Estimating Head Orientation with Stereo Vision

Diplomarbeit

Edgar Seemann

Interactive Systems Labs
Universität Karlsruhe (TH)

Advisors:

Prof. Dr. Alex Waibel
Dr.-Ing. Rainer Stiefelhagen

November 27, 2003

Hiermit versichere ich, die vorliegende Diplomarbeit persönlich und ohne unzulässige Hilfsmittel angefertigt zu haben. Alle verwendeten Quellen sind im Literaturverzeichnis aufgeführt.

Karlsruhe, 30. November 2003

Abstract

Interpretation of human behaviors in video data is essential for natural and intuitive human-computer interfaces. In this context, the estimation of a person's head pose plays a major role, since heads and faces are continuously used in interaction between people.

In this work we present a method for estimating a person's head pose with a stereo camera. Our approach focuses on the application of human-robot interaction, where people may be further away from the camera and may move freely around in a room.

First, the 3D scene is reconstructed from the images of a stereo camera by calculating depth information. Subsequently, the face is extracted with a color-based face tracking approach. Finally, the resulting 3D face model is preprocessed by a number of normalization algorithms. The estimation is based on neural networks, which are trained to compute the head pose from gray scale and depth information.

We show that depth information not only helps improving the accuracy of the pose estimation, but also improves the robustness of the system when the lighting conditions change.

The system can handle pan and tilt rotations from $-90°$ to $+90°$ and achieves high accuracy in a realistic environment. It doesn't require any manual initialization and doesn't suffer from drift during an image sequence. Moreover, the system is capable of real-time processing.

Acknowledgements

This work was conducted at the Interactive Systems Labs as part of my studies at the Universität Karlsruhe (TH). I would like to thank all members of the laboratory for participating in the various data collections performed during this work. I am particularly grateful for the help of Kai Nickel who was always there for advice regarding the stereo camera and implementation details. Furthermore I thank my advisor Rainer Stiefelhagen for his constant support.

Contents

Chapter 1

Introduction

1.1 Motivation

Advancing human-robot interaction has been an active research field in recent years [Perz2001, Agah2001, Koku2000, Adam2000, Mats1999]. A major challenge is the tracking and interpretation of human behaviors in video data, since it is essential for enabling natural human-robot interaction.

In order to fully understand what a user does or intends to do, a robot should be capable of detecting and understanding many of the communicative cues used by humans. This involves in particular the recognition and interpretation of human faces. In interaction between people faces are continuously used to signal interest, emotion and direction of attention. Monitoring such information therefore can be used to make interaction with a robot more natural and intuitive.

Monitoring a person's head orientation is an important step towards building better human-robot interfaces. Since head orientation is related to a person's direction of attention, it can give us useful information about the objects or persons with which a user is interacting. It can furthermore be used to help a robot decide whether he was addressed by a person or not [SYW2001].

While eye gaze certainly is another important cue for a person's direction of attention [Stie2002], in most realistic scenarios image resolution of a person's eyes isn't sufficiently high to see the pupils or irises.

Hence, in this work we propose a method for estimating a person's head pose. Estimation is based on image pairs obtained from a stereo camera.

1.2 Possible Applications

We distinguish three main application areas for head pose estimation:

- Active control applications
- Passive understanding
- Applications for which head pose estimation is a prerequisite

In active control applications a user directly controls an interface with his head pose. He can, for example, use it for pointing when hands are otherwise engaged or as a complementary information when desired action has many input parameters. This is also of particular importance for users with disabilities. Another scenario in this area is car headlight control. If a driver perceives, for example, the silhouettes of pedestrians next to the road, changing the focus of the headlights a little to the road border might help him to better see and avoid dangerous situations.

Passive understanding techniques might be used in dialog applications, where the computer tries to understand what is going on in a room, meeting etc. It is, for example, interesting to know who is talking to whom during a meeting. In human-robot applications it is crucial to identify whether the user was talking to the robot, another person or referring to a different object. For all these applications the head pose gives an indication of the user's focus of attention and therefore helps to understand the context of the dialog.

Applications for which head pose estimation is a prerequisite include face recognition and emotion detection. Knowing the head pose of a person helps finding facial features like eyes, nose, mouth etc. which might be useful to determine a person's identity. These facial features are also crucial for emotion detection. Since these are mostly expressed by the mimic.

In this work, the application we are particularly interested in is a human-robot interaction scenario. A robot should be able to detect the direction of attention of a person in his field of view. In particular, the robot should be

able to determine if a person is looking at him and execute speech commands if he was addressed. Additionally, he should be able to process a user's pointing gestures, which he recognizes via the tracking of hands and the head orientation.

1.3 Requirements for a Head Pose Estimation Technique

Current head pose estimation techniques are still error-prone. They are generally unable to accurately track large rotations under rapid illumination variation, which are common in interactive environments (and airplane or automotive cockpits) [MRC2002].

On the other hand special head tracking hardware (like magnetic sensors) provides an accuracy which is sufficient for most applications. However this hardware is expensive and intrusive. Users often feel discomfort because the hardware restricts their natural motion.

In order to be feasible in practice a head pose estimation technique for human-robot interaction consequently has to satisfy certain criteria. Based on [MA2000] we state these six criteria:

- Non-intrusive (no markers, magnetic sensors etc.)
- Passive
- Robust to occlusions, deformations and lighting fluctuations
- Accurate
- Able to track rotations from -90 to +90 degrees
- Capable of real-time processing

In this work we are proposing a technique which has been designed by taking into account all of these six criteria.

Our focus is especially on robustness under different lighting conditions. This is an obstacle for current image-based techniques. Not only do the color

values vary if there is artificial light or day light, but additionally the image edges and shadows change if the position of the light source is altered.

This is one of the reasons why we propose a technique that uses stereo vision. Depth information is largely invariant to changing lighting conditions and can therefore improve the result of head pose estimation.

