions which are necessary to apply this method to hand tracking are developed thereafter.

4.3.1 Tracking Faces

In our approach to keep track of a particular face we follow [Bennewitz et al. 2005]. In the absence of a means to re-identify previously seen faces a distance-based cost function is used to solve the data association problem. We apply Kalman filtering to smooth the obtained trajectory and to support the data association. The general process is shown in Algorithm 4. The entries c_{ij} of the cost matrix C reflect the distance of the new observations to the expected positions of the already tracked faces (*tracker*) which must not exceed a maximum distance. If it does, no assignment is considered by setting c_{ij} to maximum. The maximum distance depends on the time elapsed since the last update and the maximum Kalman filter error.

Then, the correspondences are obtained using the Hungarian method as described above, also depicted in Figure 4.1. For each observation that cannot be assigned, the appearance of a new face is assumed, hence a new *tracker* is set up. An associated Kalman filter is initialized with a constant velocity motion model which for appropriately high frame rates is a tolerable approximation of the true motion. Successful assignments are used to update the associated Kalman filter and a (thusly smoothed) trajectory is obtained as the sequence of centers of the observation rectangles.

To provide a means to deal with both false and missing detections, we additionally maintain a probabilistic *belief* which reflects the detection frequency. Each time step the belief is updated in virtue of following equation:

$$bel(f|z_{1:t}) = \left[1 + \frac{1-\Pr(f|z_t)}{\Pr(f|z_t)} \cdot \frac{\Pr(f)}{1-\Pr(f)} \cdot \frac{1-bel(f|z_{1:t-1})}{bel(f|z_{1:t-1})}\right]^{-1}. \tag{4.5}$$

Here, f denotes the existence of a face, z_t is the observation (face detected/not detected) at time t, and $z_{1:t}$ refers to the observation sequence up to time t. The prior probability $\Pr(f)$ is commonly set to 0.5. Therefore, the second term in the product in Eq. (4.5) becomes 1 and can be neglected. Further values that have to be specified are the probability $\Pr(f|z = det)$ that a face exists if it is detected in the image and the probability $\Pr(f|z = \neg det)$ that a face exists if it is not detected (anymore). We experimentally found out that adequate values for those parameters are 0.9 and 0.2, respectively. Using the update rule in Eq. (4.5), the belief increases if positive observations occur and decreases otherwise.

If the belief drops below a certain threshold, the face is considered not to exist anymore. Either it moved out of the field-of-view or it was a false positive in the first place. In these cases, the corresponding *tracker* is deleted.

Algorithm 4 HungarianAssign

```
Input: the set $O$ of new observations
Output: updated/new trackers
 1: set up cost matrix $C$
 2: for all $o \in O$ do
 3:     for all trackers $t$ do
 4:         $p \leftarrow pred\_pos(t)$            ▷ get predicted position from Kalman filter
 5:         compute $dist(o, p)$            ▷ compute Euclidian distance to detection
 6:         if $dist < MAX\_DIST$ then
 7:             set $c_{ij}$ proportional to $dist$
 8:         else
 9:             $c_{ij} \leftarrow MAX\_COST$
10: HungarianMethod($C$)            ▷ call Hungarian method with cost matrix $C$
11: for all $o \in O$ do
12:     if $assigned(o)$ then
13:         update corresponding tracker
14:     else
15:         create new tracker
```

4.3.2 Tracking Hands

To track hands, we apply the same method as with faces. However, hands and their movements reveal some peculiarities which considerably increase the complexity and are as follows:

1) Hands can be of either left or right *laterality*.

2) Hands may move much faster and with much more varying speed than faces do.

3) The detection rate for hands will be lower than for faces.

The first point means we detect hands of either laterality with varying certainty (see Section 3.4), and one and the same hand may occasionally get classified differently. However, we expect a strong tendency toward a particular (= the true) laterality over time. We therefore require that assigning observations of one type to *trackers* of the opposed type is less likely than to the same type, depending on the respective certainty. The second point is obvious and leads to much larger displacements between frames (which makes low frame processing times so vital). The last point depends partly on the second: especially during fast movements we have an increased incidence of *motion blur* in such a way that only contourless colors remain, thus making hand detection

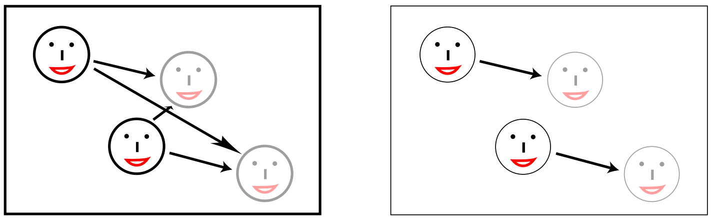

(a) (b)

Fig. 4.1: The assignment problem with two existing faces and two new observations. In (a) the new observations in black are assigned to each of the existing faces (in gray) with the arrow length indicating the costs. In (b) the global minimum is found as desired.

impossible. Another reason for lower detection rates is that there will be innumerable hand shapes which have not been seen by the detector and hence will get rejected.

In the following we will devise strategies to tackle these problems adequately.

Laterality of the hands In order to solve the first problem, we fall back on the recursive Bayesian update scheme as of Eq. 4.5 and additionally maintain a belief about the hands' laterality. Thus, we incorporate the certainty given by the *specific* hand detector, which distinguishes between right and left hands:

$$bel(h_r|z_{1:t}) = \left[1 + \frac{1-\Pr(h_r|z_t)}{\Pr(h_r|z_t)} \cdot \frac{1-bel(h_r|z_{1:t-1})}{bel(h_r|z_{1:t-1})}\right]^{-1} \quad (4.6)$$

Here, $bel(h_r|z_{1:t})$ denotes the belief that the hand is of right laterality given the observation sequence up to time t. Obviously, it holds that $bel(h_l|z_{1:t}) = 1 - bel(h_r|z_{1:t})$.

We performed experiments to determine the probability $\Pr(h_r|z = h_r)$ that a hand is a right hand given it was detected as such and the probability $\Pr(h_r|z = h_l)$, respectively. In doing so, we only considered detections with a certainty of 100%. This means, we adjust

$$\Pr(h_r|z = h_r) = \frac{1}{2} + \left(\Pr_{\max}(h_r|z = h_r) - \frac{1}{2}\right) \cdot certainty_z \quad (4.7)$$

with $0.5 < certainty_z \leq 1$. Thus, the belief remains nearly unaffected by uncertain measurements but is largely governed by certain observations. Note that in case the hand is found only by the generic hand detector $\Pr(h_r|z = h_r) = 0.5$, i.e., the believe does not change.

To reflect the fact that we only want to assign hand observations of a particular type to hand trackers of the same type, we define the following assignment probability

$$P_{assign} = \Pr(h_r|z = h_r) \cdot bel(h_r|z_{1:t}) + \Pr(h_l|z = h_r) \cdot bel(h_l|z_{1:t}) \tag{4.8}$$

We then adapt the cost function to incorporate both the distance measure and P_{assign} in accordance with following formula

$$c_{ij} = \frac{\alpha}{P_{assign}} + \frac{1-\alpha}{e^{-\frac{d^2}{\sigma^2}}} \tag{4.9}$$

Here, d is the distance between predicted and detected and is the argument for the normal distribution N $\sim (0, \sigma^2)$. σ^2 is chosen in such a way that if d exceeds MAX_DIST the probability gets close to zero. To determine the weighting factor α, we define P_{assign}^{worst}, that is, the lowest probability for which assignment will be possible, and set it equal with the corresponding distance probability:

$$\frac{\alpha}{P_{assign}^{worst}} = \frac{1-\alpha}{e^{-\frac{d_{worst}^2}{\sigma^2}}} \quad \Rightarrow \alpha = \frac{P_{assign}^{worst}}{P_{assign}^{worst+e^{-\frac{d_{worst}^2}{\sigma^2}}}} \tag{4.10}$$

Fast hand movements For the second problem we reduced the needed time for processing a single frame. Currently, we achieve a frame rate of 20fps. This keeps the displacements within a tolerable limit. In addition, instead of using a constant velocity model, we switched to a constant acceleration model for the Kalman filter which proved to adapt faster to changing velocities, thus providing better predictions.

Missing hand detections As the phases of non-detection can be longer than could be bridged by solely using the Kalman filter predictions, further tracking techniques are applied to follow the hand in motion during such phases. The CAMShift algorithm by Bradski [1998] is a simple yet efficient color-based tracking algorithm and will be used in this work. It is described in the following.

CAMShift

CAMShift, short for continuously adaptive mean-shift, by Bradski [1998], is a color-based tracking algorithm. It is grounded on the general mean-shift algorithm to find the mode (peak) of an arbitrary probability distribution. Mean-shift proceeds in a series of iterations or until a certain convergence criterion is met, for instance, the mode changes less than a given threshold ϵ. It starts with an arbitrarily positioned search window covering a part of the distribution, as in Figure 4.3(a). The mean of this part is computed and the search window centered thereat (Figure 4.3(b)). This is

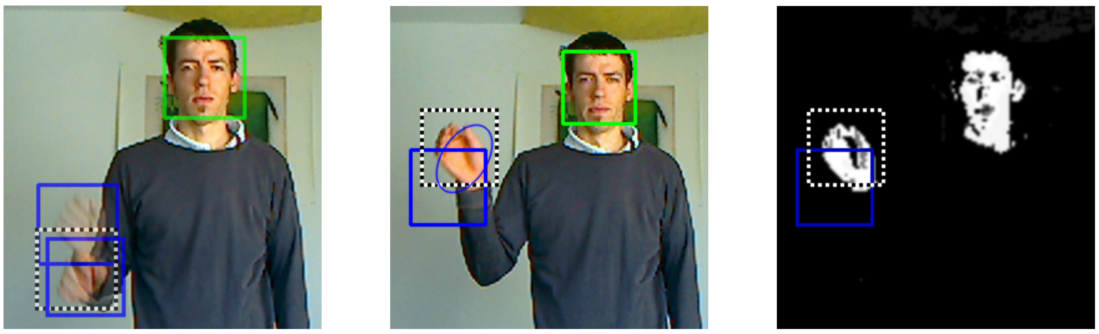

(a) (b) (c)

Fig. 4.2: Tracking with Kalman filter. (a) Shows a downward moving hand during two frames which is detected in both frames. The dashed rectangle denotes the Kalman filter prediction for the second frame which is remarkably close to the true position. In (b) we see an upward moving hand which was not re-detected. Again, the dashed rectangle denotes the Kalman filter prediction. The blue ellipse is obtained by the CAMShift algorithm, and corresponds to the centroid of the respective back-projection which can be seen in (c).

repeated either a fixed number of iterations or until a certain convergence criterion is met, e.g., the search window does not shift more than ϵ, which is commonly set to 1 pixel. This iterative process is also depicted in Figure 4.3.

The contribution of Bradski [1998] is to adopt this general algorithm for the purpose of object tracking. Therefore it is necessary to transform the input image into a probability distribution image where the pixel intensities reflect the likelihood that they belong to the tracked object. Although many ways are conceivable to obtain such a transformation, CamShift utilizes the color information of the object in question for this purpose, by means of histogram back-projection (described in Section 3.1.3). The back-projection is re-computed every frame as is the current search window size, the objects size and, optionally, the objects orientation using image moments analysis. Thus the algorithm adapts to the various changes the tracked object can undergo due to, e.g., illumination changes, object articulations, and the like. Hence its name.

As can be seen in Algorithm 4.3.2 the mean can be computed efficiently using statistical image moments of 0th and 1st order. In general, the moment of order i, j is given by

$$m_{ij} = \sum_x \sum_y I(x,y) \cdot x^i \cdot y^j \tag{4.11}$$

Accordingly, the 0th order moment is computed as: $m_{00} = \sum_x \sum_y I(x,y)$ and corresponds to the area of the distribution. Together with the 1th order moments $m_{10} = \sum_x \sum_y xI(x,y)$, and $m_{01} = \sum_x \sum_y yI(x,y)$ the centroid of the distribution

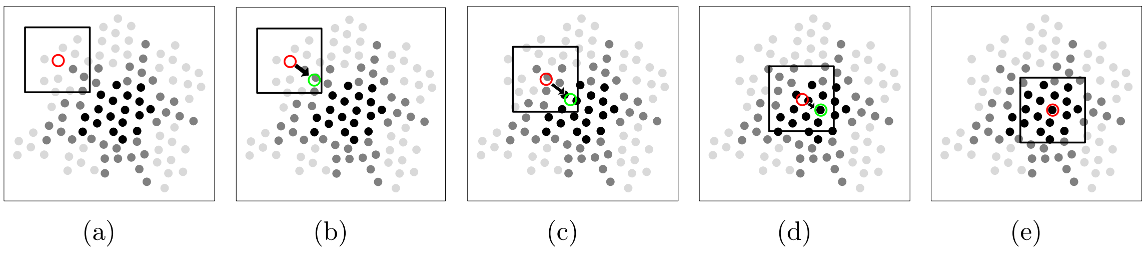


Fig. 4.3: The mean-shift algorithm. Dots signify probability values. The black rectangle depicts the search window, the red circle its current position (center), the green circle the new center. (a) The new mean within the search window is computed and found to be at the green circle. (b) The search window is shifted according to the arrow. (c) Centered at the previously computed mean (a) is repeated. (d) The search window is shifted again. (e) The new mean is computed but does not change anymore. The algorithm converged.

is calculated as in line 6 of the algorithm. After convergence is reached, the objects dimension and orientation is computed again using image moments. From the intermediate variables a, b, and cm

$$a = \frac{m_{20}}{m_{00}} - x_t^2,\ b = 2\left(\frac{m_{11}}{m_{00}} - x_t y_t\right),\ c = \frac{m_{02}}{m_{00}} - y_t^2, \tag{4.12}$$

we find the orientation Θ and the dimensions w_t and h_t of the tracked object as given by lines 10-12.

Our tracking algorithm (Alg. 4) is now easily extended by considering all *trackers* which have not been assigned a new observation. The center of the skin blob in the vicinity of the expected position for the corresponding hand is found by means of CAMShift and is used to update the trajectory as supplied before. However, we adjust the probability $\Pr(h|z = det)$ such that the corresponding belief $bel(h|z_{1:t})$ will get decreased slightly over time. This prevents tracking with CAMShift alone for too long a time period since then not a hand may be tracked but a similar colored false positive. Figure 4.2 illustrates the complete tracking process.

Of course, it is possible that tracking with CAMShift also fails. Strategies do deal with the most common reasons are therefore presented in the following section.

4.3.3 Tracking Failures

Basically, there are three cases that can lead to tracking failure: the object moved outside of the cameras' field-of-view, it got occluded, or it moved or changed its direction

Algorithm 5 CAMSHIFT

```
Input: $\Omega$ the probability distribution image
 1:    $sw$ search window $(x_s, y_s, w_s, h_s)$
Output: target window $tw$ with $(x_t, y_t, w_t, h_t)$ and $\Theta$ the orientation
 2: $c_1 \leftarrow (x_s, y_s)$
 3: while $\neg converged$ do
 4:     $c_2 \leftarrow c_1$
 5:     calc new position using image moments
 6:     $c_1 \leftarrow (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}})$
 7:     $converged \leftarrow |c_2 - c_1| < \epsilon$
 8: $x_t = c_1^x,\ y_t = c_1^y$
 9: $\Theta = \frac{1}{2} \tan^{-1} \left( \frac{b}{a-c} \right)$
10: $w_t = \sqrt{\frac{(a+c)+\sqrt{b^2+(a-c)^2}}{2}}$
11: $h_t = \sqrt{\frac{(a+c)-\sqrt{b^2+(a-c)^2}}{2}}$
```

of motion too rapidly.

Outside field-of-view This case is detectable straightforward as the last known position should be close to the image borders and together with the prediction of the Kalman filter, which should indicate an even closer or already outside field-of-view position, one can safely assume that the tracked object indeed moved outside of the field-of-view. As a simple strategy, we reset the Kalman filter as we have no knowledge about the objects' further motion. Again, the probability $\Pr(x|z = det)$ used to update the belief $bel(x|z_{1:t})$ (x signifying either hand or face) is modified such that the tracked object is considered to be lost after a reasonable time ($\approx 1s$), i.e., the probability dropped below the threshold, and the associated tracker is deleted.

Occlusion Without an indication for an out-of-view state the object might have become occluded. Currently, we use the Kalman filter predictions to be able to re-detect the object when it reappears. Of course the objects motions might change during occlusion. In this case the object is declared as lost and the corresponding tracker is deleted.

In the domain of hand tracking, however, we consider one important special case which is observable frequently: one hand covers the other, partially or completely. To detect and resume tracking at the end of such an occlusion is crucial for assigning subsequent hand detections to their respective tracker. Inspired by the work of Shamaie

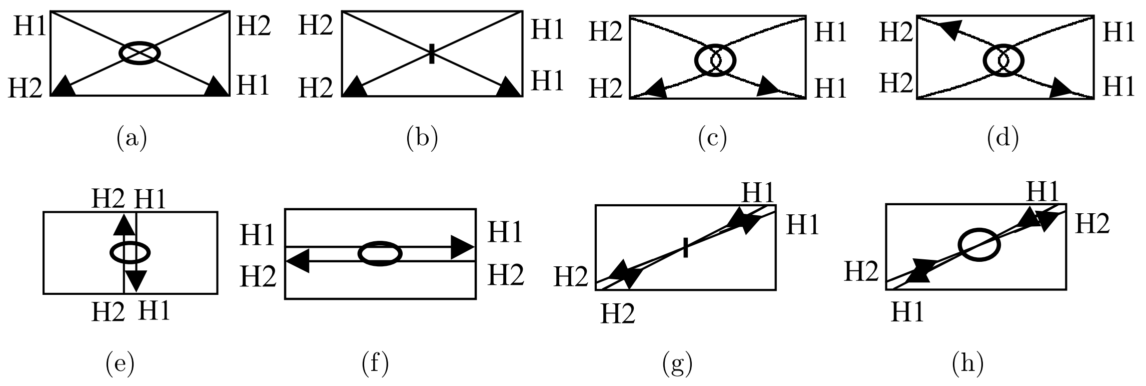


Fig. 4.4: The path of the hands in the 8 types of the bimanual movements. The thick ellipses represent the occlusion areas (a, c, d, e, f, and h), and the small lines represent collision (b and g).

& Sutherland [2003], we present a solution which is able to to accurately handle the described situation.

The general idea is to determine whether the two hands pass each other as shown in Figure 4.4 (a), (e), (f) and (h) or collide and return in opposite direction, (b) and (g) in the same figure. In some cases they may not collide but return in opposite direction as in (c) and (d). Such a collision of hands obviously involves a (short) pause of hand movements, or, in other words, the velocity of hands approaches zero. By detecting this short pause and comparing the hands position at the end of occlusion with their position prior to occlusion one can resolve the correct position of each hand.