1.4 Related Work

Head pose estimation techniques can be divided into two main approaches:

- Feature-Based Head Pose Estimation
- View-Based Head Pose Estimation

Feature-based techniques try to find facial feature points in an image from which it is possible to calculate the actual head orientation. These features can be both obvious facial characteristics like eyes, nose, mouth etc. and "artificially" computed points.

View-based techniques on the other side, try to analyze the entire head image in order to decide in which direction a person's head is oriented.

We will continue to describe and analyze these two main approaches in detail.

1.4.1 Feature-Based Techniques

Feature-based techniques mainly differ in the method they use for finding the facial features. Azarbeyajani et al. [AHP1993] presented a recursive estimation method based on tracking of small facial features using a extended Kalman-Filter framework. Matsumoto and Zelinsky [MZ2000] proposed a template-matching technique for facial feature detection. Six templates of eye and mouth corners are stored and their position is updated each frame.

In general, the calculation of the head pose from the position of the facial feature points is rather straight forward. Usually a 3D head model is used. The feature points of the 3D model are projected onto the image plane. If

the orientation of the 3D model corresponds to the head orientation in the image, the projected feature points must lie close to the found feature points in the image. Matsumoto and Zelinsky [MZ2000] are using stereo vision to calculate the head orientation. The projection onto the image plane is therefore not necessary any more. In this case the problem results in the calculation of a rotation matrix R and a translation vector t which minimize the squared fitting error E in the following equation:

$$E = \sum_{i=0}^{N-1} w_i (Rx_i + t - y_i)^T (Rx_i + t - y_i)$$

where N is the number of features, x_i the coordinate of a feature in the 3D model and y_i the coordinate of a feature acquired via feature detection. w_i is a weighting factor for each measurement. This problem can be solved using least squares.

Feature-based methods usually have the limitation that the same points must be visible over the entire image sequence, thus limiting the range of head motions they can track [YZ2001]. For a human-robot application where users move freely in the room such a limitation is a serious drawback.

Another drawback of this approach is that robust detection of facial features is extremely difficult under unconstrained conditions and large variations in head position. [SB2002]

1.4.2 View-Based Techniques

There is a wide range of view-based approaches for head pose estimation. In this section we want to introduce some of these and approaches and present different techniques they apply.

Template Matching

Template matching is probably the easiest and most straight forward approaches. For each person and angle range a reference image is stored in a database. Captured images are afterwards compared to these reference frames. The reference frame which has the most similarity to the image is

selected as head pose. Since angle ranges of 30 degrees are common, this technique gives only a rough estimate of the head pose.

Ellipse Fitting

Ellipse fitting tries to align an ellipse with the outline of the head. The actual head pose is subsequently estimated from the shape of this ellipse. Ellipse fitting is rather fast, but not very accurate.

Eigenspace-based

Srinivasan and Boyer [SB2002] proposed a head pose estimation technique using view-based eigenspaces. They used principal component analysis to build seven eigenspaces for different angle ranges from example images. Any test image x could afterwards be projected onto an eigenspace by evaluating the inner product of x with the eigenvectors y_i. The norm of the resulting vector $c_i = x^T y_i$ gives the fraction of energy of the test image lying in the eigenspace. The eigenspace with the highest fraction of energy should finally correspond to the angle range of the test image.

Since head orientation can only be matched to one of the seven eigenspaces, this technique gives only a rough estimate of the actual head pose.

Optical Flow and Motion Detection

Optical-flow-based approaches try to estimate the velocity and direction with which a pixel has moved from one image to another. The relative motion of an object can directly be calculated from these pixel velocities. Horn and Schunck [HS1980] have proposed a technique to calculate the optical flow in arbitrary image sequences efficiently. Morency et al. [MRC2002] applied this technique to head pose estimation and extended it for the use with stereo information.

However, calculating the relative head rotation from each image frame to the next, causes an accumulation of the estimation error. This phenomenon is called drift and leads to a degrading performance on long image sequences. Another drawback of this technique is the fact that the initial head pose has

to be known. Manual initialization is therefore required.

Neural-Network-based

Stiefelhagen et al. [SYW2001] proposed a neural network based approach for head pose estimation in human-robot interaction. They normalize the histogram of the face image and map it to neural network input units. Furthermore they are retrieving image edges and feed them as additional information into the neural network.

This technique is accurate when lighting conditions do not change. Under changing lighting conditions, however the performance degrades considerably.

1.4.3 Summary

Feature-based approaches tend to be rather accurate. Moreover facial features can be used for additional applications like face recognition or emotion detection. However, it is difficult to find facial feature points, especially when heads in an image are small and camera resolution is not sufficiently high.

View-based approaches work also if heads are small. Furthermore they do not limit the angle range because of occlusions. The combination with stereo vision, however, is sometimes complex.

Figure 1.1 shows a quick overview of the two main approaches.

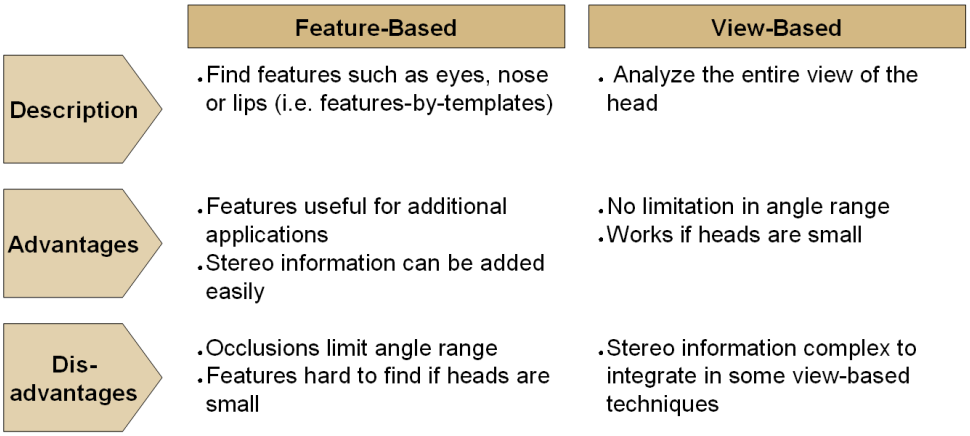


Figure 1.1: Feature-based versus view-based approaches

Chapter 2

The Head Pose Tracking Technique

In order to have a technique which is feasible for human-robot interaction, we have decided to use a view-based approach in our work. This is mainly due to the fact, that view-based approaches promise better performance in non-restricted environments, where the user is allowed to move freely (see 1.4.3).

A neural network approach was chosen because it promised to be accurate, fast and requires no manual initialization. Moreover the stereo information can be added easily.

Figure 2.1 shows the advantages of neural networks compared to other view-based approaches.

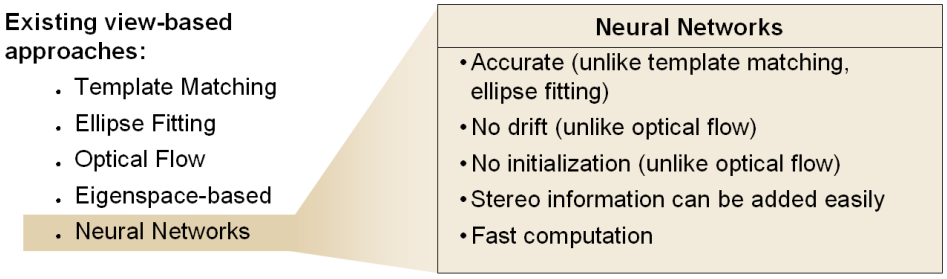


Figure 2.1: Advantages of neural networks

2.1 Overview

The head pose estimation system consists of four main parts (see figure 2.2).

In the first step depth information is calculated from the left and right camera image of the stereo camera. By combining color and depth information we obtain a 3D reconstruction of the scene.

During the second step the head of the user is extracted. We use a sophisticated head tracking technique on the basis of [NS2003]. The technique searches for skin color regions and uses the additional depth information to select the skin color region which is most likely to correspond to the user's head. In the end we obtain a 3D head model of the found head.

The third step transforms the 3D head model into an input pattern for neural networks.

In step four this input pattern is propagated through a trained neural network, which outputs an estimation for the head orientation. The head pose estimates are subsequently filtered to smooth occurring noise.

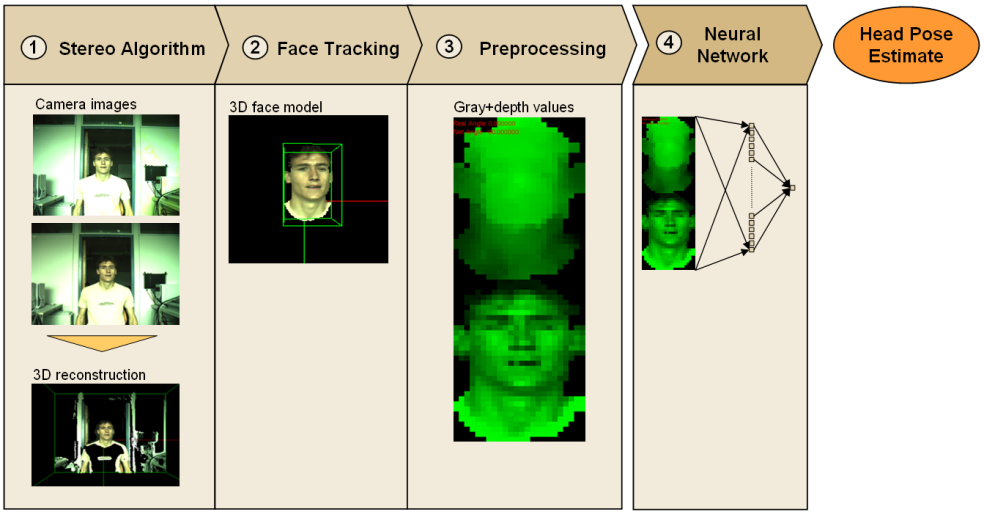


Figure 2.2: The components of the head pose estimation system

In the following sections we give a detailed overview of the system components and their theoretical backgrounds.

2.2 Stereo Vision

In our system a Mega-D digital stereo head from Videre Design was used (see figure 2.3). The camera is capable of resolutions of up to 1280x960 pixels at a frame rate of 15 fps.

Figure 2.3: The stereo camera used in our application

Calibration of the camera is accomplished with a set of checkerboard images (see figure 2.4) and is based on a camera calibration technique developed by Zhang [Zhan2000].

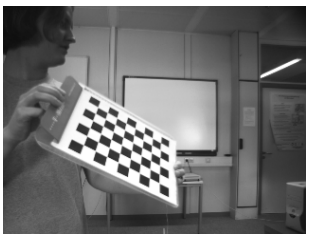


Figure 2.4: Example calibration image

The Small Vision System (SVS) library which is delivered with the camera, provides a computational efficient algorithm for calculating depth information. The algorithm is optimized for real-time computation and will be briefly described in the following subsections.

2.2.1 Stereo Algorithm

The stereo algorithm to calculate the image's depth information can be divided into two parts (see figure 2.5).