To achieve this we first have to detect the occlusion case at the first place. For this, we rely on the Kalman filter predictions: once they indicate that the hand rectangles might intersect the next time step, an occlusion alarm is raised and a large rectangle is formed as the union of the two hand rectangles. Within this region the skin color blobs are monitored as to catch exactly the moment when they merge. Starting from this moment the velocities of the horizontal and vertical sides of the bounding box are analyzed according to following measurements

$$v_v = \sqrt{v_{h1}^2 + v_{h2}^2},\ v_h = \sqrt{v_{v1}^2 + v_{v2}^2} \tag{4.13}$$

with $h1$ and $h2$ denoting the horizontal sides and $v1$ and $v2$ the vertical sides of the rectangle respectively. Once we observe that one of these measures drops below a certain threshold ϵ we conclude that the hands collided and therefore will move back again to were they came from. However, in some movements as in Figure 4.4 (e) and (f), a horizontal or vertical pause may be detected, although no collision occurs. To

Algorithm 6 ResolveOcclusion

```
Input: measurements $v_v$, $v_h$, $s_v$, and $s_h$
Output: position of hands after at occlusion end
 1: if $s_v < \epsilon_1$ then
 2:     if $v_h < \epsilon_2$ then
 3:         hands horizontally back
 4:     else
 5:         hands horizontally pass each other
 6: else if $s_h < \epsilon_1$ then
 7:     if $v_v < \epsilon_2$ then
 8:         hands vertically back
 9:     else
10:         hands vertically pass each other
11: else if $v_h < \epsilon_2$ then
12:     hands horizontally back
13: else if $v_v < \epsilon_2$ then
14:     hands vertically back
15: else
16:     hands pass each other
```

detect such cases, the measurement s_v and s_h are defined

$$v_v = \text{dev}(|v_{h1}| - |v_{h2}|),\ v_h = \text{dev}(|v_{v1}| - |v_{v2}|) \tag{4.14}$$

as the standard deviation of the velocity differences of the respective rectangle sides. Occlusions can finally be resolved according to a simple algorithm listed in Algorithm 6.

4.4 Establishing People

In this section, we describe how we assign hands to corresponding faces thus forming our compact person model. In the simplest case, there is only one person observed and up to two hands. In case these are within the action range of the person assignment is possible ad hoc, utilizing the knowledge of each hands' laterality. In more challenging settings ambiguities are possible and in order to resolve them, we propose a probabilistic method to determine the assignment costs. The corresponding assignment is found using yet again the Hungarian method.

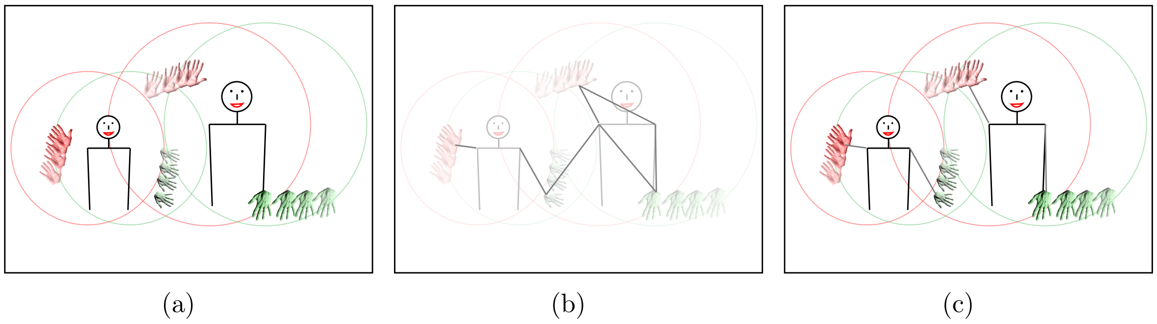


Fig. 4.5: The assignment problem with four hands and two faces. Indicated are moving hands colored according to their laterality (red = left, green = right) with the intensity signifying the confidence. In (b) all possible assignments are plotted, considering the action area as defined in Chapter 3. In (c) the resulting assignments are depicted which are as desired.

We proceed by sorting all currently tracked hands into two lists, one for the left hands, i.e., $bel(h_l|z_{1:t}, h) > 0.5$, and one for the right hands $(bel(h_r|z_{1:t}, h) > 0.5)$ respectively. In doing so, we divide the problem into two parts, each of which is solved in the same way. We therefore consider the assignment of hands in one list only, e.g. assign left hands to the persons' "left-hand-slots".

Let n be the number of persons currently observed. For each hand in our list we maintain a 1D histogram with $n+1$ bins, with b_i storing how often the hand was assigned to person p_i, or has been left unassigned (b_0). Using this histogram, we maintain an assignment of a particular hand to a person over time. We then define the assignment costs c_{ij} as

$$c_{ij} = \frac{1}{\text{hist}(h_i, p_j)} + c_{\text{dist}}(h_i, p_j). \tag{4.15}$$

Here, $\text{hist}(h_i, p_j)$ denotes the normalized bin value for person p_j of hand h_i, such that assignment costs increases as the histogram value decreases. The second term, distance costs $c_{\text{dist}}(h_i, p_j)$, evaluates the geometric relationship between hand h_i and person p_j and is given as

$$c_{\text{dist}}(h_i, p_j) = \begin{cases} max & \text{if } dist\_x(h_i, p_j) > \text{action radius} \\ max & \text{if } \left|size(h_i) - size(p_j)\right| / size(p_j) > c_1 \\ c_2 \cdot \text{dist\_x}(h_i, p_j) & \text{otherwise} \end{cases} \tag{4.16}$$

Here, the action radius refers to maximum arm length as introduced in Chapter 3, and $dist\_x(h_i, p_j)$ denotes the distance between hand h_i and person p_j in x direction. With the second condition we incorporate our knowledge that the ratio of the observation size of the hand and the face of a person must be within c_1. c_1 corresponds to the number of scalings we allow the hand observation size to differ from the face observation size (see Chapter 3.

Maximal costs are given for assigning a hand to an unknown person, i.e., let it left unassigned. If the hand is out of reach, assignment costs are also maximum.

The maximum costs *max* are chosen in dependence of n and a small proportion of the distance to the face of person p_j,

$$max = \frac{1}{c \cdot \frac{1}{n+1}} \tag{4.17}$$

This ensures that in the beginning the hands are assigned to the closest person and do not change assignment immediately, once a new person is recognized.

The optimal assignment is found via the Hungarian method and the histogram bin values are increased accordingly.

5 Head Pose Estimation

To estimate the head orientation we use a system developed as part of an earlier masters' thesis which has been adapted to suit the needs of our work. This chapter describes the system according to [Vatahska, Bennewitz, & Behnke 2007] and the extensions conducted.

For us, knowing the head's pose is of twofold interest. By continuously observing the current head orientation we can derive important head gestures, most notably, head shake and head nod. Additionally, we will assume that a person who is pointing at something will also be looking at the target. The knowledge of the respective angles can compensate for our lack of 3D information, as will be seen later.

Head pose can be described by the heads three degrees of freedom, that is, by rotational angles about three axis orthogonal to one another, commonly known as roll Θ_x, pitch Θ_y and yaw Θ_z. The general idea is to locate the facial features and, by means of the relative positions, to infer the 3D head orientation. The mapping between facial feature positions and corresponding head orientation is then learned from suitably labeled data using a function approximator; in our case neural networks are applied.

5.1 Locating Facial Features

Prior to locating facial features, a face detection process is conducted. Again, the object detection scheme by Viola & Jones [2001] is employed. This time, two classifiers are being used, one for frontal faces and one for left profiles (right profile when flipping

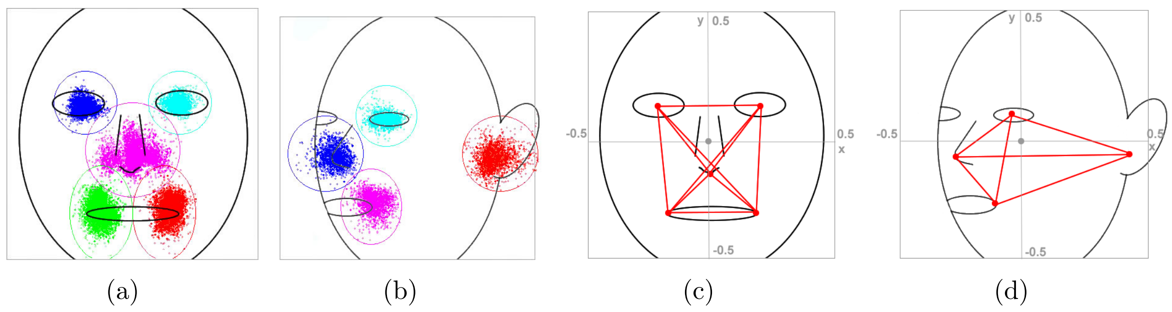


Fig. 5.1: Facial features in frontal faces and profiles. (a) and (b) show the distribution of facial feature points in a normalized face bounding box. (c) and (d) show the input features used for the neural network. Pictures taken from [Vatahska, Bennewitz, & Behnke 2007]

the image). Both classifiers cover roll rotations within the interval $\Theta_x \in [-25°, +25°]$ and pitch rotations $\Theta_y \in [-40°, +40°]$. The yaw angle of frontal faces can be within $\Theta_z \in [-40°, +40°]$, whereas with profiles $\Theta_z \in [30°, 90°]$. Note, that the two classifiers already achieve a rough classification in terms of the yaw angle. Moreover, they allow to use two different sets of facial feature points to describe the pose. In a frontal face, we expect five features to be detectable, namely two eyes, one nose and two corners of the mouth. In profile faces, four features are used, which are again the nose as well as the eye, the mouth corner, and the ear of the respective side, see also Figure 5.1.

Locating these features is achieved by nine individual classifiers that have been trained with the Viola & Jones [2001] algorithm. To speed up the detection process the classifiers are applied only within their corresponding search areas which have been empirically determined by plotting the feature positions of the (manually labeled) training data, normalized w.r.t. the face bounding box (Figure 5.1 (a) and (b) illustrate the obtained distribution).

To deal with multiple detections and to effectively diminish the influence of outliers a mean-shift algorithm is applied to cluster the set of detections into a single mean position. The mean position of all detections is refined by determining those detections which are too far away from the mean position and then recomputing the mean on the remaining set of detections. This is repeated until the change of the mean position is less than some threshold ϵ.

5.2 Estimating Pose

We observe that under pose change the distances between the facial features change accordingly due to the 2D image projection of the face. Further, the feature positions within the face bounding box are affected. For instance, when lowering the head, the distance between an eye and the nose increases whereas the distance between a corner of the mouth and the nose decreases. The positions within the face bounding box altogether move downward. These correlations between head rotation and facial feature positions are thus learned automatically from appropriately labeled data. This corresponds to finding a function that maps the positions to the respective 3D orientation angles. Such a function can be approximated using machine learning techniques such as neural networks, which is what is used here.

Two neural networks are being trained according to the two different facial feature sets. The feature positions are normalized w.r.t. the face bounding box center, such that $x_{fp}, y_{fp} \in [-0.5, +0.5]$. In addition to the positions, the distances between features are also taken as input values for the neural network, as depicted in Figure 5.1 (c) and (d). The neural network is structured as follows: the output layer consists of three units describing the three rotation angles Θ_x, Θ_y and Θ_z respectively. A single hidden layer with six units connects the output layer with the input layer containing 30 units (frontal face) or 20 units (profile).

For training the networks appropriate training data is needed, i.e., images with arbitrary rotated heads, labeled feature positions and the corresponding angles. The exact measurement of the rotation angles, however, is a challenging task. Moreover, to sufficiently cover the search space several thousand images should be collected. To overcome this problem, synthetic face images are generated by rendering a 3D head model which is rotated around the three axis at will. Resilient back-propagation is used to train the network with the data.

This completes the pose estimation system. When applying it, the facial features are detected, the input values for the neural net computed and the test performed which yields the estimated angles.

5.3 Extensions

To apply the pose estimation system within the scope of our gesture recognition application several improvements have been conducted. We found that for our purposes the estimation was not precise enough and that the estimated values were subject to large variations. Furthermore, the detection rate of facial features drops significantly in real images which may be due to the low quality, as this behaviour can especially be observed when the person is far away from the camera.

Our first contribution was the introduction of Kalman filters for both the facial feature positions and the estimated angles. For this reason, we are able to compensate for detection failures of the facial features by taking the corresponding Kalman filter

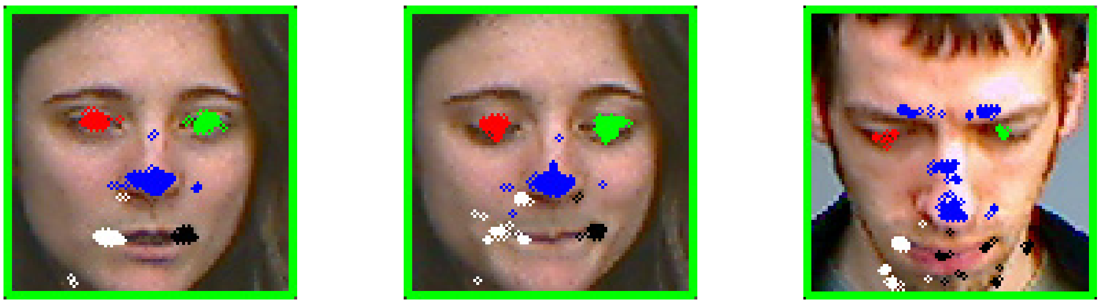

(a) (b) (c)

Fig. 5.2: Feature detection results. In (a) the normal case where all detection cluster into one point. (b) Two clusters are found, one at the nose and one at the left mouth corner. (c) In rare cases (depending on the head pose) the detection results are littered all over.

prediction as input for the neural network. Of course there might be grave reasons why a facial feature does not get detected for a while, e.g., it is occluded. To appreciate this fact, the Kalman filters are continuously reset after only a small number (3-4) of frames. By filtering the estimated angles, we counteract the variations such that we obtain a smooth angle trajectory.

When scrutinizing the detection failures, it became evident that clustering the set of detections of some features (mainly the mouth corners) into a single point failed above average. When plotting the set of features (Figure 5.2), one can observe that in case of 5.2(b) actually two clusters are formed: one at the expected mouth corner and the other at the nostrils. As this happens rather frequently, there are rare cases when even more clusters are formed, affecting other features, too. We therefore developed the following strategy do deal with this problem: in case clustering into a single cluster fails, we repeatedly employ a k-means algorithm with increasing k to find the clusters. We thereby took advantage of the recent work of Arthur & Vassilvitskii [2007], where a k-means algorithm is presented with a simple, randomized seeding technique that improves both the speed and the accuracy of k-means dramatically.

To find the optimal number of clusters which best explains the distribution of observation points, we use the Bayesian information criterion (BIC) which can be computed as

$$BIC = n \ln \left(\frac{RSS}{n} \right) + k \ln(n), \tag{5.1}$$

where n denotes the number of observations, k the number of clusters, and RSS the residual sum of squares. k is chosen such that the values of BIC are lowest. The cluster which is closest to our Kalman filter prediction in terms of an Euclidean distance measure is chosen as our final feature position.

Furthermore, we introduce a detection quality score q_i for each facial feature i as

$$q_i = \frac{1}{n_c} + \frac{obs_c}{obs_{total}}, \tag{5.2}$$

with n_c denoting the number of clusters found, obs_c its support in relation to the total observation data. q_t is used to alter the measurement noise covariance matrix of the Kalman filter for feature i, such that for low values higher noise is assumed and vice versa.

Similarly, an overall estimation quality score $\mathbf{q}$ is defined as the sum of all detection quality scores

$$\mathbf{q} = \sum^{K} q_i. \tag{5.3}$$

This is used to adapt the measurement noise covariance for the Kalman filters for the angle estimation in the same manner as before.

One problem, which can not be diminished easily, is that the estimated angles for the profile case are much worse as compared to the frontal ones. This is mainly inherent as Θ_z and especially Θ_x rotations do not result in much change of the projected positions onto the image plane anymore. However, we observed that the position of the face bounding rectangle from the profile face detection process was prone to arbitrary shifts although the face or its position did not change accordingly. As the facial feature positions within the face bounding box rectangle get more and more weight the more the face is rotated in Θ_z direction, this results in variations of the estimated angles. To stabilize the face bounding box we make use of the back-projection of Chapter 3 by calculating the centroid of the skin color distribution within the face bounding box. Applying Equation 4.11, the center is given by

$$c = (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}). \tag{5.4}$$

Centering the face bounding box at c makes us independent from the position provided by the face detection thus diminishing another source of impreciseness.

Finally, we also performed extensive experiments with different configurations for the neural network used. In particular, we evaluated different activation functions for perceptrons of the hidden layer. Here we found the sigmoid or logistic function

$$y = \frac{1}{1 + e^{-x}} \tag{5.5}$$

to perform best.

6 Recognizing Gestures

In this chapter, we address the last and actual step in recognizing gestures. Knowing the trajectories of faces and hands, the challenge is to discover meaningful structures and reject non-meaningful motions. What is meaningful and what is not is up to the designer and has to be decided beforehand. In our work, we focus on a small set of gestures which are of importance to our interaction scenario:

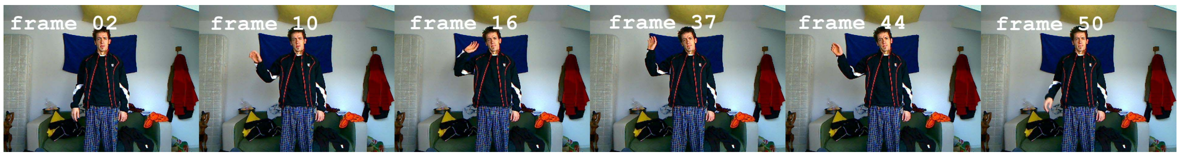
waving Common gesture used at the beginning or end of an interaction, but also to get the attention of someone. See Figure 6.1 (a).

pointing This parametric mono-manual gesture is carried out to draw the attention of someone to something. This means, in addition to recognizing the pointing gesture, the target has to be estimated by, amongst others, using the head pose estimation. For an illustration see Figure 6.1 (b).

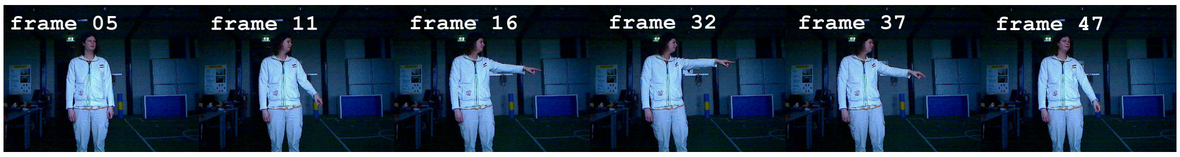
thisbig Inspired by [Wilson & Bobick 1999], we include an iconic gesture which is used to convey information about the size of an object, i.e., a parametric bimanual gesture which depends on the distance between the two hands. See Figure 6.1 (c) for an example.

dunno This bimanual gesture is used to express ignorance, as shown in Figure 6.1(d). (*dunno*: informal short for *don't know*).