First, corresponding pixels have to be found. If we know which pixel in the left image correspond to a pixel in the right image we can determine the real world distance of the object the pixel belongs to. This is done in step two of

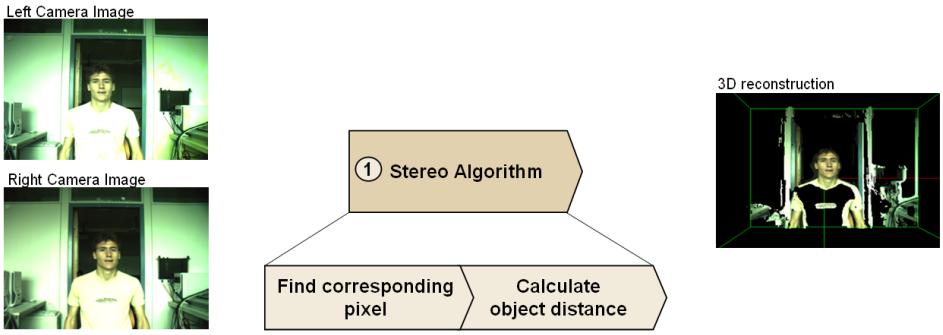


Figure 2.5: The stereo process in detail

the stereo algorithm. We obtain a so-called disparity image, where color values represent the distance of an object (see figure 2.6). The 3D reconstruction of the scene can afterwards be calculated from this disparity image. OpenGL is used to be able to look at the reconstruction under different angles.

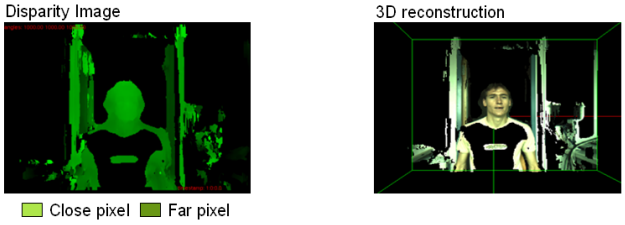


Figure 2.6: Disparity image (left) and 3D reconstruction (right)

2.2.2 Finding Corresponding Pixels

In this processing step for each pixel in the left image a corresponding pixel in the right image is searched. Of course, if there is no texture in a certain image area it is difficult to find a matching pixel, since it might be mistaken with its neighbors. That is why a Laplacian-of-Gaussian feature is computed on each image to enhance edge information.

In order to match a pixel p_l in the left image to the corresponding pixel p_r in the right image, sub windows around these pixels are used. The sub window s_l around p_l stays fixed and we scan for the sub window s_r in the right image with the largest correlation to s_l (see figure 2.7). The correlation is measured by summing the absolute value of differences over the two sub windows.

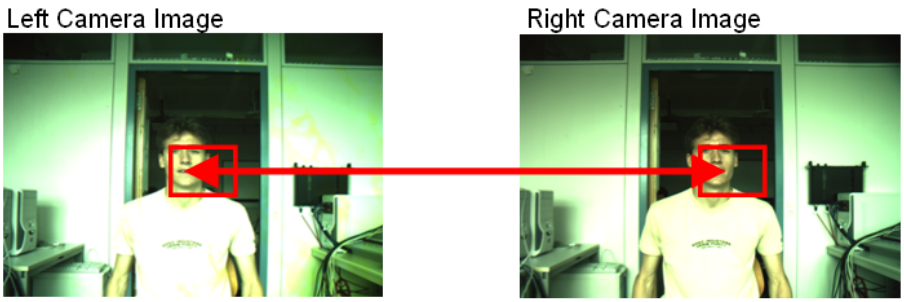


Figure 2.7: Finding corresponding pixels via the comparison of image windows

To double-check if the matched pixels p_r and p_l really belong together, the same procedure is performed the other way round. The sub window s_r around the pixel p_r in the right image stays fixed and the most correlated subunit in the left image is searched. If this results in the known sub window s_l around p_l the search has been consistent and will be accepted. This technique is called left-right check and is particularly effective in eliminating match errors in non-textured regions of the image, and at disparity boundaries.

Finally, post-filtering is performed. A confidence measure based on edge energy is used to eliminate matches whose confidence is below a certain threshold. This threshold can be adjusted manually.

2.2.3 Calculating Object Distance

If we know which pixel in the left image correspond to which pixel in the right image, we can now calculate the distance a pair of pixels. This is accomplished by a simple formula which follows from the intercept theorems.

As you can see in figure 2.8, we have the following interrelationship:

$$\frac{r}{b} = \frac{f}{d_l - d_r} \tag{2.1}$$

where r is the object distance, b the length of the camera baseline, f the focal length of the camera lens. d_l and d_r are the distances of the pixels matched pixels p_l and p_r from the sensor center. The difference $d_l - d_r$ is called disparity (see [Jähn1997]).

The disparity is anti-proportional to the object distance r. The closer an object is, the further away are there images on the two camera sensors. Consequently, if an object gets to close to the camera, the disparity cannot be computed any more. The range in which the distance of an object can be calculated is called horopter.

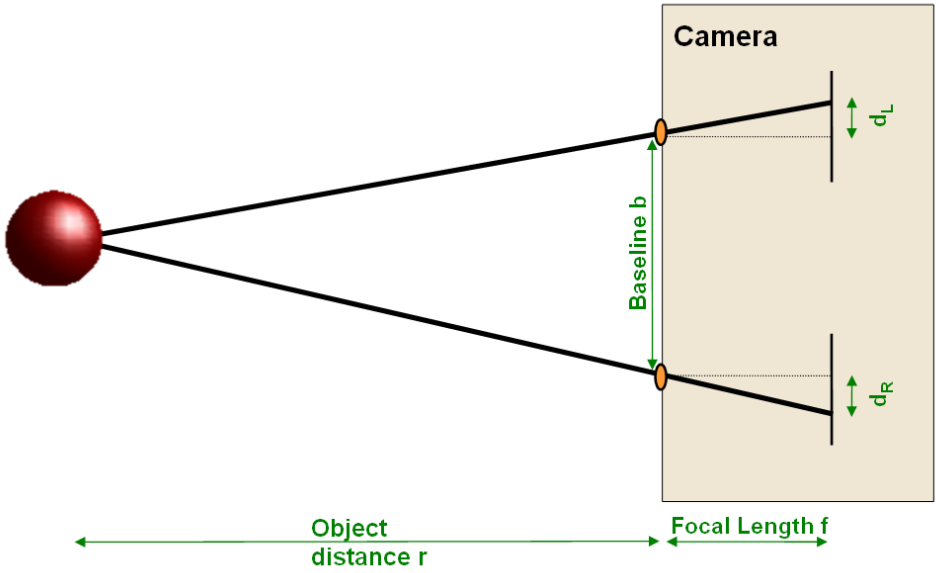


Figure 2.8: Calculating the distance of an object

By multiplying formula 2.1 by b, we obtain the formula for the object distance:

$$r = \frac{b \cdot f}{d_l - d_r} \tag{2.2}$$

2.3 Face Detection and Extraction

Robust face detection is crucial for head pose estimation. Without the correct position of the head no useful head orientation can be computed. In this section a head detection and extraction technique will be presented which is reliable and operates in real-time.