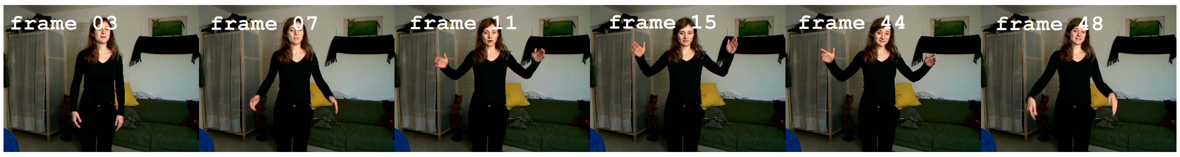
head nod/shake Using the head pose estimation data alone, we also recognize the commonly used head gestures nod and shake, see Figure 6.1(e) and 6.1(f).



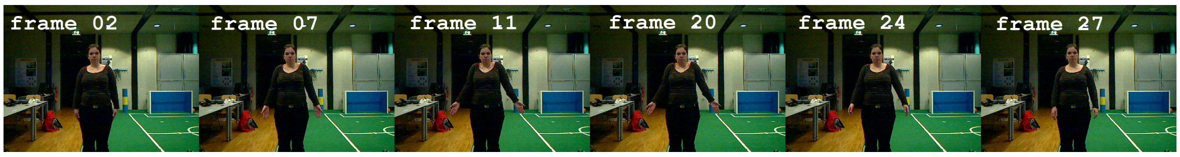
(a)



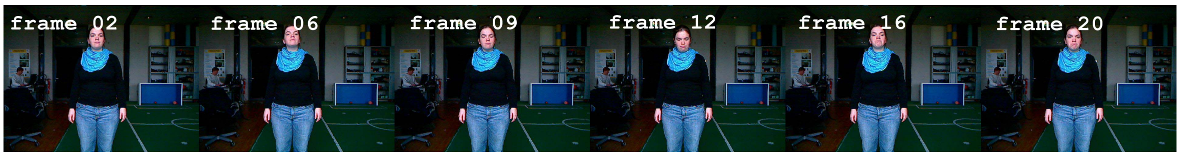
(b)



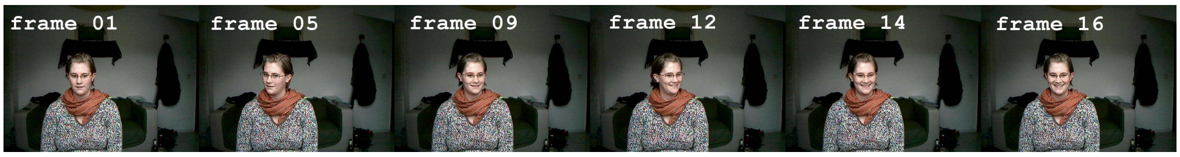
(c)



(d)



(e)



(f)

Fig. 6.1: The gestures that are recognized in this work: (a) waving, (b) pointing, (c) thisbig, (d) dunno, (e) nod and (f) shake

Each such gesture has a specific spatio-temporal pattern that might vary from execution to execution and also from person to person, or is dependent on a parameter. Yet there is an essence that remains unique. Provided with a sufficient number of examples, this essence can be learned. For this purpose, statistical modeling techniques are usually applied and we chose hidden Markov models (HMMs), introduced in Section 6.2, to accomplish this task.

At first, however, we address the problem of how to represent the motion data. Besides coordinate transformations to obtain translational and scaling invariance, other transforms can be applied to emphasize the characteristics of the gesture and/or to diminish non-meaningful variability which could hamper the recognition process. This *feature extraction* process is described in Section 6.1. A few words on modeling and training the individual gestures (and non-gestures) are provided in Section 6.3. We discuss the application of our models to finally recognize gestures and to extract the optional parameters in the final section.

6.1 Feature Extraction

So far, the tracking module continuously provides the positional data $\mathbf{x_r} = (x, y)$ of faces and hands in the image sequence relative to the robot/camera. We are, however, interested in hand movements in reference to the person they belongs to. As a first step, we therefore transform the hand coordinates into a *egocentric* reference system which is straightforwardly done by subtracting the face coordinates:

$$\mathbf{x} = \mathbf{x}_r^{hand} - \mathbf{x}_r^{face}$$

With this we acquire translational invariance as it does not matter anymore where in the image the person is located or whether she moves in-plane while performing the gesture or not.

As a second step, the coordinates are normalized with respect to the face bounding box size, thus making them independent from the distance between person and robot. Note that, however, the face bounding box size is subject to jitter which introduces unwanted jumps to the scaled hand trajectory. We therefore use a Kalman filter to track the size of the bounding box, and in doing so smooth the changes in size. The preliminary gesture trajectory is then described as a sequence of the respective position data:

$$\mathbf{g}_{mono} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n),$$

for monomanual gestures and

$$\mathbf{g}_{bimanual} = ((\mathbf{x}_1^l, \mathbf{x}_1^r), (\mathbf{x}_2^l, \mathbf{x}_2^r), \ldots, (\mathbf{x}_n^l, \mathbf{x}_n^r)),$$

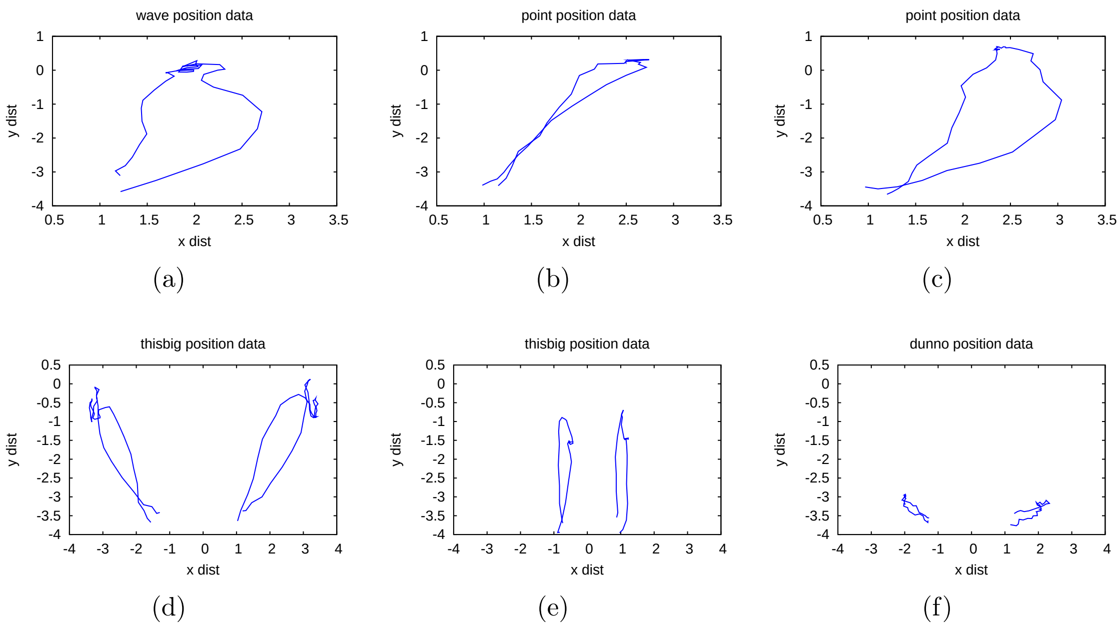


Fig. 6.2: Example trajectories of the hand gestures transformed into egocentric coordinates and normalized. (a) The waving gesture. (b) The pointing gesture. (c) Another pointing gesture, with a different trajectory, due to the parameter (target location). (d) The thisbig gesture. (e) Another thisbig gesture, with different trajectory, due to the parameter (distance between hands). (e) The dunno gesture.

for bimanual gestures, respectively, with l denoting the left hand, r the right hand, and n the sequence length.

Similarly, we can define a head gesture trajectory, by using the continuous head pose estimation $\mathbf{\Theta} = (\Theta_x, \Theta_y, \Theta_z)$ as of Chapter 5:

$$\mathbf{g}_{head} = (\mathbf{\Theta}_1, \mathbf{\Theta}_2, \ldots, \mathbf{\Theta}_n)$$

Obviously, the angles can be used directly without any transformation as the available observation data is inherently egocentric and invariant toward scaling.

Figure 6.2 illustrates the transformed gesture sequences, and Figure 6.3 the untransformed angular sequences considered in this work.

Although these transformations can be sufficient for recognition purposes, one usually tries to find a better suited description of the motion data. Features are selected which represent best the characteristics of the respective gesture and, at the same time, show more robustness toward variations. As the amount of training data is usually limited,

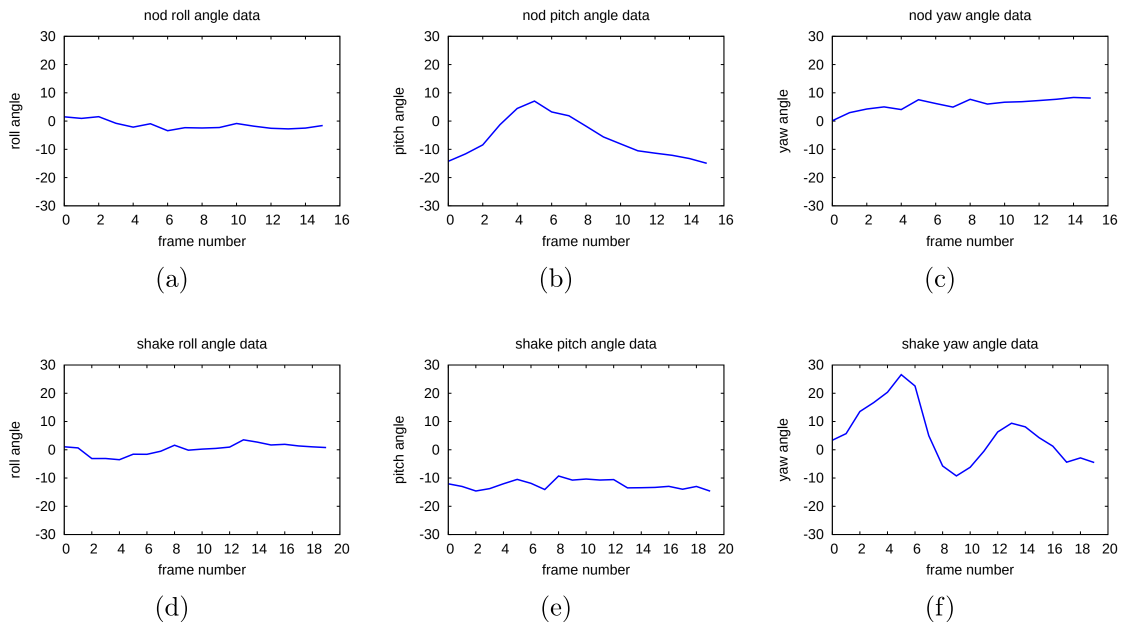


Fig. 6.3: Example angle trajectories of the head gestures. All three angles roll, pitch and yaw are shown, in this order. (a) and (d): The roll angle stays around zero, as expected. (b) and (e): The pitch angle's idle position is at $-10°$, yet the head nod is clearly noticeable. (c) and (f) We see that the head shake was actually performed twice, whereas during the head nod yaw does not change significantly.

we cannot cover all possible variations of a particular gesture. That is why features are beneficial.

If we for example consider the *waving* gesture in Figure 6.2(a), we observe that in this case the hand, while moving up, describes a wide arc. However, others might just lift the hand up straight. This leads to the realization that this arc may not be of substantial meaning in terms of the gesture. Instead, the fact that the hand moves up is sufficient. As a naïve solution one could drop the x-coordinate. However, we should not extract too specific features as other gestures may reveal characteristics that depend precisely on the x-coordinate. Thus, determining the best features is a non-trivial task and depends largely on the gestures to be described. We therefore consider in the following each of the three gesture groups - monomanual, bimanual, and head gestures - separately.

6.1.1 Monomanual Gestures

Monomanual gestures are by far the most common and have therefore received wide attention within the gesture recognition research community. Consequently, others have investigated the question of how to represent them in feature space, for example in [Yoon et al. 1999] and, more recently, in [Montero & Sucar 2004] with similar results. On the basis of raw position data, features are derived such as Cartesian velocity, polar velocity, or angular velocity. In addition, chain codes to describe changes in direction, mesh codes which work similar to histograms, or trajectory momenta are used. Yoon et al. [1999] show that all of these features are grounded on three basic information (r, ϕ, v), i.e., the relative distance of the hand to the coordinate origin, r, the angle obtained between each gesture point and the origin, ϕ, and the Cartesian velocity, v.

Based on this, but opposed to the work of Yoon et al. [1999], where the origin is defined to be the center of the gesture trajectory (which is unknown until the end of the gesture which in turn prohibits online recognition) we conveniently choose the face as the center. With it, r is computed as

$$r = \|\mathbf{x}\| = \sqrt{x^2 + y^2}, \tag{6.1}$$

and the angle ϕ as

$$\phi = \mathrm{atan2}(y, x). \tag{6.2}$$

Note that these transformations correspond to the general transformation of Cartesian coordinates into polar coordinates. Finally, v is computed as the distance between two successive trajectory points:

$$v = \|\mathbf{x}_t - \mathbf{x}_{t-1}\| \tag{6.3}$$

Together, these three measures define our final feature vector

$$\mathbf{f}_{mono} = [r, \phi, v]\,. \tag{6.4}$$

Figure 6.4 shows the feature trajectories obtained by transforming the gesture trajectories depicted in Figure 6.2(a), 6.2(b), and 6.2(c), respectively.

6.1.2 Bimanual Gesture

A bimanual gesture could straightforwardly be represented as the combination of features for both the left and the right hand. As we will do this for comparison reasons (see Chapter 7), we can exploit the relationship between the two hands to perform better. Another problem is that the gesture *thisbig* is governed by a systematic variability, i.e., the distance between the hands during the hold phase. Depending on this

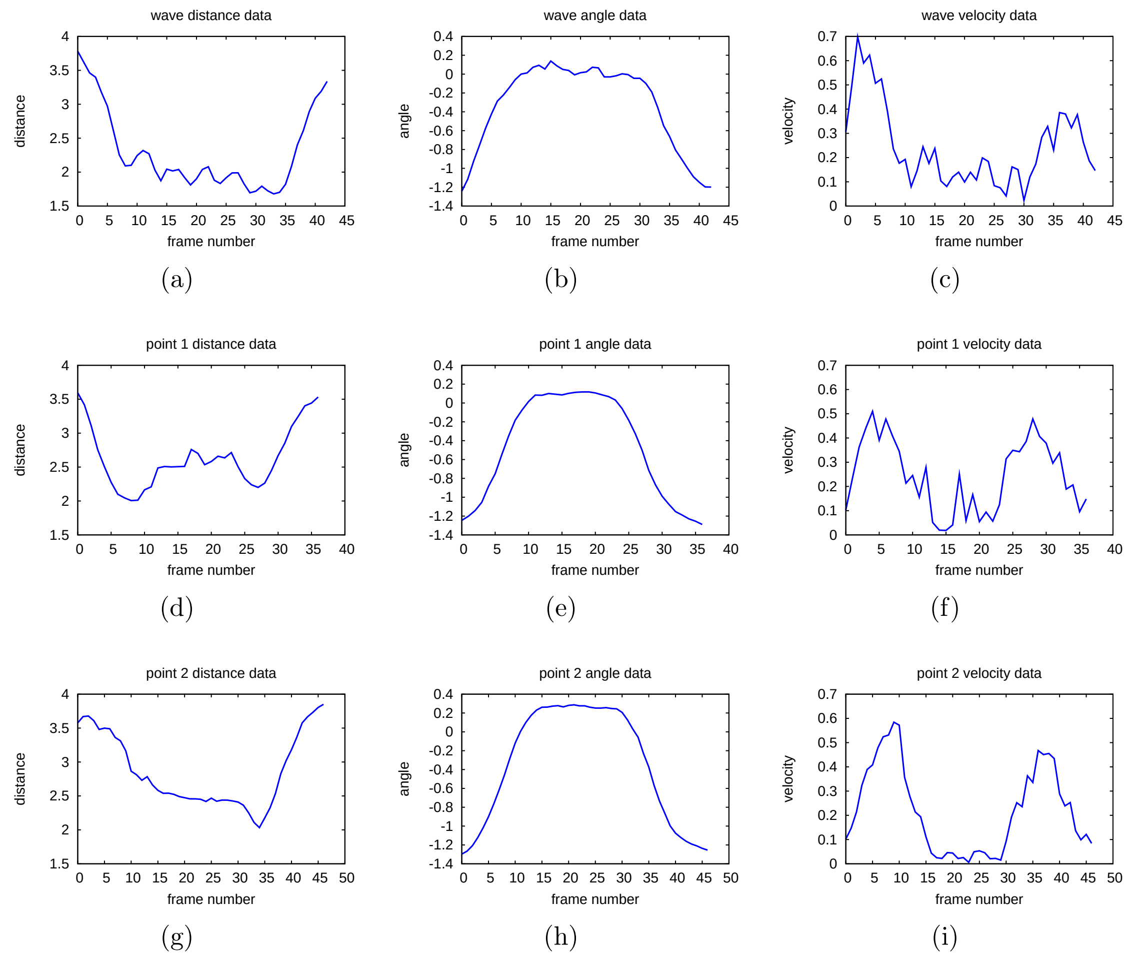


Fig. 6.4: Feature trajectories for the gestures *wave* and *pointing* from Fig. 6.2(a), 6.2(b) and 6.2(c), respectively. The angular feature is quite smooth as opposed to the velocity. Note that both pointing gestures look under the feature transformation much more similar than before, as desired.

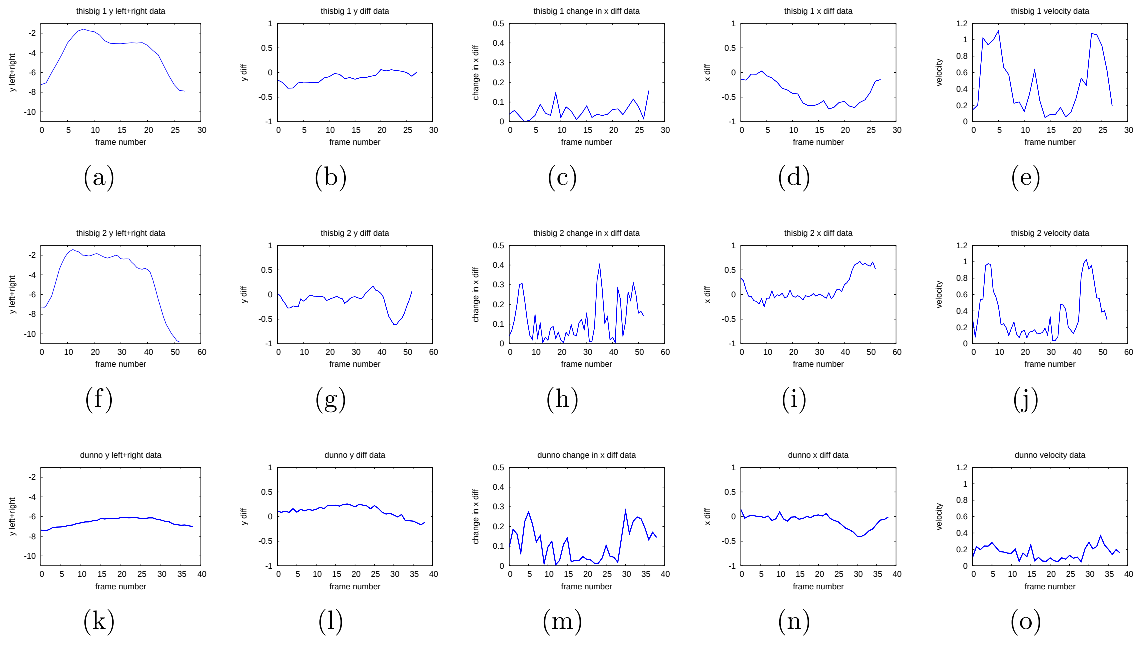


Fig. 6.5: Feature trajectories for the gestures *thisbig* and *dunno* from Fig. 6.2(d), 6.2(e) and 6.2(f), respectively.

parameter, see Figure 6.2(d) and 6.2(e), the observed trajectories are far from being similar. Through appropriate feature selection it is possible to diminish the influence of this parameter on our feature stream. This should yield robust feature trajectories which are likelier to be recognized as being an instance of *thisbig*.

Our feature transformation is inspired by the observation that bimanual gestures are inherently synchronized, which leads to a symmetric execution. Thus, we expect the differences of the hand position to vary around 0 for most of the execution time. This leads to the features

$$d_x = \left|x_t^l\right| - \left|x_t^r\right|, \text{and} \quad d_y = \left|y_t^l\right| - \left|y_t^r\right|. \tag{6.5}$$

Clearly, both these features would perfectly describe the state of idleness, too. What is missing are the dynamics of the motion. We observe that both gestures involve a lift-up of both hands followed by, after a short pause during the hold phase, a downward motion. This is included by

$$y^{lr} = y_t^l + y_t^r. \tag{6.6}$$

In x-direction we have the source of the variability, the x distance between hands can be either small, medium or large. Thus, it is not appropriate to include this information directly. However, we want to hint that the hands are moving x wise whilst d_x is not changing much. Therefore we incorporate

$$\Delta d_x = \left(\left|x_t^l\right| + \left|x_t^r\right|\right) - \left(\left|x_{t-1}^l\right| + \left|x_{t-1}^r\right|\right). \tag{6.7}$$

Finally, the motion information per se, the velocity of the hands, is considered as

$$v^{lr} = v_t^l + v_t^r, \tag{6.8}$$

with v_t according to Equation 6.3. We can then again specify the final feature vector as

$$\mathbf{f}_{bimanual} = \left[d_x, d_y, y^{lr}, \Delta d_x, v^{lr}\right]. \tag{6.9}$$

Figure 6.5 shows the feature trajectories obtained by transforming the gesture trajectories depicted in Figure 6.2(d) and 6.2(e), and 6.2(f), respectively.

6.1.3 Head Gestures

In case of head gestures, we do not expect excessive or systematic variability. A nod for example should always start with a small pitch angle which increases and decreases again, possibly repeated a few times. Similar holds for the shake gesture, only with the yaw angle affected this way. As we cannot make sure that the user is always looking straight into the camera, the initial condition might not hold all the time. We therefore consider additionally angular velocities, which are independent of the actual angles. As such our final feature vector becomes to

$$\mathbf{f}_{head} = \left[\Theta_x, \Theta_y, \Theta_z, \Delta\Theta_x, \Delta\Theta_y, \Delta\Theta_z\right]. \tag{6.10}$$

For the angle velocity trajectories of head gestures in Figure 6.3, see Figure 6.6.

This completes our meditation on feature extraction and we will derive the basics of hidden Markov models, which are used to learn the gestures, in the following.

6.2 Hidden Markov Models

Hidden Markov models (HMMs) are an important tool for discovering structure in time-varying data. Many real-world data may be considered generated by a physical process which can switch between a number of different states. Which state s_i the

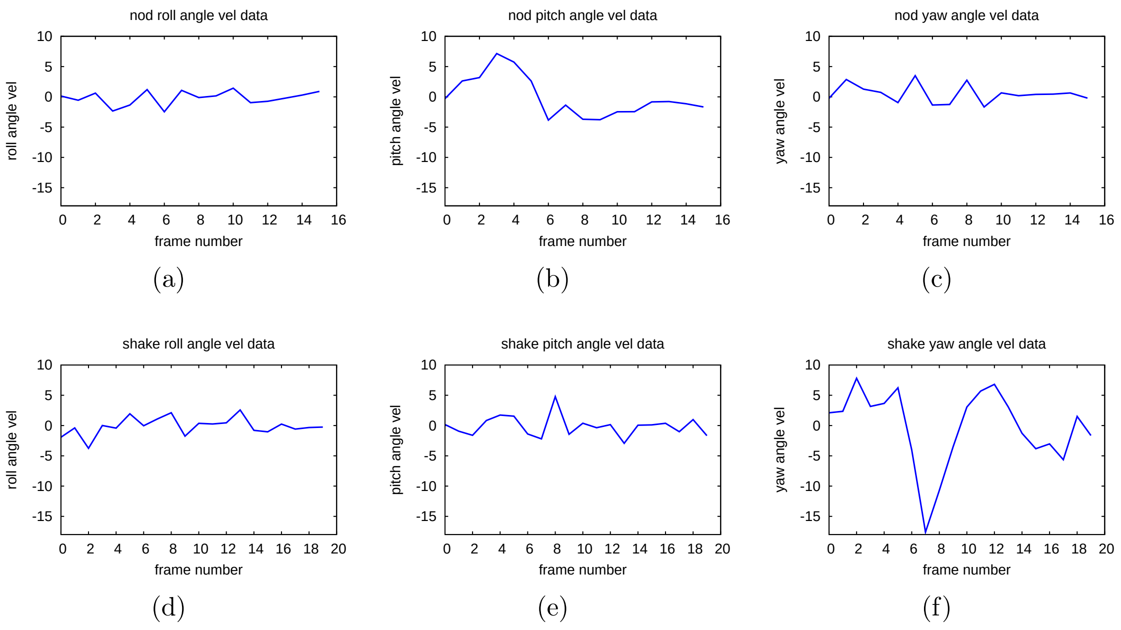


Fig. 6.6: Example angle trajectories of the head gestures as is. All three angles roll, pitch and yaw are shown, in this order. (a - c) The nod gesture. (d - f) The shake gesture.

process is in, at any given time $t+1$, depends only on the state at some previous time t. This is termed a (first order) Markov process, i.e., the Markov assumption holds:

$$\Pr(x_{t+1} = s_i | x_1, \ldots, x_t) = \Pr(x_{t+1} = s_i | x_t) \tag{6.11}$$

The signals observed result from this process, but the process itself and its state changes are not observable. As the (hidden) physical process makes a path from one state to another, observations with different characteristics are emitted. The characteristics are often probabilistic relationships between the underlying state and the data generation process. Consequently, two identical state paths may produce two different output sequences. For example, an identical word uttered by two different people will not sound the same due to differences in accent, sex, age, background noise etc., but the underlying phonetic ordering (i.e. state path) is identical. To extract and classify corresponding information from such signals, one must recover the underlying state dynamics which give rise to the observed data. HMMs are used to model the generation process in order to try and recover the hidden states underpinning the observed sequence.

Hidden Markov Models have a long, successful history, especially in speech recognition. Since the early 1980s they have been intensively studied. A sound mathematical grounding together with the existence of efficient algorithms for training and analysis are strong criteria for their application. In [Rabiner 1990] a tutorial on HMMs is given which forms the basis of the following introduction.

Formally, a Hidden Markov Model is a finite state machine having a set of hidden states, $S = \{s_i\}$, $i = 1, \ldots, N$, an output alphabet (observations), $O = \{o_k\}$, $k = 1, \ldots, M$, transition probabilities, A, output (emission) probabilities, B, and initial state probabilities $\pi = \{\pi_i\}$. The state at time step t is denoted as x_t. As the states and observations are usually understood, an HMM is therefore defined as a triple: $\lambda = (A, B, \pi)$ with

- $A = \{a_{ij}\}$ and $a_{ij} = \Pr(x_{t+1} = s_j | x_t = s_i)$ denoting the probability to transition from state s_i to s_j,
- $B = \{b_i(o_k)\}$, denotes the probability to output the symbol o_k in state s_i, and
- $\pi_i = \Pr(x_1 = s_i)$ signifies the probability to choose s_i as initial state.

Three canonical problems arise in the context of Hidden markov models:

Evaluation problem Given a HMM λ and an observation sequence O determine $\Pr(O|\lambda)$, i.e., the probability that λ produced O. An efficient solution to this problem is known as forward-backward algorithm.

Decoding problem Given a HMM λ what is the 'best' hidden state sequence (in a maximum likelihood sense) that could have generated a given observation sequence O. This problem is solved by the Viterbi algorithm.

Learning problem Deals with the question how to estimate the model parameters (state transition and output probabilities) given an observation sequence O or a set of such sequences. The Baum-Welch reestimation algorithm provides the solution.

In the following the solutions to the three outlined problems are presented.

6.2.1 Evaluating HMMs

To compute the likelihood that a given HMM λ produces an observation sequence O, and given that the underlying state sequence X is unknown, one has to sum over all possible state sequences that could lead to the given observation sequence. This can be expressed as follows

$$\Pr(O|\lambda) = \sum_X \Pr(O, X|\lambda) = \sum_X \Pr(O|X, \lambda) \Pr(X|\lambda). \tag{6.12}$$

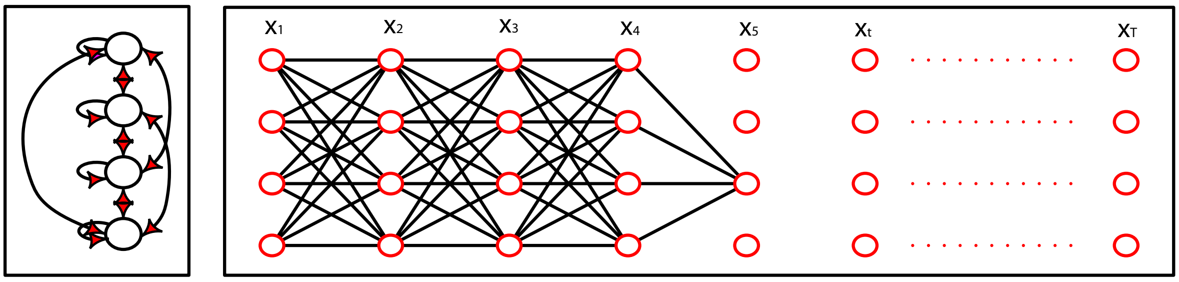


Fig. 6.7: Illustration of the lattice that is spanned to compute the forward variable $\alpha_t(5)$.

One brute force solution for this equation would be to exhaustively enumerate all possibilities:

$$\Pr(O|\lambda) = \sum_X b_{x_1}(o_1)b_{x_2}(o_2)\dots b_{x_T}(o_T)\pi_{x_1}a_{x_1x_2}a_{x_2x_3}\dots a_{x_{T-1}x_T} \tag{6.13}$$

However, this leads to combinatorial explosion with a complexity of $O(2T\cdot N^T)$, as there are N^T possible state sequences each requiring $2T$ computations. This is where the forward-backward algorithm comes into play, reducing the complexity to a reasonable $O(N^2 \cdot T)$.

The key idea of the algorithm is to span a lattice of N states and T time steps (that is, the length of the observation sequence) as in Figure 6.7. Then, the sum of probabilities over all paths coming to each state s_i at time t is stored. For this purpose, a variable $\alpha_t(j)$, the *forward* variable, is defined as

$$\alpha_t(i) = \Pr(o_1, o_2, \dots, o_t, x_t = s_i|\lambda), \tag{6.14}$$

and denotes the probability of the partial observation sequence $o_1, o_2, \dots, o_t$ at time step t in state s_i given the model λ. The advantage is that subsequent values can be computed recursively. Starting with the induction initialization

$$\alpha_1(i) = \pi_i b_i(o_1), \tag{6.15}$$

it holds for the subsequent induction steps that

$$\alpha_{t+1}(i) = \left\{\sum_{i=1}^{N} \alpha_t(i)a_{ij}\right\} b_j(o_{t+1}),\ 1 \le t \le T-1\ . \tag{6.16}$$

That means, we compute the probability that the system is in state s_j at time $t+1$. State s_j is reachable from all preceding states s_i with $a_{ij} > 0$. Therefore, $\sum_{i=1}^{N} \alpha_t(i)a_{ij}$

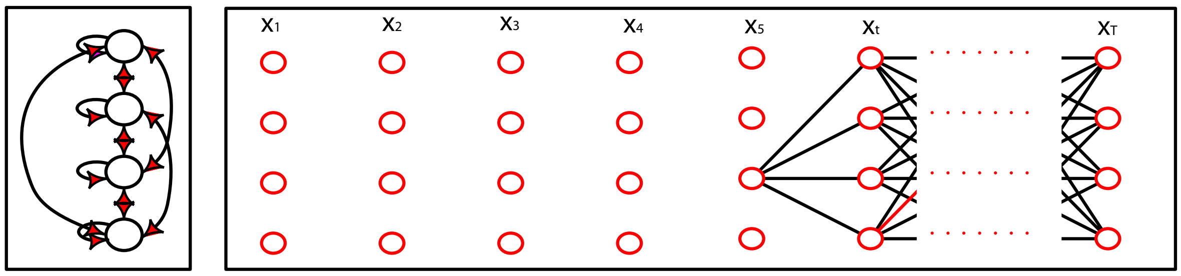


Fig. 6.8: Illustration of the lattice that is spanned to compute the backward variable $\beta_t(i)$.

denotes the summed probability for reaching state s_j from all preceding states. This term is multiplied with the probability of producing the next observation of our sequence in state s_j. This is repeated for all states of the system $1 \leq j \leq N$. If time step T is reached, the sum over all $\alpha_T(i)$ yields the probability of the observation sequence O at time T:

$$\Pr(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{6.17}$$

In analogy, it is also possible to define a variable $\beta_t(i)$, or *backward* variable, which in the end sums up to the equal probability of an observation sequence O, but computation starts from time step $t = T$. Initialization is given by

$$\beta_T(i) = 1, \tag{6.18}$$

whereas the actual recursion is computed as

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j). \tag{6.19}$$

Figure 6.8 illustrates the the computation of the backward variable.

In practice, only the forward algorithm is used to solve the evaluation problem, as it also allows to compute probabilities for partial observation sequences, i.e., can be used for online recognition. The backward algorithm, however, is useful for the learning problem as described in Section 6.2.3.

6.2.2 Decoding Hmms

The second problem deals with the analysis of the internal structure of the HMM and provides insight about internal segmentation of the pattern that is represented by the HMM. This can be useful for continuous recognition as we will see later.

There are several possible criteria for finding the most likely sequence of hidden states. One is to choose states that are individually most likely at the time when a symbol is emitted. This approach is called posterior decoding. Posterior decoding works fine in the case when the HMM is ergodic, i.e. there is transition from any state to any other state. If applied to an HMM of another architecture, this approach could give a sequence that may not be a legitimate path because some transitions are not permitted.

The Viterbi algorithm operates globally by choosing the best state sequence, called Viterbi path, that maximizes the likelihood of the state sequence for the given observation sequence. We define

$$\delta_t(i) = \max_{x_1, x_2, \ldots, x_{t-1}} \Pr(x_1, x_2, \ldots, x_t = s_i, o_1, o_2, \ldots o_t | \lambda) \tag{6.20}$$

to be the maximal probability of state sequences of the length t that end in state s_i and emit the t first observations for the given model. This probability can be calculated using a similar dynamic programming scheme as in the last section except that it uses maximization in place of summation at the recursion and termination steps. Additionally, it keeps track of the arguments that maximize $\delta_t(i)$ for each t and s_i, storing them in the $N \times T$ matrix ψ. This matrix is used to retrieve the optimal state sequence at the backtracking step.

The basis step is accordingly given by

$$\delta_1(i) = \pi_i b_i(o_1), \quad \psi_1(i) = 0 \tag{6.21}$$

In the recursion step the probability and the previous state of the most probable path coming to each state s_i at time t is computed and stored:

$$\begin{aligned} \delta_t(j) &= \max_{1 \le i \le N} \left[\delta_{t-1}(i) a_{ij}\right] b_j(o_t), \ 2 \le t \le T-1, && (6.22) \\ \psi_t(j) &= \arg\max_{1 \le i \le N} \left[\delta_{t-1}(i) a_{ij}\right], \ 2 \le t \le T-1\ . && (6.23) \end{aligned}$$

The algorithm terminates with the computation of the most likely path x_T^* and its probability P^*:

$$\begin{aligned} P^* &= \max_{1 \le i \le N} \delta_T(i) && (6.24) \\ x_T^* &= \arg\max_{1 \le i \le N} \delta_T(i) && (6.25) \end{aligned}$$

To reveal the actual state sequence, path backtracking is performed by means of the matrix ψ:

$$x_t^* = \psi_{t+1}(x_{t+1}^*), \ t = T-1, \ldots, 1 \tag{6.26}$$

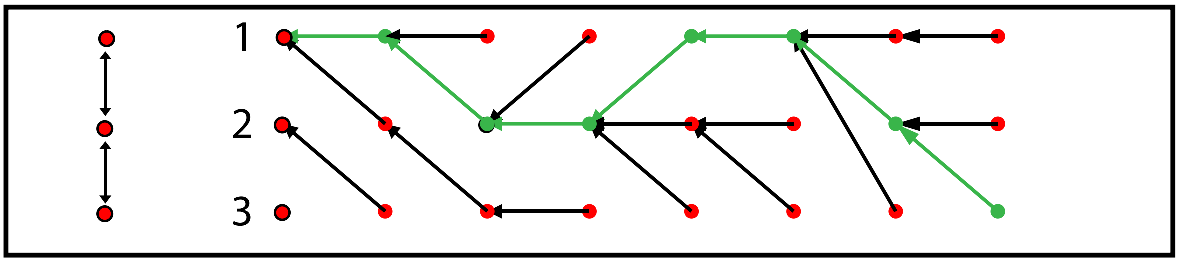


Fig. 6.9: Illustration of the lattice that is spanned to compute the Viterbi path and how backtracking is performed.

In Figure 6.9 we show the Viterbi path computation for a three state HMM where for each state at each time step t the optimal predecessor is stored. This can be used to trace back the optimal state sequence.