2.3.1 Pattern-Based Vs. Color-Based Face Detection

Two main approaches for face detection are distinguished.

Pattern-based techniques scan for image areas that have the characteristics of a human face. They do not depend on color information and are hardly sensible to illumination changes. Unfortunately, pattern-based techniques tend to be computational expensive, especially on high-resolution images. Moreover, faces may be found on random structured surfaces like trousers, t-shirts etc. A further disadvantage is that non-frontal faces are hard to detect with pattern-based methods.

Color-based techniques on the contrary are rather fast, since they restrict the search area to skin-colored regions. However, to determine these regions a color-model has to be built. Skin color changes under different lighting conditions and the color-model has therefore to be adapted to ensure robustness. An advantage of color-based methods is that they are easily able to detect non-frontal faces. On the other hand, wood and hair are often confounded with skin-colored regions.

Figure 2.9 shows a quick comparison of the two face detection approaches.

In order to incorporate the advantage of both techniques, we implemented a combination of them in our system. A pattern-based face detector is used to find an initial skin-colored region. On the basis of this region a color-model is built. Subsequently, a color-based face detector extracts the face from the image. To improve detection performance the color-based face detector makes use of the depth information obtained from the stereo algorithm.

Figure 2.10 shows an overview of the face detection process.

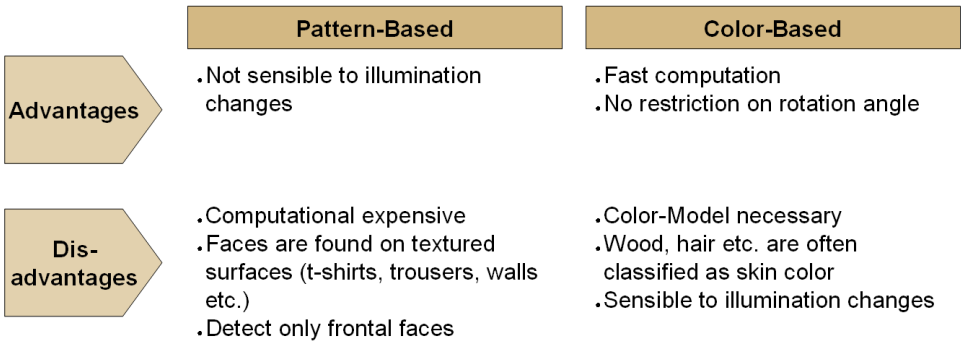


Figure 2.9: Pattern-based vs. color-based face detection

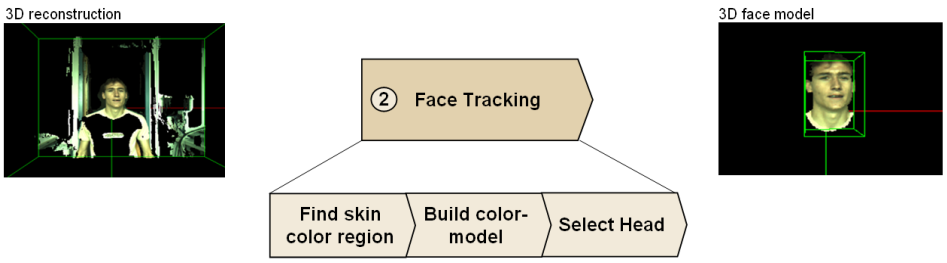


Figure 2.10: The face detection process

2.3.2 Finding A Skin Color Region

A skin-colored region can be used to create a color-model, which is adapted to the current lighting conditions (see 2.3.3). There are various possibilities for retrieving such a region from the image.

The most obvious method is to select a region manually. Since this has to be done only once at the beginning, it is a feasible solution. However, the resulting system wouldn't be passive any more (see system requirements section 1.3).

Another possibility is to analyze the silhouette of a person in the image. As the depth information is available, the background can be separated easily from the rest of the image. With the help of a body model, we afterwards select a foreground object which has the shape of a human silhouette. Subsequently the top part of the silhouette can be chopped off. A drawback of this approach is the fact, that it is not ensured that the person is in a frontal position. Therefore the selected region may contain also non-skin-colored pixels like dark hair. This approach was used in Nickel and Stiefelhagen [NS2003].

The finally implemented method uses a pattern-based face detection algorithm based on [VJ2001]. The algorithm consists of a cascade of simple feature classifiers which are evaluated on different scales of the image. It is able to detect all frontal faces in an image. For a more detailed description of the algorithm see appendix B.1. In order to double check if the found regions really represent a human face, the depth information of the stereo camera is used. We verify not only the face dimensions (height and width), but can also check if the found region is the top part of a human silhouette in the image.