6.2.3 Learning HMMs

Our last problem is related to the learning of the model parameters $\lambda^* = (A, B, \pi)$ for a given observation sequence $O = o_1, o_1, \ldots, o_T$ such that $\Pr(O|\lambda^*) > \Pr(O|\lambda)$ for $\forall\lambda$, i.e., to adapt model parameters maximally to training samples. As there is no analytic solution known to determine optimal model parameters, the Baum-Welch reestimation algorithm is usually applied. This algorithm is an iterative *maximum likelihood estimator* that locally maximizes $\Pr(O|\lambda)$ and is proven to converge, however, not necessarily to the global optimum.

Starting from an arbitrary model λ, $\Pr(O|\lambda)$ is computed using the forward-backward algorithm. Meanwhile, the frequencies of state transitions and emitted symbols are counted. In the next step, the model parameters are adapted in such a way that more frequent state transitions and emitted symbols become more likely than less frequent ones. Clearly, the adapted model will now yield a higher probability for the given observation sequence. Note that these two steps correspond to the expectation and maximization steps of the EM algorithm; in fact, the Baum-Welch algorithm can be considered as a 'realization' of the EM algorithm.

To attain our goal, we define the variable

$$\xi(i, j) = \Pr(x_t = s_i, x_{t+1}|O, \lambda), \tag{6.27}$$

which denotes the probability that at time step t the transition from state s_i to state s_j is used under emission of the observation symbol o_t. Utilizing the α and β variables, this can be computed as

$$\xi(i,j) = \frac{\alpha_t(i) a_{ij} b_i(o_{t+1}) \beta_{t+1}(j)}{\sum_i \sum_j \alpha_i(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}. \tag{6.28}$$

Further, we define $\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j)$ as the probability to be in state s_i at time step t

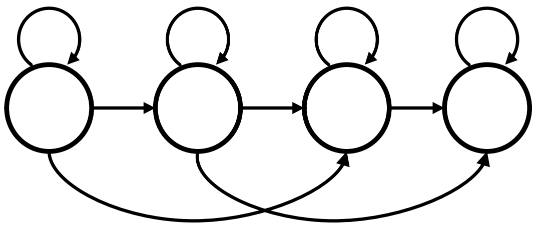
In the *expectation step*, the expected values for the number of transitions is computed by summing over the time index t, i.e.,

- $\sum\limits_{t=1}^{T-1} \gamma_t(i)$ the expected number of transitions from state s_i, and
- $\sum\limits_{t=1}^{T-1} \xi_t(i,j)$ the expected number of transitions from state s_i to state s_j.

Using these estimates, *the maximization step*, that is, a method to reestimate the model parameters, can be given as:

- $\overline{\pi}_i = \gamma_i(1)$
  The new initial probabilities correspond to the expected frequency of stops in state s_i at time $t = 1$.
- $\overline{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$
  The state transition probabilities equal the expected number of transitions from state s_i to state s_j, divided by the expected number of transitions from state s_i.
- $\overline{b}_j(o_k) = \frac{\sum_{t=1 o=o_k}^{T-1} \gamma_t(i)}{\sum_{t=1}^{T-1} \gamma_t(i)}$
  The emission probability is then the expected number of times in state s_j under emission of symbol o_k, divided by the overall expected number of times in state s_j.

With this, the reestimation step is complete and we obtain a new model $\overline{\lambda} = (\overline{A}, \overline{B}, \overline{\pi}$ with the property $\Pr(O|\overline{\lambda}) \geq \Pr(O|\lambda)$. This process is repeated until convergence is reached. We can use either of the algorithms presented above to test unseen observation sequences against this new model.


(a)

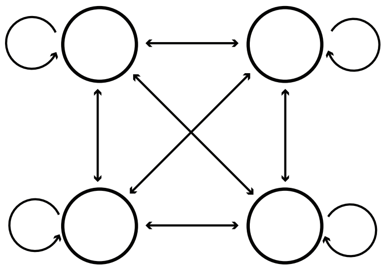

(b)

Fig. 6.10: Two typical HMM architectures. (a) A classical left-right HMM which is often used to model time sequential data such as speech and gestures. In this configuration with skip states, it is also called 0-1-2 model, in reference to the possible transitions in each state. (b) Ergodic HMM, where each state is reachable from every other state with one single step.

6.2.4 Practical HMM Issues

When dealing with HMMs, there are few aspects that have to be considered. The first is that of the general topology of the HMM. The structure of the HMM is governed by the transition probabilities a_{ij}. When setting some of them to zero, a particular ordering can be enforced. Note that, according to the formulas from the last section, zero probabilities will not get updated. Thus the initial structure of the HMM does not change during training.

In this work, we only consider *left-right* (see Figure 6.10(a)) and *ergodic* (Figure 6.10(b)) models. The main property of the latter is that every state is reachable from any other with one step, i.e., all $a_{ij} > 0$. This type of model is usually applied if there is no knowledge about possible structure of the process to model. Left-right models have the property that the state index necessarily increases over time or stays the same. Due to this, they advance the hidden states from left to right. More formally, this is expressed as $a_{ij} = 0, i < j$. Additionally, a skip limitation can be introduced, with $a_{ij} = 0, j > i + \Delta$, where Δ denotes the maximal number of states that can be skipped. Its sequential character has led to the assumption that this kind of model is especially suited to capture time sequential signals such as speech, or gestures.

Another design aspect relates to the observation symbols. Up to now, only discrete, i.e., finitely many, output symbols have been considered. In many real world problems the data is of continuous nature and discretizing it usually leads to information loss. There are nevertheless reasons to use discrete HMMs as they are faster to process and require less training data.

Using a continuous HMM effectively means that instead of probability distributions we have continuous density multivariate output distributions which are usually modeled by a number of mixtures of Gaussians. Be aware that good initial estimates are essential for continuous HMMs. To obtain such estimates, we use the segmental k-means procedure, also known as Viterbi training, since it aligns partial observation sequences to states based on Equations 6.23. Note that it could be sufficient to apply Viterbi training alone. However, in rare cases it can lead to suboptimal models. This is why Baum-Welch reestimation is usually applied subsequently.

To summarize, when it comes to modeling statistical processes with HMMs, the topology has to be determined, how many states to use, the observation representation (discrete or continuous), and in the latter case the number of mixtures of Gaussian. The answers to these questions depend on the case at hand and are often experimentally determined, as in our case.

6.3 Modeling and Training Gestures

As mentioned, the left-right structured HMMs are the means of choice for time-sequential processes such as gestures. We therefore based our further evaluations on this fundamental type of HMM. From our experiments (see Chapter 7), we found that models with a state number depending on the average observation length (three to six), no skip states, and one or two mixtures of Gaussians, perform best. The low number of states may be surprising. But to break down the complexity of such lengthy gestures as *waving* or *pointing*, we decided to subdivide a gesture into three parts.

This decision was partly inspired by [Nickel, Seemann, & Stiefelhagen 2004] where a pointing gesture is modeled using three HMMs according to the *begin* phase, *hold* phase and *end* phase. In our terminology, this corresponds to the preparation, hold/stroke, and retraction phase. This subdivision is furthermore reasonable as it allows to model the hold phase explicitly. We observe that it is the most volatile, for example during *waving*, one may move one's hand forth and back one time, two times or several times. This could be covered by allowing the corresponding HMM to jump back to the start state again in case the observations should support such a transition. Moreover, knowing to be in the hold phase of a gesture can be used to estimate an associated parameter, if applicable.

Note that for the less complex and shorter gestures, this subdivision is neither necessary nor reasonable. Therefore *dunno*, *nod* and *shake* have been left monolithic. We end up in modeling a total of 12 gestures: *prep_wave, hold_wave, retr_wave, prep_point, hold_point, retr_point, prep_thisbig, hold_thisbig, retr_thisbig, dunno, nod, shake.*

6.3.1 Collecting Training Data

For there does not exist useful standard data, we had to collect the training data ourselves. For this, we asked five different people to perform the gestures in front of a standard webcam. We chose two different locations, different lighting conditions - sunlight, daylight, artificial light - and different backgrounds, most of which were cluttered. Figure 6.1 from the beginning of this chapter was taken from videos recorded under these conditions.

From each person, around 15 repetitions of the same gesture were recorded at a rate of 20fps, such that we end up in a database of 75 samples per gesture. From these, around 50 were taken for training the HM models. For this, the videos were manually labeled. That is, we marked the start and endpoint of each gesture as well as the intermediate points denoting the end of the preparation phase and hold phase, respectively. The videos were processed to obtain the 2D trajectory data plus head rotation angles. This data was split into pieces corresponding to the label file information and feature transformed and thus constitutes our trainings and test set. In case of the bimanual gestures and the *shake* gesture, we could double the acquired data by swapping left and right hands, and flipping the sign of the yaw angle.

6.3.2 Rejection Of Non-Gestures

Given N trained reference models $\{\lambda\}$ for gestures to recognize and an observation sequence O there will always be a winning class λ_i according to

$$\lambda_i = \arg \max_{i=1,\ldots,N} [\Pr(O|\lambda_i)] \tag{6.29}$$

The task of rejection is to validate if O is indeed generated from λ_i. One straightforward approach would be to define a threshold likelihood which has to be exceeded by λ_i. However, this has several drawbacks. First, deriving such a threshold from the training data is not very reliable. Consider the case that one performs a gesture rather slowly. In this case more state transitions (probably self transitions) are required to find a path that fits the observations. And although this could be a perfect recognition result with all the other likelihoods being far below this one, it could fall below such a global threshold and get rejected thusly. The other drawback is that such a decision can be undertaken only after the end of the gesture has been detected, which prohibits its use in an online approach. More promising is to compare the likelihoods of the $\lambda_j, j \neq i$ and demand that $\frac{\Pr(O|\lambda_j)}{\Pr(O|\lambda_i)} < \theta \; \forall j \neq i$, which does not suffer from first described problem, but again requires a fixed threshold. An approach where this problem is circumvented is described in [Xiao-Hui & Chin-Seng 2006].

In this work, we consider an additional alternative model $\tilde{\lambda}$ against which λ_i is tested and accepted if it scores higher. Defining such a model, however, is difficult as it has to cover the entire space of possible alternatives for good performance. An obvious approach is to build a model using a large number of non-meaningful patterns. Such models are called *filler* models or *garbage* models [Xiao-Hui & Chin-Seng 2006]. Clearly, it is difficult to obtain a representable amount of data which covers arbitrary motions.

Nevertheless, we pursued this approach by extracting all motion data not being marked as gestures as training patterns for our garbage model $\tilde{\lambda}$. Additionally, the gesture segments not belonging to the same gesture category were re-used, such that we were able to collect around 2000 samples for training $\tilde{\lambda}$.

As in general nothing can be said about the order of arbitrary motions, the left-right HMMs are not the adequate choice for $\tilde{\lambda}$. Instead an ergodic design is chosen with five states, and again one or two mixtures of Gaussians.

Another approach was presented in [Lee & Kim 1999] and again, slightly adapted, in [Yang, Park, & Lee 2006], where a *threshold* model is defined which is based on the meaningful models. The general idea is to copy all states from all trained models and arrange them in a fully connected, i.e. ergodic, HMM with smoothed output probabilities. More exactly, let S_G be the set of states j of all gesture models. A dummy start state s is connected to all these states with transition probabilities $a_{sj} = \frac{1}{|S_G|}\ \forall j \in S_G$, which in turn are connected to a dummy end state e with transition probabilities $a_{je} = 1 - a_{jj}\ \forall j \in S_G$, with a_{jj} being the self transition of state j. The two dummy states observe no symbol and are passed without time delay. Therefore every state can be reached from every other in a single transition. The output probabilities $b_j(k)$, k being the emitted symbol, are flattened via floor smoothing. For this reason, the meaningful models are expected to have a higher likelihood for their target pattern, whereas for non-gesture patterns the threshold model should perform better than any trained model due to the broader variances.

We adapted this approach for our work and found it to outperform the garbage model, see Chapter 7.

6.3.3 HMM Networks

As explained, some gestures were subdivided according to the three phases preparation, hold and retraction. They can, however, not be treated independently. Rather, the phases appear in a specific order which has to be considered during recognition. We therefore define regular grammars for the complex arm gestures. The grammar for

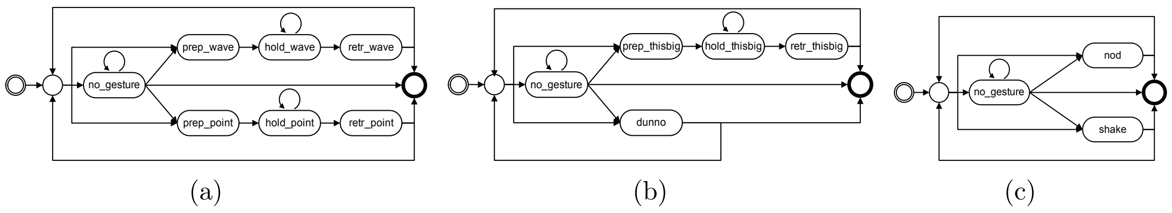


Fig. 6.11: The HMM network structures. The first two for mono- and bi-manual gestures, and the right for head gestures. The start and end states of the HMMs are drawn bold and are non-emitting. As a branching point, a ”dummy state” which is also non-emitting is introduced for each HMM.

waving, for instance, can be given as follows

$$wave = prep\_wave \rightarrow hold\_wave^{+} \rightarrow retr\_wave,$$

where $^{+}$ refers to the operator ”at least once”. This specifies the order of the phases and indicates that the hold phase, e.g., moving the hand left and right, can occur several times.

Generally, a certain grammar is desirable which governs the allowable recognition sequences. As we consider monomanual, bimanual, and head gestures independently, we define three such grammars. These are shown in Figure 6.11(a), 6.11(b), and 6.11(c) respectively.

In practice, such a grammar is realized by building a single large HMM with constant transitions between the models, optionally with a transition back to the start of the model as in case of the *hold_ ** models. Further, a dummy state is introduced which serves as branching point and merging point respectively and is non-emitting.

Finally, it can be seen, why Viterbi decoding is applied in such a case. Mainting the summed probabilities in a large HMM is far more time consuming than computing the most likely path. This is especially true if we represent probabilities logarithmically which is usually the case as to prevent underflow. We note that the Viterbi path is only an approximation of the forward probability. In practice, however, both probabilities are very close ([Rabiner 1990].

6.4 Recognition

Once a person is modeled in the belief, the gesture recognition system is activated for that person. Up to four HMM networks are constructed, one for the head gestures, one for the right hand, if observable, one for the left hand, if observable, and one for the bimanual gestures if both hands are observable. The observations are fed into the network and the best path is computed using the Viterbi algorithm. Every path through the network passing through t emitting states, where t is the number of observations so far, is a potential recognition hypothesis. However, most of the paths will have very low probabilities and thus, only the n best will be kept at each time step; all other are discarded, for efficiency reasons. The path with the highest likelihood will effectively form our single hypothesis. Clearly, in the beginning of a gesture, the highest scoring path can change more or less frequently, but the deeper we advance into the network the more reliable this hypothesis will become.

For reliable online recognition of the crucial hold phase of a gesture, we therefore demand a minimum number of times the path is indeed in the respective model. Only then, we start estimating possible parameters. In cases where no distinct path is found, we make use of the Viterbi backtracking to determine the actual path taken.

One problem with those individual HMM networks with different-dimensional inputs is, that they cannot be compared directly and thus cannot be used to discern monomanual from bimanual gestures. Currently, this concerns mainly the *thisbig* gesture, where each of the hand motions can resemble a *pointing* gesture, depending on the distance between hands. Since the bimanual network aims at the symmetry of gestures, we can however safely assume that indeed a bimanual gesture was performed and reject possible monomanual hypotheses. This can additionally be backed up by comparing both monomanual recognition networks which should report the same hypothesis with similar likelihoods.

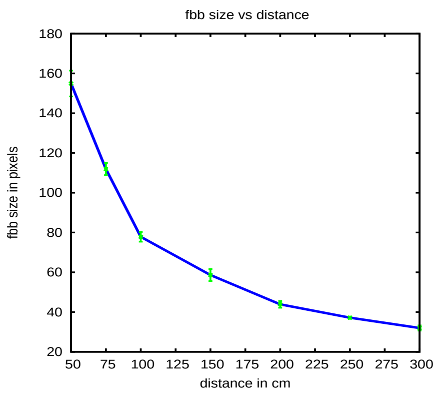
As this settles the recognition of gestures as such, we consider in the following the parameter-dependent *pointing* and *thisbig* gestures. During the hold phase of these gesture we additionally estimate the parameter they depend on.

6.4.1 Parameter Dependent Gestures

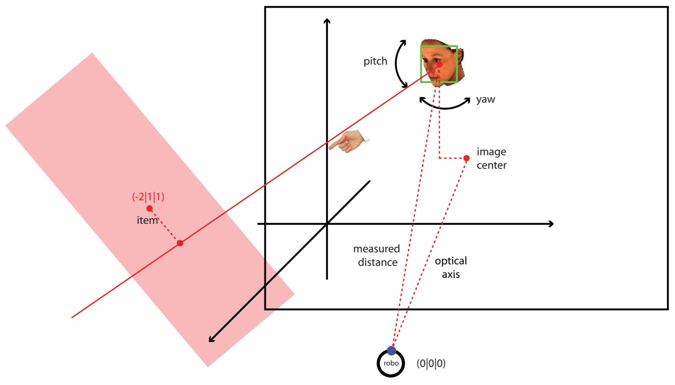
The task is to recover the parameter of gestures dependent thereof. Estimating these parameters is only possible, if we have a mapping from the 3D world to the 2D projection we observe.

For this reason, we calibrated the camera by measuring the fixed size of some objects in pixels given a distance of $200cm$. Surprising enough, the relation turned out to be free of distortions, that is, $1pixel \approx 0.375$ throughout the image. In order to obtain an estimation of the distance of the person to the robot, (r), the width of the face bounding box of different people standing at different positions in front of the camera was recorded. Having expected a linear relation, we found that in the close-up-range this is not the case. However, we treat it piece-wise linear, as plotted in Figure 6.12(a). Of course this leads only to a rough estimation of r, but it is sufficient for our purposes.

Now, estimating the distance between the hands during the hold phase of *thisbig* is



(a)



(b)

Fig. 6.12: Diagram of the dependence of the face bounding box size from the distance to the robot/camera.

simple. With the horizontal halfway distance in pixels $d_x = (x^l + x^r)/2$, we compute the angle φ as

$$\varphi = \text{atan2}(d_x, 200).$$

Then, the distance r_f is estimated via linear interpolation and the distance in centimeters d_{cm} is calculated as

$$d_{cm} = 2 \cdot r_f \sin \varphi. \tag{6.30}$$

This distance is repeatedly computed during the hold phase. Moreover, once *retr_ thisbig* follows up, the mean computed during the entire hold phase is signaled for a final estimation.

For the *pointing* gesture, matters are slightly more involved and are therefore described in an own section in the following.

Pointing Target Estimation

In the following we assume that we know the positions of objects that can be the target of pointing gestures. We further assume that the person who is pointing is also looking at the pointing target. Finally, we assume this gaze direction coincides with our head pose estimation.

The first step is to derive the 3D position of the person who is currently pointing. Using again the distance estimation, r is obtained as before and the offsets d_x and d_y between the detected face $\mathbf{c}_f$ and the image center $\mathbf{c}_i$, which coincides with the intersection of the optical axis with the image plane, are determined. The angles Θ

between r_f and the optical axis along the y axis as well as φ along the x axis are computed as:

$$\begin{aligned}\Theta &= \text{atan2}(d_y, 200)\\ \varphi &= \text{atan2}(d_x, 200)\end{aligned}$$

Θ, φ, and r define a straight line through the focal point and $\mathbf{c}$, in a spherical coordinate system. Using the transformation equations,

$$\begin{aligned}x &= r\sin\Theta\cos\varphi\\ y &= r\sin\Theta\sin\varphi\\ z &= r\cos\Theta,\end{aligned}$$

the Cartesian coordinates of $\mathbf{c}$ are attained.

Using the head pose estimation, the line of sight l is computed with $\mathbf{c}$ as initial point and a unit vector $\mathbf{u}$ pointing toward the camera. $\mathbf{u}$ is rotated accordingly to the pitch angle Θ_y and yaw angle Θ_z. The roll angle Θ_x does not convey information for pointing target estimation. Finally, for each entry in the list of 3D positions of known items $\mathbf{i}$ the distance between the item in question and l is calculated analytically as

$$d = \|\mathbf{i} - (\mathbf{c} - (\mathbf{u}\cdot\mathbf{c} - \mathbf{u}\cdot\mathbf{i}/\mathbf{u}\cdot\mathbf{u})\mathbf{u})\|\,. \tag{6.31}$$

The object with the minimal distance is identified as the pointing target. Figure 6.12(b) summarizes the steps performed to estimate the pointing target.

This concludes the description of our efforts undertaken so far to recognize gestures. We performed a variety of experiments to investigate the usability of our approach and present them in the following chapter.

7 Experiments

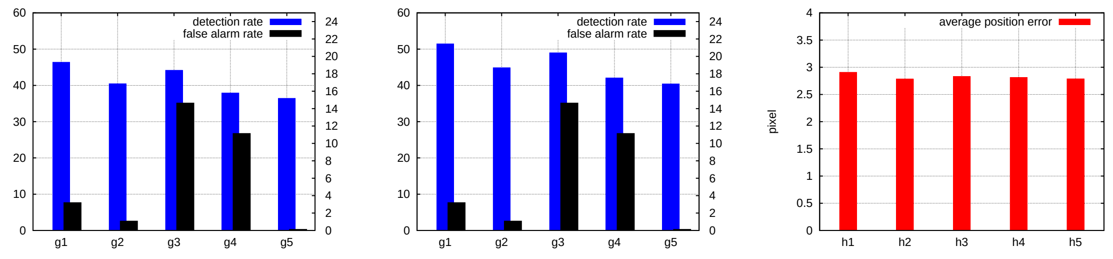
Our approach to gesture recognition has been implemented and tested comprehensively to evaluate its applicability. In this chapter, we present various experiments which cover the individual parts of our system, namely hand detection, tracking and gesture recognition.

7.1 Hand Detection Experiments

As described in Chapter 3, we trained several classifiers with either different positive or negative sample sets. As training takes arbitrarily long (at least 4-7 days on a decent computer), we had to be somewhat restrictive. Therefore, we first build intermediate classifiers with only a few hundred positive samples of a single person. In doing so, we could experiment with the parameters to choose for training. These include the number of stages to train, the set of features to use (basic rectangular or rotated or both), minimum detection rate and maximum false alarm rate per stage etc.

The final classifiers were trained using the full feature set, 20 stages and a minimum detection rate of 99.5% and a false alarm rate of 50% per stage. We selected around 5000 positive samples. In detail, we trained five generic hand classifiers and six specific (=right) hand classifiers, as follows:

$\mathbf{\Gamma}_1$ denotes the one described in Section 3.3 with the positive samples duplicated by the factor 16. From the resulting 5000 positive sample sets, around 470 were randomly selected to constitute the training set of ∼7500 positive samples. For the second classifier $\mathbf{\Gamma}_2$, we did not duplicate the positive samples but took the complete set. $\mathbf{\Gamma}_3$ and



(a) (b) (c)

Fig. 7.1: Test results for the generic hand classifiers. (a) The overall detection rate plus false alarm rate. (b) The true rates, when subtracting missing skin detections. (c) The average distance in pixels from the true hand position.

$\mathbf{\Gamma}_4$ relate to $\mathbf{\Gamma}_1$ and $\mathbf{\Gamma}_2$, but use positive samples containing 10% less than the respective face bounding boxes. Finally, we trained the comparison classifier $\mathbf{\Gamma}_5$, without the restrictions imposed on the other four, see Section 3.3.

Similarly, we trained right hand classifiers $\mathbf{\Sigma}_1$, $\mathbf{\Sigma}_2$, and $\mathbf{\Sigma}_4$ corresponding to the respective generic classifiers. Additionally, we trained a tuned classifier $\mathbf{\Sigma}_3$, where only left hands were used as negative samples. To be able to directly compare the outcome of this tuning, a tuned version of $\mathbf{\Sigma}_4$, $\mathbf{\Sigma}_5$, was trained too. A sixth classifier $\mathbf{\Sigma}_6$ relates to $\mathbf{\Gamma}_5$, yet again using only left hands as background.

For our test, we selected 500 images that have not been used for training. In doing so, we minded not to include too similar images (in terms of hand shape, hand position and background) to capture a reasonable broad range. The hands in the images were manually labeled for their position and their laterality. See Figure 7.2 for a selection of these images.

7.1.1 Generic Hand Classifier

From experience, we knew that we have to expect a high false alarm rate. Thus we set the minimum number of detections that account for a single hit as high as 15. We then processed all test images, counting the number of detected hands which were indeed hands (= detection rate) and the number of hand detections that in fact were not hands (=false alarm rate).

The results are given in Figure 7.1. In (a) we see the overall detection rates in blue and the overall false alarm rate in black. For the detection rates we see that the smaller versions perform worse and conclude that we might have trimmed too much context information. That $\mathbf{\Gamma}_1$ outperforms $\mathbf{\Gamma}_2$ may be explained that less different hand shapes were considered during training, thus the selected features are more general and accept

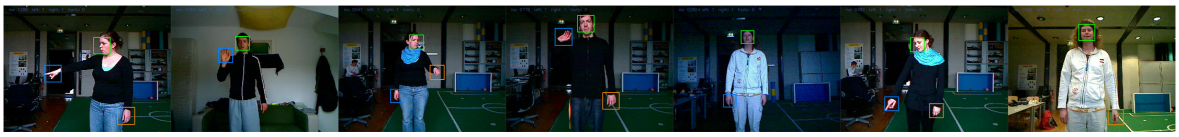


Fig. 7.2: Test images used for the hand classifier experiments. We chose images containing different people, different backgrounds, and different lighting conditions . Example detection results of the generic classifier $\mathbf{\Gamma_5}$ and the specific classifier $\mathbf{\Gamma_3}$ are drawn in. Orange denotes right hand detections, right with respect to the image. Blue denotes left hand detections whereas grey denotes unknown laterality.

more unseen samples.

The obtained detection rates so far are not what we expect, but can easily be regulated, i.e. increased, by lowering the minimum number of single detections. Yet this comes to the price of an increasing false alarm rate. When looking at these, there only remains one acceptable classifier, $\mathbf{\Gamma}_5$, which is a somewhat unexpected. Our explanation is that it might not be of advantage to include images containing hands to the negative sample set, as we did. Another reason might be that with the fixed size we chose to crop the positive samples, a lot of additional variation was introduced, as one and the same hand can look completely different, at least seen from a features perspective.

Another source for the low detection rates is that in some cases hand detection did not occurred in the first place as no skin blob could be found. This accounts for a decrease of $\sim 5\%$. The reassessed rates are therefore given in Figure 7.1(b). We argue, that in real world scenarios, where we track the hands and additionally update the histogram, the problem should not happen too often. This is also backed up by our experiments in Section 7.2.

In Figure 7.1(c) we plotted the average distance of the correct detections to the labeled positions. We see that there are no significant differences between the classifiers. An average error of less than 3 pixels is certainly within the acceptable range.

In the following, we only consider $\mathbf{\Gamma}_5$ as it by far outperforms the other in terms of the false alarm rate. Surely, this is to prefer over a high detection rate as we can bridge a missing detection easily, but considering too many hand candidates is decreases robustness. Moreover, by adjusting the minimum number of detections we can increase the detection rate. Figure 7.3 shows some results referring to this. We see that the false alarm rate increases only moderately and thus acceptable detection rates of up to $\sim 75\%$ can be achieved. Also the distance errors increase only slightly.

As we prefer low false alarm yet wish an acceptable detection rate, we finally settled for the minimum number of six detections, with a false alarm rate of only $\sim 0.63\%$ and a detection rate of $\sim 71\%$. This also forms the basis of the subsequent experiments.

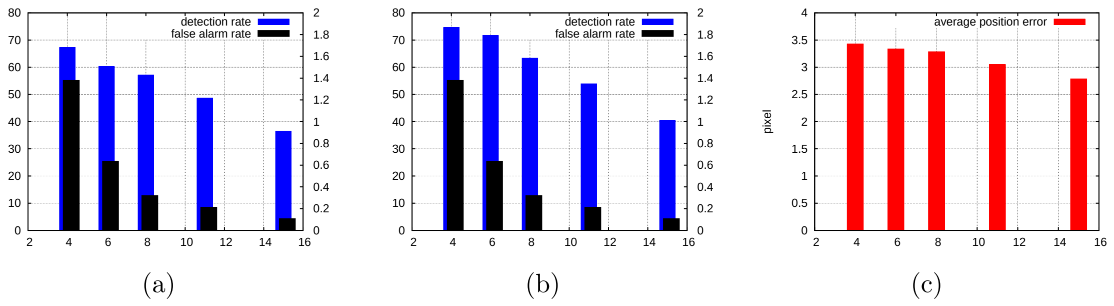


Fig. 7.3: Test results for our winning hand classifier $\mathbf{\Gamma}_5$ with different minimum number of detections.

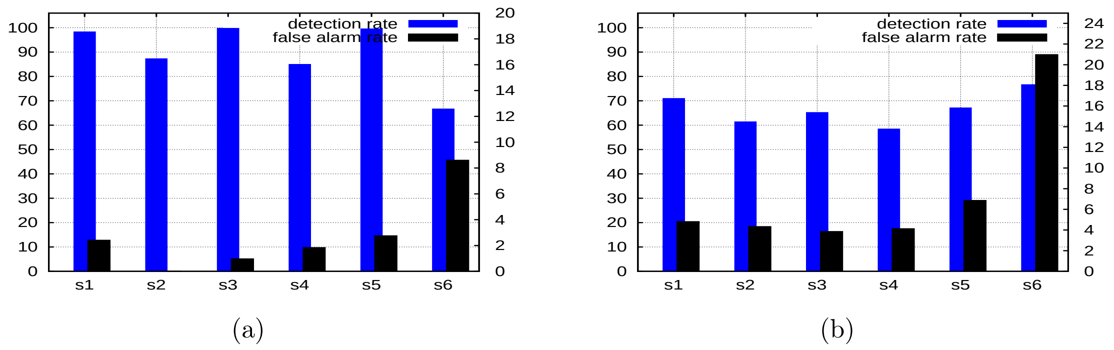


Fig. 7.4: Test results for the specific hand classifiers. In (a) the performance of the of the classifier on left hands is shown. Likewise, (b) shows the performance on right hands.

7.1.2 Right Hand Classifier

We used the same test set as before and applied the specific classifiers for each correct hand detection of the generic hand classifier. This time, we counted the number of detections of a right hand given it was indeed a right hand (= detection rate) and when it was a left hand (= false alarm). The same was repeated for the left case.

The results are given in Figure 7.4. (a) depicts the performance of the specific classifiers for left hands. This time, the size of the positive samples the classifiers were trained on have no influence on the performance. This comes expected as the features used to discriminate right from left hands will likelier focus on the structure of the hand itself than on the context, for example on the existence and position of a thumb. This also explains why $\mathbf{\Sigma}_6$ performs far worse as it was trained on positive samples which did on average contain more context. Thus the fine structures needed to discern right from left hands may not be available.

When looking at Figure 7.4(b), we are surprised to find one and the same classifiers

Table 7.1: Performance of our hand detectors.

	Detection rate	**False alarm rate**	**Avg. dist.**
Generic detector	81.25	0.10	2.89
Specific detector	89.50	5.56	-

to perform considerably worse than for left hands. This can only be explained under consideration of the test set of images. This set consists of images extracted from gesture training data videos where primarily single handed gestures are performed. These were mostly carried out with the left hand. We postulate therefore that a loosely dangling right hand is much harder to classify as it looks very similar to a loosely dangling left hand. This is especially the case if the thumb is not clearly visible, which is also likely to happen in this position.

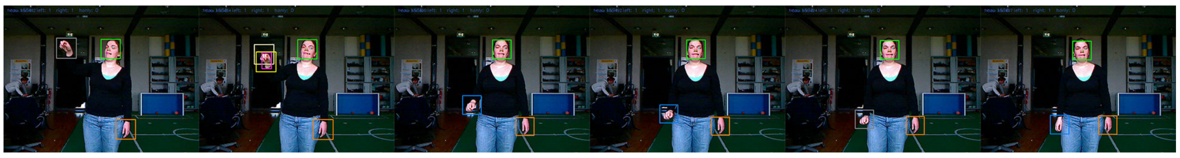
For an overall performance of the classifiers we therefore have to average both results. Nevertheless, the resulting detection rates remain remarkably high with at the same time sufficiently low false alarm rates. Also the ability to distinguish between left and right hands with a detection rate of almost 90% and a false alarm rate of 5% is highly satisfactory.

We note that the results depend on the distance of the people to the camera. When the people leave the normal interaction radius of approximately 2m, the performance of the hand detection system gets worse, at least when using a standard webcam with a resolution of 640×480 pixel.

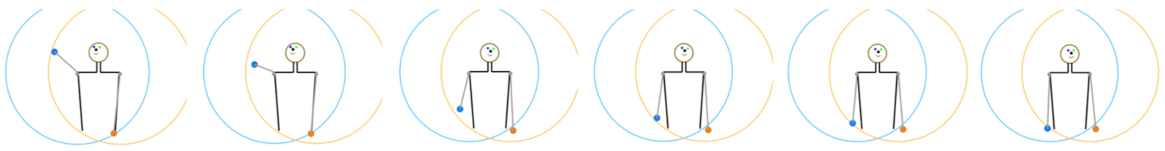
7.2 Tracking Experiments

In the following experiment we investigated our assumption that the hand detection rate should increase while tracking since we expect to find all skin regions. Additionally, we wanted to know how accurate tracking is performed when no hand is detected. In the absence of ground truth data (e.g., from data gloves), we yet again manually labeled a test image sequence, this time of 500 successive frames for their hand positions (and their laterality). A part of this sequence is shown in Figure 7.5(a).

The results for the best classifiers $\mathbf{\Gamma}_5$ and $\mathbf{\Sigma}_4$ are summarized in Table 7.1. We see that the average error distance has even improved as compared to the detectors performance alone. This hints that our skin regions extraction works satisfying. As expected, the detection rate increased to over 80%, which is more than sufficient to



(a)



(b)

Fig. 7.5: Test image sequence for the tracking experiment. The colors of the drawn in detections have the same meaning as in Figure 7.2. Additionally, in case of non-detections the previous position is drawn in white together with the Kalman filter prediction in yellow. Magenta finally, denotes the CamShift convergence. This can be observed in the second frame of (a). (b) shows the our compact person model maintained by means of the detection and tracking steps. Additionally, the head pose estimation is indicated.

robustly track the hands using Kalman filters and our probabilistic belief. Also the ability to distinguish between left and right hands with a detection rate of almost 90% and a false alarm rate of 5% is highly satisfactory. In Figure 7.5(b) we show, that our compact person model is reliably maintained throughout the sequence with the correct hand assignment.

In a second test we examined how quickly a person model together with two hands was established in average, by processing 50 previously unseen video snippets until the model was fully established. In Figure 7.6 the magenta curve denotes the results. As we accept faces and hands only when $bel(f|z_{1:t}) > 0.95$ and $bel(h|z_{1:t}) > 0.95$, respectively, this signifies that at best after 7 frames the model can be complete. This is because we start hand detection only after the face got accepted which happens at best at frame number 3. While the face and both hands have been detected not later than after 16 frames, establishing the model can take considerably more time. These cases, where the hand does not get re-detected often enough, coincide with the person standing farther away from the camera than 2-3m.

With a processing speed of 20fps, a person model is complete after 1.5s in 90% of all cases which is sufficiently fast. Note that recognition of head and monomanual gestures can start even earlier.

We also tested our approach how well it handles multiple person tracking. For this, we recorded videos with different numbers of people. Figure 7.7 shows an example

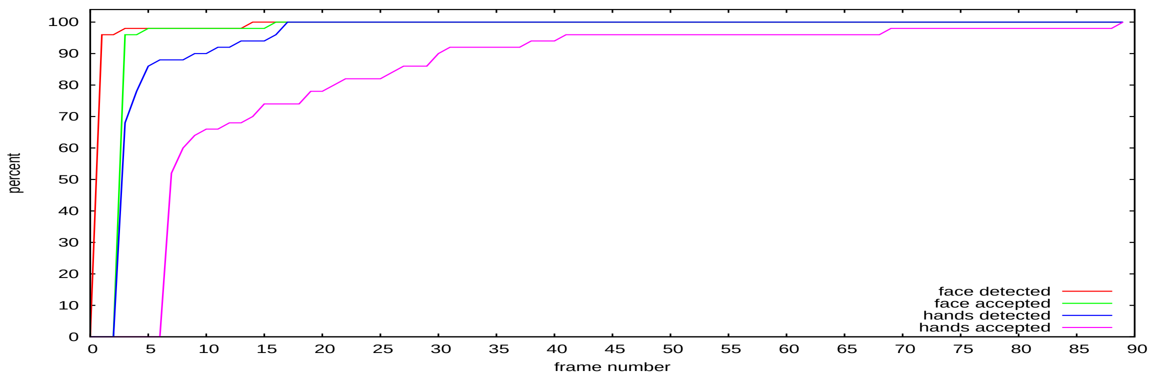


Fig. 7.6: Test results for model establishing. The red curve denotes the percentage of faces detected dependent on the frame number. The green curve denotes the percentage of faces accepted by means of the belief. As can be seen, once found, a face is re-detected constantly. The blue curve denotes the percentage of hands detected. We see that once we start detecting nearly 70% of the hands are detected immediately, which reflects the detection performance of our classifier. The harder cases in farther distance are not re-detected constantly and therefore take some time until they are accepted as indicated by the magenta curve.