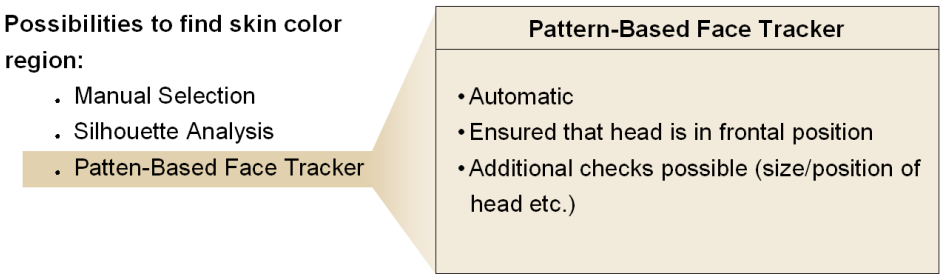


Figure 2.11: Obtaining a skin-colored region

2.3.3 Building The Color-Model

The Chromatic Color Space

Yang, Lu and Waibel [YLW1997] showed that skin color agglomerates in a small region in the chromatic color space (also referred to as rg-space). Therefore skin color classification can be done by defining a skin color distribution in rg-space. Two different representations for skin color distribution are distinguished. Parametric models like e.g. a mixture of Gaussians and non-parametric models like i.e. histograms. In this work an adapted version of the color-model from [NS2003] is used which is based on histograms.

The transformation from the RGB color space to the rg-space is done by the following equations:

$$r = \frac{R}{R+G+B}, \quad g = \frac{G}{R+G+B} \tag{2.3}$$

Colors in the rg-space are intensity normalized, which means that RGB-colors with the same hue, but different intensity values are projected to the same rg-color.

A Histogram-Based Color-Model

Starting from the known skin color region (see subsection 2.3.2), histograms can be build. We define the histograms H_+ and H_-:

$$H_+(x) = \text{Number of } x \text{ in skin color region}, x \in \text{rg-space} \tag{2.4}$$
$$H_-(x) = \text{Number of } x \text{ not in skin color region}, x \in \text{rg-space} \tag{2.5}$$

The histogram value H_+ represents the frequency of a certain color value in the skin color region. Thus, the relative frequency:

$$P_+(x) = \frac{H_+(x)}{n} \text{ with } n \text{ total number of pixels in the region} \tag{2.6}$$

represents the empiric probability for x under the condition that x is skin color. Hence, we have:

$$p(x|skin) = P_+(x) \tag{2.7}$$

However, we are actually interested in the probability $p(skin|x)$, which means the probability for skin color under the condition, that we observe a pixel with color x. Luckily, *Bayes' Rule* helps us to compute this probability:

$$p(skin|x) = \frac{p(x|skin) \cdot p(skin)}{p(x)} \tag{2.8}$$

Like $p(x|skin)$ the terms $p(skin)$ and $p(x)$ can be calculated empirically. $P(skin)$ is just the ratio of the number of pixels in the known skin color region to the total number of pixels n. And $p(x)$ is the ratio of the number of pixels with color x to n.

Analogously, the probability $p(notskin|x)$ can be computed out of the histogram values $H_-(x)$. Consequently, a pixel l with color x is only considered to be skin-colored, if the following holds:

$$\frac{p(skin|x)}{p(not\ skin|x)} > 1 \tag{2.9}$$

Moreover, the higher this ratio, the more probable is pixel l skin-colored.

The above color-model is adapted to the current lighting conditions. If the illumination changes, the color-model has to be readapted accordingly. This is done, when the detection of a head in the image fails for several frames.

2.3.4 Face Extraction

As mentioned in the last subsection the color-model can be used to classify pixels by skin color probability. Figure 2.12 shows the result of such a classification. Darker points represent pixels with high skin color probability, brighter ones pixels with low skin color probability. White pixels represent pixels where equation 2.9 does not hold. In the following, we will refer to this representation as probability map.

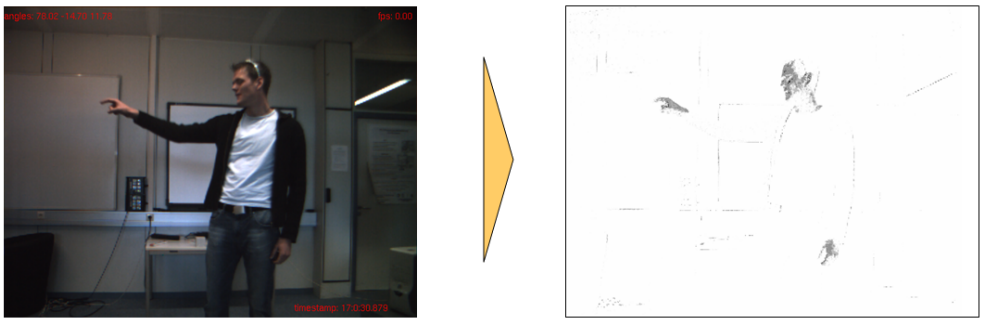


Figure 2.12: Pixels classified by skin color probabilities

Morphological Filtering

In order to obtain skin color regions from the above probability map, we have to find clusters of skin-colored points. For this purpose we use morphological filtering, which forms connected regions in the probability map.

Morphological filtering is based on two operations: dilatation and erosion. The dilatation operation sets the value of a pixel in the probability map to the the maximum of its neighbors. The erosion operation, on the other hand, sets the value to the minimum of its neighbors.

The neighborhood of a pixel can be defined arbitrarily. Common neighborhoods are 4-connectivity or 8-connectivity (see figure 2.13). The neighborhood is also referred to as structuring element.

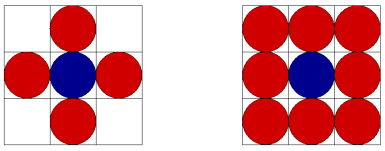


Figure 2.13: Structuring elements for morphological filtering

In our system, we utilize the 8-connectivity structuring element.

On binary images dilatation augments the number of pixels with value 1, erosion on the other side removes them. dilatation and erosion are often applied in combination. An erosion operation followed by a dilatation is called morphological opening, a dilatation operation followed by an erosion morphological closing. Figure 2.14 shows how morphological closing with a 2x1 structuring element removes a hole in an object.

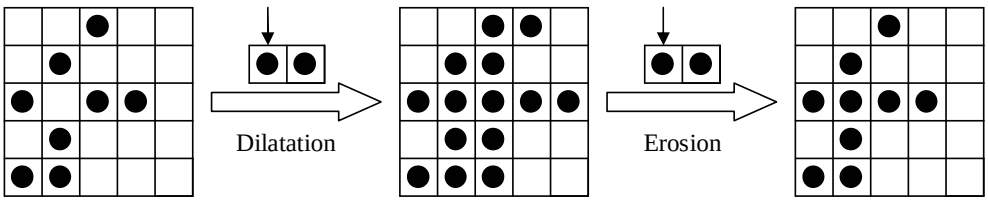


Figure 2.14: Morphological closing with a 2x1 structure element removes hole in object

This is exactly what we want to do in our application. The isolated skin color pixels should be grouped to skin color region or blobs. By applying a morphological closing operation we can obtain these connected skin color regions (see figure 2.15).

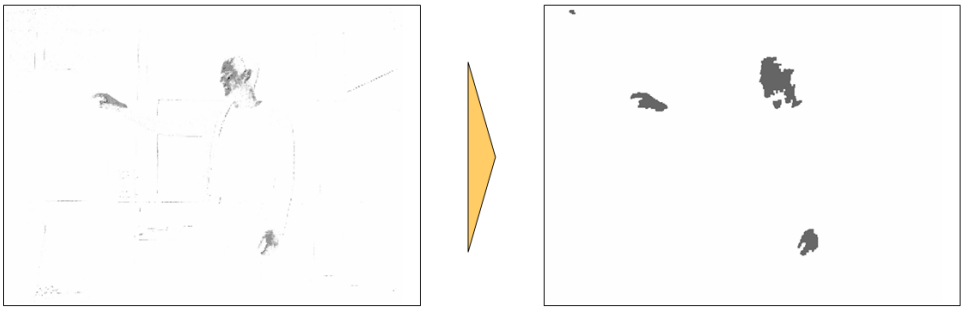


Figure 2.15: Skin color blobs resulting from morphological filtering

Verifying Position and Dimensions

What still needs to be done, is the selection of the skin color blob, which corresponds to a head in the original image. This is accomplished by verifying the position and dimensions of the blobs.

In section 2.2 it has been showed how the distance of an object can be determined with stereo vision. Knowing the distance of an object enables us not only to calculate the 3D position of the object, but also to compute the real-world size of the object. As for the distance calculation this can be done by applying intercept theorems:

$$\frac{ps}{f} = \frac{rs}{b} \tag{2.10}$$

	mean	variance
head width	0.20	0.04
head height	0.275	0.04
head area	0.03	0.013
head ratio	1.41	0.35

Figure 2.16: Parameters of the body model

where ps is the size in pixels of the object, f the focal length of the camera, b the object distance and rs the real-world size. Multiplying equation 2.10 by b yields a formula for the real-world size of an object.

The real-world position and size of an object can be compared to a predefined body model. In accordance to [NS2003] the body model is defined via Gaussians with the following parameters:

For each blob a head score S is computed by simply multiplying the probabilities resulting from the different Gaussians of the body model (see equation 2.11).

$$S = \prod_i \frac{1}{\sigma_i \sqrt{2\pi}} \exp^{\frac{(x-\mu_i)^2}{2\sigma_i^2}} \tag{2.11}$$

with σ_i and μ_i the parameters from the body model displayed in figure 2.3.4.

Subsequently, the blob with the highest head score is selected. Additionally we check whether the y-coordinate is above a certain threshold, since we do not expect faces to occur close to the ground.

The presented technique for head detection is robust and real-time processing is possible. Another advantage of this technique is the fact, that we not only get a bounding box of the found face, but also the face mask. Meaning that we also know the contour of the found face. As we learned from head pose estimation techniques like ellipse fitting, the contour of the face is an important cue in the estimation of head orientation.

When combining the found face blob, with the depth information from the stereo algorithm, we obtain in the end a 3D face model of the person in the image (see figure 2.17). This model can be used to estimate the head pose.

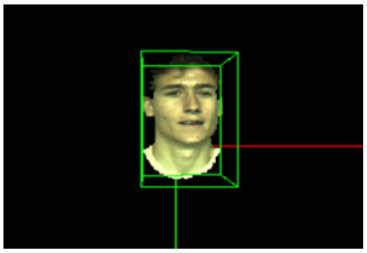


Figure 2.17: 3D face model obtained by face detection and extraction algorithm

2.4 Preprocessing

In order to prepare the the head data obtained during face extraction for the neural network, the face model has to be converted and mapped to the input units of the neural network. This process is divided into four small steps (see figure 2.18). Step 1 consists of scaling the face to a fixed size. Step 2 executes a downsampling of the face resolution. In step 3 the z-values of the face pixels are normalized. Finally, step 4 converts the color values to gray values and normalizes them with histogram-normalization.