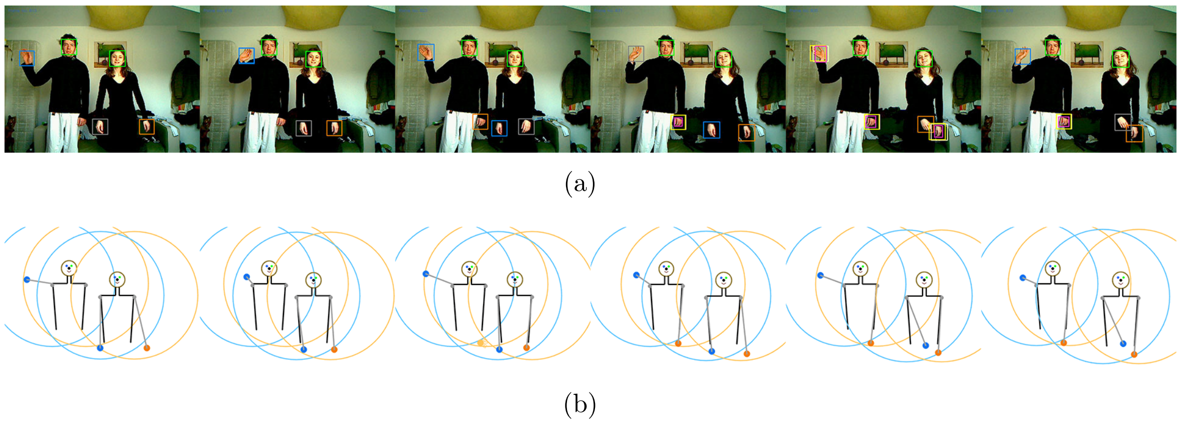


Fig. 7.7: Sample sequence for multiple person tracking. In (a) depicted are the detection/tracking results and in (b) the respective person models.

Table 7.2: Final HMM configuration.

	States	**Skips**	**Mixtures**
Monomanual	3-6	0	1
Bimanual	4-8	0	2
Head gestures	6-7	0	1

sequence with two people. In 7.7(b) the corresponding person models are depicted. As can be seen, the assignment of hands is correct and does not get confused when hands come close together as at the end of the sequence. We found that we can reliably track up to three person. As the performance of our system depends on the number of faces and hands to be tracked, further optimizations would be necessary to track even more people.

7.3 Gesture Recognition Experiments

Our first experiments with regard to gesture recognition were conducted to find out the the best configuration parameters for the HMMs. We varied the number of states, the number of skip states, and the number of mixtures of Gaussians. The initial configuration were three states, zero skip states, and one mixture of Gaussians. Setting skip states higher than zero did in no case improve recognition results. On the contrary, some transition probabilities could not get learned properly due to data insufficiency. Similarly, using more than two mixtures of Gaussians resulted in data insufficiency. Increasing the state number gave no consistent picture: some models performed slightly better, some slightly worse. When inspecting the learned transition probabilities of the models performing worse, we could see that some forward transitions were set to one, i.e., that this state was unnecessary.

We then compared the average observation length of all gestures, and discovered that it was those models for gesture phases or gestures that took longer than the others that most profited from the higher state number. This intuitively makes sense and we therefore decided for a variable state number depending on the average observation sequence length of each gesture (phase).

For the bimanual gestures, for which we had doubled the available training data, we noticed a slight improvement when using two mixtures of Gaussians over using a single Gaussian. Table 7.2 summarizes the final HMM configuration for each of the three

Table 7.3: Recognition of monomanual gesture phases.

	p_wave	*h_wave*	*r_wave*	*p_point*	*h_point*	*r_point*	*rec. rate*
p_wave	25	0	0	0	0	0	100%
h_wave	0	25	0	0	0	0	100%
r_wave	0	0	25	0	0	0	100%
p_point	0	0	0	25	0	0	100%
h_point	0	0	0	0	25	0	100%
r_point	0	0	1	0	0	24	96%

recognition networks.

After training all individual HMMs, we tested their ability in distinguishing the corresponding gesture or gesture phases (preparation (p), hold (h), and retraction (r) phase). For this isolated gesture recognition task, we used the Viterbi path and counted the number of correctly recognized gesture phases from the number of all test sequences.

Table 7.3 shows the percentage of correctly recognized segments for monomanual gestures. As can be seen, using the extracted features, the individual phases of monomanual gestures can correctly be recognized. Only one error occurs for a segment containing a retraction point phase which is classified as retraction wave. This can be explained by the fact that both retraction phases contain similar movements in the end. When considering a whole observation sequence consisting of all three phases, this error does not occur since the preparation and hold phase are correctly recognized.

Table 7.4: Recognition of bimanual gesture (phases).

	dunno	*p_thisbig*	*h_thisbig*	*r_thisbig*	*rec. rate*
dunno	50	0	0	0	100%
p_thisbig	1	49	0	0	98%
h_thisbig	1	0	49	0	98%
r_thisbig	1	0	0	49	98%

For the recognition of bimanual gestures shown in Table 7.4, it can be seen that in a single test sequence, the phases of *thisbig* are classified as *dunno*. Finally, the results for the head gestures are given in Table 7.5.

When sequences in which persons are not performing any gesture are included into the test set, we achieve a overall recognition rate of 90% for monomanual as well as for bimanual gestures. The largest part of this error results from the fact that it

Table 7.5: Recognition of head gestures.

	nod	*shake*	*rec. rate*
nod	24	1	96%
shake	0	25	100%

sometimes happens that no gesture phases are classified as the preparation phase of a gesture. This is not critical since no gesture gets the highest likelihood as soon as the composed HMM expects the hold phase to start. For the head gestures we achieve only 80% as especially the nod gesture often involves only small variations in the angle trajectory and is therefore often classified as non-gesture.

We also performed the same experiments with the raw position data, that is, without feature extraction but relative to the face coordinates and normalized. We were surprised to find that recognition results only got slightly worse, around 3-5%. We suggest, however, that training and test data may not contain the whole variation that is to expect from real life data. Moreover, as we only consider a small set of gestures, these can be distinguished with relative ease. We suspect therefore that our feature transformation will have more effect on a full application scenario.

The next experiment is designed to evaluate the performance of the HMMs on sequences containing whole gestures. We used the composed HMMs, which also contains the no gesture model, to infer the gesture and counted how often the most likely hypothesis corresponded to the true gesture. We evaluated the Viterbi path on each frame. Figure 7.8 shows the results for all six gestures. As can be seen, the gestures can be reliably recognized after processing only few frames. Nodding seems to be most difficult to recognize because sometimes the people barely move their head. And, again, we made the observation that *thisbig* sometimes tends to be classified as *dunno*.

For our last experiment, we wanted to compare the actual likelihoods each of the models produces given an unknown observation sequence. For this, we selected five test samples. They are all shown in Figure 7.10. The *pointing* gesture in shown in the two top rows, the *waving* gesture in the second two rows. In the middle rows we see the *thisbig* gesture. This is followed by an sample of non-gesture. The two bottom rows show the dunno gesture. Further, we draw in the detection and tracking results. The head gestures are not included in this experiment, as we wanted to compare the likelihoods.

We then split the compound HMMs into three pieces, one for each gesture. This means, that one model consists of the three phase models in case of *thisbig*, *pointing* and *waving*, and of the one gesture model otherwise. As we wanted to compare the

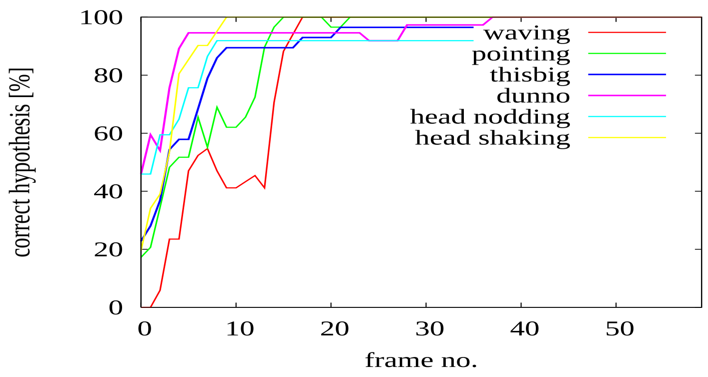
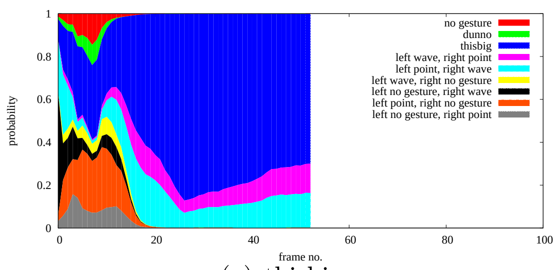


Fig. 7.8: Number of frames after which the most likely hypothesis is the correct gesture.

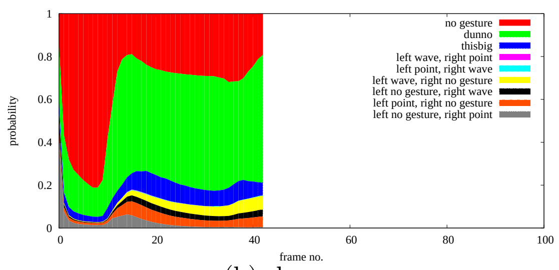
likelihoods for each of these gesture model, we split the data for bimanual gestures into two data sets, one for the left hand and one for the right hand. We then computed both the likelihood the monomanual gesture models produced given these datasets. The likelihoods for all monomanual gesture models given a specific observation sequence (e.g. thisbig) where then summed together with bimanual gesture model probabilities. This sum was then normalized to one. For the result see Figure 7.9.

Note that we used the forward algorithm to compute the likelihoods. We observe that the likelihoods develop similar to the maximum likely path probability used for the experiment before. As we use Viterbi path decoding in practice for efficiency reasons, the procedure described above to determine the most likely gesture for both monomanual and bimanual gestures cannot be applied. We therefore rely on the heuristics to prefer the bimanual gestures over possible monomanual gestures, which is also backed up by these results.

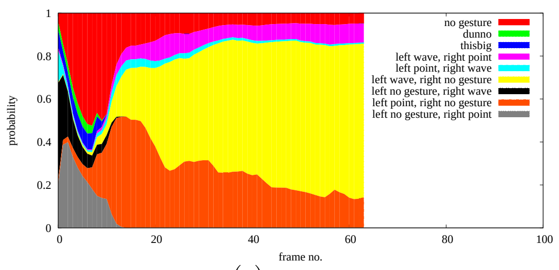
Finally, we repeated the above experiment to investigate whether our system is able to recognize gestures when only part of it could be observed. Clearly, this is only of interest, if the part which is observable includes the hold phase. This is, however, an important case since a gesture can start without an explicit preparation phase. To simulate such a situation, we computed the probabilities for each model as above with the observation sequence beginning with the hold phase. Figure 7.11 shows the results. In fact, all of the correct gestures could finally be recognized which is a strong hint toward the robustness of our approach.



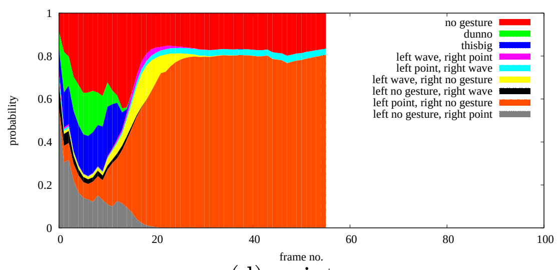
(a) thisbig



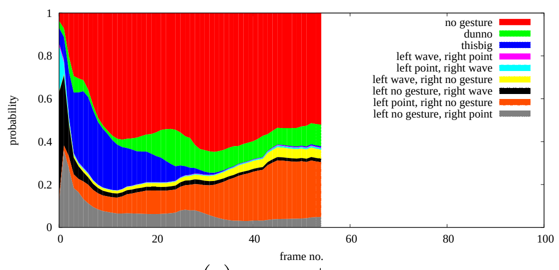
(b) dunno



(c) wave



(d) point



(e) no-gesture

Fig. 7.9: Likelihood development for each model given an unknown observation sequence. Each of these sequences starts with non-gesture motion of around six frames followed by the actual gesture.

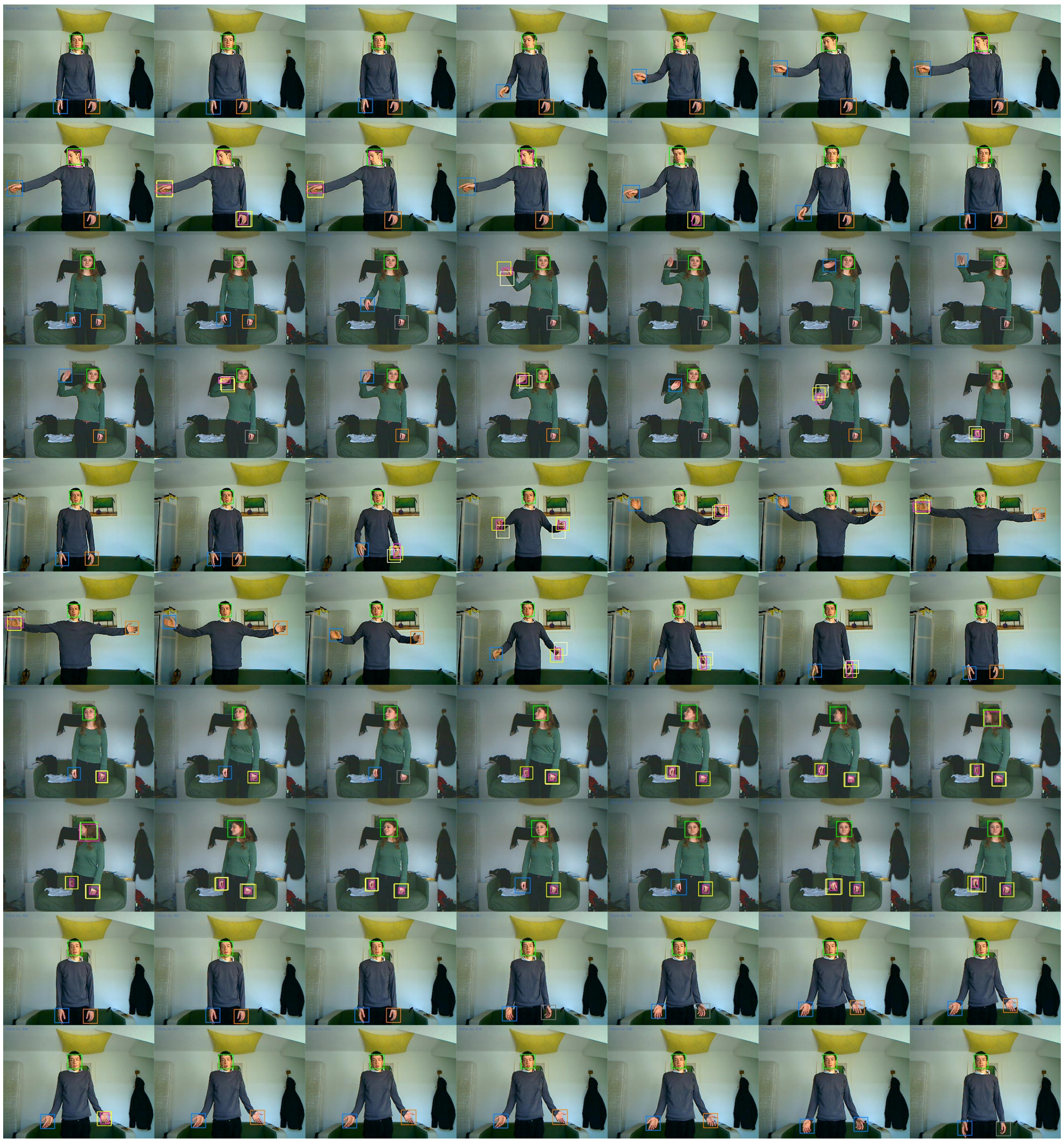
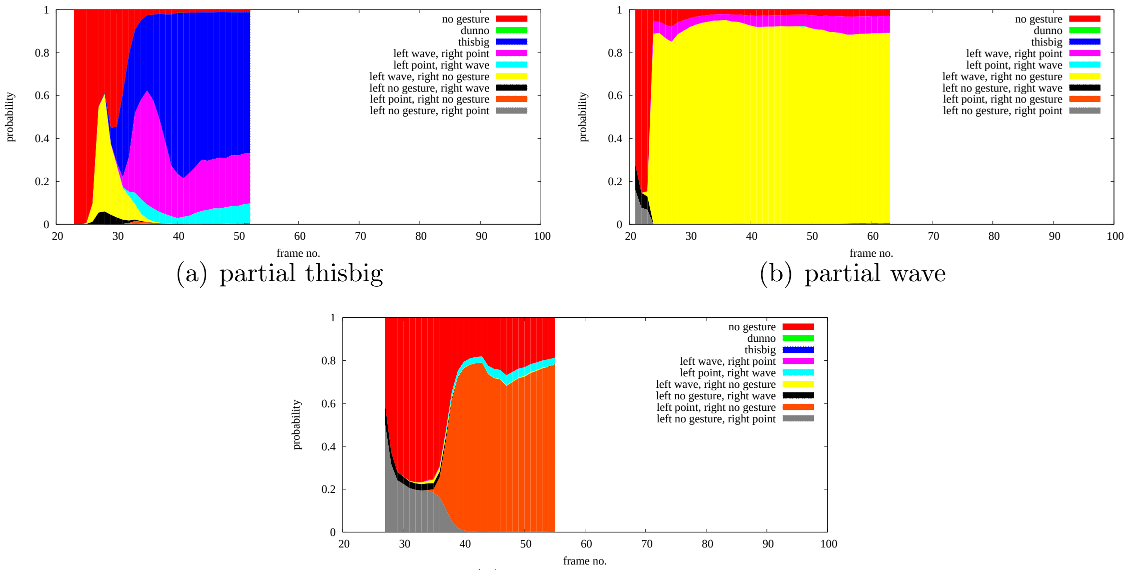


Fig. 7.10: Gestures used for our final experiment with drawn in detection results. The first two rows show the pointing gesture. The second two rows the waving gesture. The middle two rows depict the thisbig gesture. The second last two rows show arbitrary movements with the arm, i.e., a non-gesture. The last two rows illustrate the dunno gesture.



(a) partial thisbig (b) partial wave

(c) partial point

Fig. 7.11: Likelihood development for each model given an unknown partial observation sequence. Each of these sequences starts with the hold phase.

7.4 Parameter Estimation Experiments

Currently, we consider two gestures conveying information which are *thisbig* and *pointing*. For the *thisbig* gesture, we asked people to indicate the size of objects. In the absence of any real object, we told them to indicate the sizes 25cm, 50cm, 100cm, and 150cm. We recorded these gestures in distances ranging from 1.5m to 2.5m, as denoted by the green circles in Figure 7.12. This way, we collected a total of 32 gestures.

We processed all these videos and estimated the parameter once the hold phase of *thisbig* was recognized. All of the hold phases could reliably be recognized. The estimation of the indicated size corresponded to the intended one in 94% of the experiments. No significant difference could be observed regarding the distance to the camera and the estimation accuracy. However, we could see once more that the nearer a person is, the better the hand detection results.

For the pointing estimation experiment, we asked the same people to point at targets arranged around the camera as in Figure 7.12. The targets are of different height such that also the pitch angle is important in order to estimate the pointing target. The people were asked to point to the targets in the order defined by Figure 7.12. Again we processed all videos obtained this way and estimated the pointing target during the hold phase. The hold phase could reliably recognized in all the videos. However, the

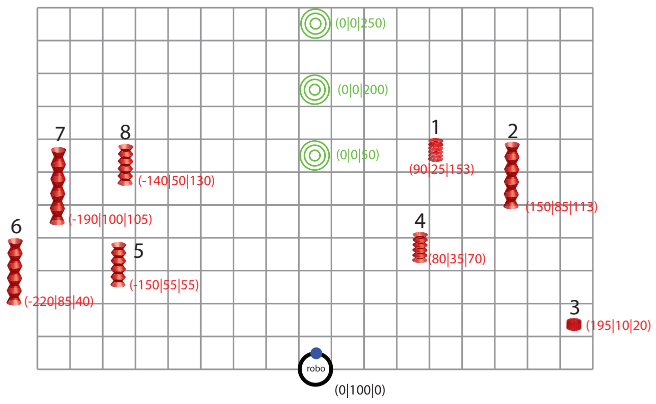


Fig. 7.12: Setup for the parameter estimation experiments. The green circles indicate the positions the people were located when performing the gestures. The red objects indicate the pointing targets. The numbers next to the targets denote the pointing order.

head pose estimation failed in case the distances were 2m or 2.5m most of the time. This is partly due to the low quality images the camera provides. With the resolution of 640

This suggests the use of a second camera with a telephoto lens which could focus on the details of the face whilst the wide angle camera captures the whole scene.

Given that the head pose estimation succeeded, we could estimate the correct target in 80% of the cases. As the pointing targets are relatively close to one another, this is a satisfying result.

8 Conclusion

The ability of a robot to detect and interpret gestures of humans is an important capability for natural interaction. In this work, we presented an approach that is able to robustly recognize gestures from monocular image sequences in real-time. We consider typical gestures performed during an interaction such as nodding, pointing, or waving. To represent people, we locate and track their faces and hands. We use trained classifiers in combination with a skin color cue to reliably detect hands given the color of already detected faces. As the experimental results demonstrate, our system is not only able to locate hands, which is an inherently difficult task, but also to determine their laterality.

The novelty of our approach is that we can recognize a variety of complex gestures using few expressive features extracted out of monocular images. We segment gestures into three phases and train HMMs for each phase separately. We construct HMMs composed of the individual-phase HMMs using regular grammars. Using the distinction between different phases, we are able to estimate parameters of gestures as soon as a certain phase is recognized. As we demonstrate in the experiments, using the trained HMMs, our system is able to reliably spot and recognize the gestures. We furthermore illustrated that parameters of gestures can be accurately estimated.

8.1 Future Work

On each of the individual steps performed along the way to recognize gestures, improvements are possible, a few of which are noted below:

Input Data The quality of the input data is certainly an important issue. With a single focus camera used in this work only a limited range is possible. For the task of head pose estimation, this range is actually limited further. Also, the maximum frame rate of such cameras is mostly limited to 30 fps which may be too little for fast movements. A combined approach, as suggested in Chapter 7, i.e., additionally using a telephoto lens might add a great deal to the robustness of the approach presented in this thesis.

Hand Detection We saw that there actually is an infinite number of possible hand configuration. Therefore, training a *general* hand classifier is an almost impossible task. Without the constraints imposed by the skin-coloured region detection step, the false alarm rate of our classifiers would be unreasonably high. Partly, this might be due to the likewise limited background samples such that there is a lot of unseen background left. One way to improve the detection rate is to train other classifiers on certain classes of hands (which might be clustered automatically based on a certain similarity measure) and apply them on the result of the general classifier. Also, it would be useful to segment the upper body from the background and ascertain that the detected hand is actually connected to the upper body by means of an arm.

Tracking Further improvements regarding tracking are possible. First and foremost, an decrease in processing time per frame should be focus of the very next work. Higher frame rates make the assignment problem much easier (or less likely to fail) and the motion models used for the Kalman filter more appropriate. The knowledge of the upper body could be neatly integrated into the tracking problem as well. Maintaining multiple hypothesis for different assignment probabilities certainly would add robustness. Additionally, tracking a non detected hand has currently its limitations in that once a face is crossed, the hand would be lost as the face is usually larger than a hand. This would lead the mean-shift algorithm to compute the mean in the face center. Other tracking techniques, that rely also on the actual shape, such as the Condensation algorithm [Blake & Isard 1996], could add robustness to the tracking step.

Recognition So far we have only considered a limited amount of gestures and it is clear that ease of separability would decrease with a larger vocabulary. Therefore more training data would be essential which is a cumbersome task as there are no standard databases for this kind of gestures so far. Another way would be to investigate more advanced training techniques which try to maximize the discriminative power of the models by training them together. With the Maximum Mutual Information (MMI) method, an information-theoretic measure is defined which is then maximized during training:

$$I^* = \max_{\lambda} = \left\{ \sum_{v=1}^{V} \left[\log \Pr(X^v|\lambda_v) - \log \sum_{w=1}^{V} \Pr(X^w|\lambda_w) \right] \right\} \tag{8.1}$$

Using such a method, that is, obtaining reliable HMMs with only little training data would also be a good basis to allow for online learning of new gestures. As we have seen, modeling and training a gesture is an elaborate task. By providing the facility of a learning mode, only few examples would be required for the robot to learn this new gesture. An approach in this direction has recently be presented by Rajko et al. [2007] for whole body gestures.

Extending the set of known gestures leads to the question of appropriate feature selection. So far, they have been manually designed to suit the need of the gestures at hand. However, it is not clear whether this would be an appropriate description of other gestures. The idea would be to learn the most discriminative features as well. Principal Component Analysis (PCA) is a commonly used technique to discover the essential information. In [Lu et al. 2007] PCA has been applied to feature analysis, consequently termed Principal Feature Analysis (PFA), in the pattern recognition domain. An adaption to our case would be interesting.

Bibliography

Arthur, David, & Sergei Vassilvitskii [2007]. "k-means++: The Advantages of Careful Seeding". In: *SODA '07: Proceedings of the 18th annual ACM-SIAM Symposium on Discrete Algorithms*. New Orleans, Louisiana: Society for Industrial, Applied Mathematics. ISBN 978-0-898716-24-5. 1027–1035.

Barczak, A.L.C. [2005]. "Toward an Efficient Implementation of a Rotation Invariant Detector using Haar-Like Features". In: *International Conference on Image and Vision Computing New Zealand*. Dunedin, NZ. 31–36.

Bennewitz, Maren, et al. [2007]. "Humanoid Robots, Human-like Machines". In: ed. by Matthias Hackel. Vienna, Austria: I-Tech Education, Publishing. ISBN 978-3-902613-07-3. Chap. Intuitive Multimodal Interaction with Communication Robot Fritz, 613–624.

– [2005]. "Multimodal Conversation between a Humanoid Robot and Multiple Persons". In: *Proceedings of the Workshop on Modular Construction of Humanlike Intelligence at the Twentieth National Conferences on Artificial Intelligence (AAAI), Pittsburgh / USA*.

Blake, Andrew, & Michael Isard [1996]. "The CONDENSATION Algorithm - Conditional Density Propagation and Applications to Visual Tracking". In: *NIPS*. 361–367.

Bradski, Gary R. [1998]. "Computer Vision Face Tracking For Use in a Perceptual User Interface". In: *Intel Technology Journal* Q2. 15. URL: `citeseer.ist.psu.edu/bradski98computer.html`.

Brand, J.D., & J.S. Mason [2000]. "A Comparative Assessment of Three Approaches to Pixel-level Human Skin-detection". In: *International Conference on Pattern Recognition*. 1056–1059.

Brethes, L., et al. [2004]. "Face tracking and hand gesture recognition for human-robot interaction". In: vol. 2. DOI: `10.1109/ROBOT.2004.1308101`. 1901–1906.

Brown, D.A., I. Craw, & J. Lewthwaite [2001]. "A SOM Based Approach to Skin Detection with Application in Real Time Systems". In: *British Machine Vision Conference*.

Campbell, L.W., et al. [1996]. "Invariant features for 3-D gesture recognition". In: *FGR '96: 2nd International Conference on Automatic Face and Gesture Recogni-*

tion. Vol. 00. Los Alamitos, CA, USA: IEEE Computer Society. ISBN 0-8186-7713-9. DOI: `http://doi.ieeecomputersociety.org/10.1109/AFGR.1996.557258`. 157.

Chen, Qing, Nicolas D. Georganas, & Emil M. Petriu [2007]. "Real-time Vision-based Hand Gesture Recognition Using Haar-like Features". In: *IEEE Conference on Instrumentation and Measurement Technology*. Warsaw, Poland, DOI: `10.1109/IMTC.2007.379068`. 1–6.

Crowley, James L., & Joëlle Coutaz [1995]. "Vision for Man Machine Interaction". In: *Engineering for Human-Computer Interaction*. Chapman & Hall. 28–45.

Epanechnikov, V. [1969]. "Nonparametric estimation of a multidimensional probability density". In: *Teoriya Veroyatnostej i Ee Primeneniya*. 156–162.

Freund, Yoav, & Robert E. Schapire [1995]. "A decision-theoretic generalization of on-line learning and an application to boosting". In: *European Conference on Computational Learning Theory*. 23–37. URL: `http://citeseer.ist.psu.edu/freund95decisiontheoretic.html`.

Hong, Pengyu, Thomas S. Huang, & Matthew Turk [2000]. "Gesture Modeling and Recognition Using Finite State Machines". In: *Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition*. Washington, DC, USA: IEEE Computer Society. ISBN 0-7695-0580-5. 410.

Joseph J. LaViola, Jr. [1999]. A Survey of Hand Posture and Gesture Recognition Techniques and Technology. Tech. rep. Providence, RI, USA.

Just, A., O. Bernier, & S. Marcel [2004]. "HMM and IOHMM for the Recognition of Mono- and Bi-Manual 3D Hand Gestures". In: *British Machine Vision Conference*.

Kalman, R. E. [1960]. "A New Approach to Linear Filtering and Prediction Problems". In: *Transaction of the ASME-Journal of Basic Engineering*. 35–45.

Kolsch, M., & M. Turk [2004]. "Robust hand detection". In: *Sixth IEEE International Conference on Automatic Face and Gesture Recognition*. DOI: `10.1109/AFGR.2004.1301601`. 614–619.

Kuhn, Harold [1955]. "The Hungarian Method for the assignment problem". In: *Naval Research Logistic Quarterly* 2. 83–97.

Lee, Hyeon-Kyu, & Jin H. Kim [1999]. "An HMM-Based Threshold Model Approach for Gesture Recognition". In: *IEEE Transaction on Pattern Analysis and Machine Intelligence* 21.10. 961–973. ISSN 0162-8828. DOI: `http://dx.doi.org/10.1109/34.799904`.

Lee, Seong-Whan [2006]. "Automatic Gesture Recognition for Intelligent Human-Robot Interaction". In: *FGR '06: 7th International Conference on Automatic Face and Gesture Recognition*. Washington, DC, USA: IEEE Computer Society. ISBN 0-7695-2503-2. DOI: `http://dx.doi.org/10.1109/FGR.2006.25`. 645–650.

Lienhart, R., & J. Maydt [2002]. “An extended set of Haar-like features for rapid object detection”. In: vol. 1. I–900–I–903 vol.1.

Lu, Yijuan, et al. [2007]. “Feature selection using principal feature analysis”. In: *MULTIMEDIA '07: Proceedings of the 15th international conference on Multimedia*. Augsburg, Germany: ACM. ISBN 978-1-59593-702-5. DOI: `http://doi.acm.org/10.1145/1291233.1291297`. 301–304.

Martin, Chr., F.-F. Steege, & H.-M. Gross [2007]. “Estimation of Pointing Poses for Visual Instructing Mobile Robots under Real-World Conditions”. In: *3rd European Conference on Mobile Robots*. Freiburg, Germany. 223–228.

Martinkauppi, J., M. Soriano, & M. Laaksonen [2001]. *Behavior of skin color under varying illumination seen by different cameras at different color spaces*. 2001. URL: `citeseer.ist.psu.edu/martinkauppi01behavior.html`.

McNeill, David [1992]. *Hand and Mind. What Gestures Reveal about Thought.* Chicago University Press.

Mitra, S., & T. Acharya [2007]. “Gesture Recognition: A Survey”. In: *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 37.3 [May 2007]. 311–324. ISSN 1094-6977. DOI: `10.1109/TSMCC.2007.893280`.

Montero, Jose Antonio, & Luis Enrique Sucar [2004]. “Feature selection for visual gesture recognition using hidden Markov models”. In: *Fifth Mexican International Conference in Computer Science*. Washington, DC, USA: IEEE Computer Society. DOI: `10.1109/ENC.2004.1342606`. 196–203.

Nickel, Kai, Edgar Seemann, & Rainer Stiefelhagen [2004]. “3D-Tracking of Head and Hands for Pointing Gesture Recognition in a Human-Robot Interaction Scenario”. In: *IEEE International Conference on Automatic Face and Gesture Recognition*. 565–570.

Ohta, Y., T. Kanade, & T. Sakai [1980]. “Color Information for Region Segmentation”. In: *Computer Graphics and Image Processing* 13.3 [July 1980]. 222–241.

Ong, Eng-Jon, & R. Bowden [2004]. “A boosted classifier tree for hand shape detection”. In: *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*. 889–894.

Rabiner, Lawrence R. [1990]. “A tutorial on hidden Markov models and selected applications in speech recognition”. In: 267–296.

Rajko, Stjepan, et al. [2007]. “Real-time Gesture Recognition with Minimal Training Requirements and On-line Learning”. In: *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*. Minneapolis, MN, USA,. DOI: `10.1109/CVPR.2007.383330`. 1–8.

Richarz, J., et al. [2006]. “There You Go! - Estimation Pointing Gestures in Monocular Images for Mobile Robot Instruction”. In: *IEEE Int. Symposium on Robot and Human Interactive Communication*. Hatfield, UK. 546–551.

Rigoll, Gerhard, Andreas Kosmala, & Stefan Eickeler [1998]. “High Performance Real-Time Gesture Recognition Using Hidden Markov Models”. In: *International Gesture Workshop on Gesture and Sign Language in Human-Computer Interaction*. London, UK: Springer-Verlag. ISBN 3-540-64424-5. 69–80.

Rime, B., & L. Schiaratura [1991]. “Gestures and speech”. In: *Fundamentals of Nonverbal Behaviour*. 239–281.

Schapire, Robert E. [1990]. “The Strength of Weak Learnability”. In: *Machine Learning* 5.2. 197–227. ISSN 0885-6125.

Sezgin, M., & B. Sankur [2004]. *Survey over image thresholding techniques and quantitative performance evaluation*. 2004.

Shamaie, A., & A. Sutherland [2003]. “A Dynamic Model for Real-Time Tracking of Hands in Bimanual Movements”. In: *International Gesture Workshop*. 172–179.

Swain, Michael J., & Dana H. Ballard [1991]. “Color indexing”. In: *Int. J. Comput. Vision* 7.1. 11–32. ISSN 0920-5691. DOI: `http://dx.doi.org/10.1007/BF00130487`.

Vatahska, T., M. Bennewitz, & S. Behnke [2007]. “Feature-based Head Pose Estimation from Images”. In: *Proc. of the IEEE/RSJ International Conference on Humanoid Robots (Humanoids)*. to appear.

Vezhnevets, V., V. Sazonov, & A. Andreeva [2003]. “A Survey on Pixel-Based Skin Color Detection Techniques”. In: *Proceedings of the Graphicon 2003*. 85–92.

Viola, Paul, & Michael Jones [2001]. “Rapid Object Detection using a Boosted Cascade of Simple Features”. In: *CVPR (1)*. IEEE Computer Society. ISBN 0-7695-1272-0. 511–518.

Welch, G., & G. Bishop [1995]. An Introduction to the Kalman Filter. Tech. rep. University of North Carolina at Chapel Hill.

Wilson, Andrew D., & Aaron F. Bobick [1999]. “Parametric Hidden Markov Models for Gesture Recognition”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21.9. 884–900. ISSN 0162-8828. DOI: `http://doi.ieeecomputersociety.org/10.1109/34.790429`.

Wilson, Andrew David [2000]. “Adaptive models for the recognition of human gesture”. Supervisor-Aaron F. Bobick and Supervisor-Bruce M. Blumberg. PhD thesis.

Xiao-Hui, Liu, & Chua Chin-Seng [2006]. “Rejection of Non-meaningful Activities”. In: *Proceedings of the 7th International Conference on Automatic Face and Gesture Recognition*. Washington, DC, USA: IEEE Computer Society. ISBN 0-7695-2503-2. DOI: `http://dx.doi.org/10.1109/FGR.2006.91`. 189–196.

Yamato, J., J. Ohya, & K. Ishii [1992]. “Recognizing human action in time-sequential images using hidden Markov model”. In: *CVPR ’92: Conference on Computer Vision and Pattern Recognition.* Champaign, IL, USA. DOI: 10.1109/CVPR.1992.223161. 379–385.

Yang, Hee-Deok, A-Yeon Park, & Seong-Whan Lee [2006]. “Human-Robot Interaction by Whole Body Gesture Spotting and Recognition”. In: *ICPR ’06: Proceedings of the 18th International Conference on Pattern Recognition.* Washington, DC, USA: IEEE Computer Society. ISBN 0-7695-2521-0. DOI: http://dx.doi.org/10.1109/ICPR.2006.642. 774–777.

Yang, M.H., D.J. Kriegman, & N. Ahuja [2002]. “Detecting Faces in Images: A Survey”. In: *Pattern Analysis and Machine Intelligence* 24.1 [Jan. 2002]. 34–58.

Yang, Ming-Hsuan, & N. Ahuja [1999]. “Recognizing hand gesture using motion trajectories”. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition.* Vol. 1. Fort Collins, CO, USA.

Yoon, Ho-Sub, et al. [1999]. “Recognition of Alphabetical Hand Gestures Using Hidden Markov Model”. In: *IEICE transactions on fundamentals of electronics, communications and computer sciences* 82.7. 1358–1366. ISSN 09168508. URL: http://ci.nii.ac.jp/naid/110003208304/en/.