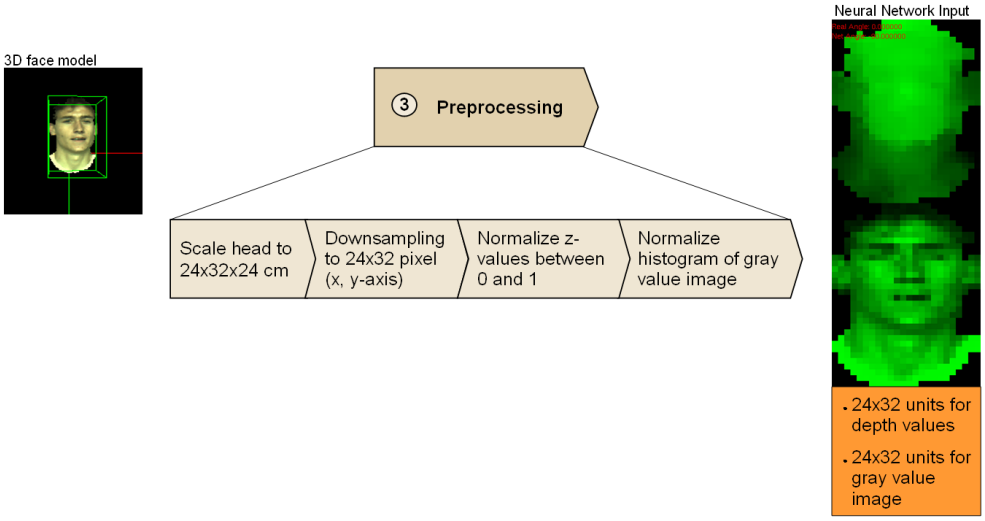


Figure 2.18: Mapping the 3D face model to the input units of a neural network

2.4.1 Resizing 3D Face Model

As we will see in section 2.5 neural network input patterns have a fixed size. Consequently, if we want to map a found head to the input units, we should resize it to a specific size. Otherwise, the head may be mapped only to a subset of the input units of the neural network. In its training phase the neural network would try to learn these size variations, which is clearly not what it is supposed to learn.

The resizing of the head is done by an affine transformation. We start from the 3D reconstruction of the face, which consists of the 3D position of the face's pixels. The 3D position of the pixels is given in real-world coordinates. Firstly, the pixels are translated to the origin of the coordinate system. Then,

we scale the face by multiplying the pixel coordinates with a diagonal transformation matrix. For the new 3D coordinate of a pixel, we obtain the following formula:

$$\begin{pmatrix} x_{new} \\ y_{new} \\ z_{new} \end{pmatrix} = \left(\begin{pmatrix} x \\ y \\ z \end{pmatrix} - \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \right) \cdot \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{pmatrix} \tag{2.12}$$

where t_x, t_y, t_z denote the translation vector to the origin and s_x, s_y, s_z denote the scaling factors in x-, y- and z-direction.

Since we use 24x32 input units for the neural network the scaling factors are calculated such that the bounding box of the head is resized to 24x32cm. Remember that the pixel are given in real-world coordinates. The bounding box is defined as the smallest box to enclose all pixels of the face model.

2.4.2 Downsampling

To obtain exactly 24x32 gray respectively depth values from the face model, we divide the bounding box of the head into a three dimensional grid with grid distance 1 cm. For every 1 cm cube within this grid, we calculate a representative pixel out of the information of all pixels in the cube.

This representative pixel may be computed in various ways, e.g.:

- taking the average of colors and coordinates
- taking the median of color and coordinates
- taking the values of a random pixel

Since pixels in a cube tend to be very similar, in our system we simply selected a random pixel.

As a result of this step we obtain a face model consisting of solely 24x32 pixels in 3D space.

2.4.3 Depth normalization

Before the depth values are normalized, color and depth information are separated. This needs to be done, because the activation of the input units of the neural network may only be described by a single value. Hence, in the following we map the depth information of the face pixels to 24x32 input units of the neural network and we map the color model to another 24x32 input units of the neural network. In total we therefore obtain a neural network input layer with 2x24x32 units.

As we will see in section 2.5 each neural network unit has an associated activation function. The activation function we use in our system is a logistic function yielding values between 0 and 1. Therefore our initial activations (the activations of the input units) should also be in the range of 0 to 1. An affine transformation in z-direction scales down our face model accordingly.

2.4.4 Gray Value Normalization

The intensities of gray values differ under changing lighting conditions. As for the size variations, we do not want the neural network to learn different intensity levels. Consequently a way of compensating these differences has to be found.

In our system a technique has been implemented, which tries to equalize the gray value histogram H of a input pattern. It starts off by building the accumulated histogram M. M is defined in the following manner:

$$M(x) = \int_{z \leq x} H(z)dz \quad \text{for the continuous case} \tag{2.13}$$

$$M(x) = \sum_{z \leq x} H(z) \quad \text{for the discrete case} \tag{2.14}$$

The new gray value x_{new} of a pixel with the current gray value x is subsequently computed from M with the below formula:

$$x_{new} = M(x) \tag{2.15}$$

As a result of this mapping, we obtain a histogram of the new gray values x_{new} which is distributed equally. This is due to the fact that the values x_{new} change faster in areas where M grows rapidly. Since rapid growth of M at x corresponds to a high number of pixels with gray value x, this kind of mapping spreads the new gray values in this region and therefore lessens extrema of the histogram.

A further enhancement of this technique is also utilized in [HB1995]. There, this technique is used not only to equalize histograms, but also to match an image's histogram to an arbitrary distribution. They call this technique histogram matching.

2.5 Estimating Head Pose With Neural Networks

In the previous sections we have seen how the data has been prepared for the final estimation of the head pose. Now, we want to have a look at how the estimation is actually performed with a neural network.

For a quick theoretical introduction to neural networks and neural network learning algorithms please refer to appendix A.

2.5.1 Neural Network Topology

When dealing with neural networks, we first have to decide which topology we want to use. Determining an optimal topology is a difficult problem and has been studied extensively (see for example [SM2002] and [Matt2002]).

Usually one starts out by first determining the number of input units. In most cases their number is rather obvious, since there have to be as much input units as the number of elements in the data vector we want to classify. However, the more input units a neural network has the more training data is necessary to train it appropriately. Consequently, techniques for reducing the dimensionality are often applied to the initial data vector in preprocessing.

In our system, the dimensionality reduction has been performed by simply sampling down the 3D face model to the relatively small size of 24x32 points. In other applications techniques like principal component analysis (PCA) are used for this purpose.

Thus, in our system the input layer of the network consists of 1536 units, which corresponds to 24x32 units for the gray value information and 24x32 pixels for the depth information. As we will see in chapter 3, additionally we trained a network which uses only the gray value information. This network has 768 input units.

The next problem that is usually considered, is the number output units. Again, this number is in most cases straight forward, since there have to be as much output units as classes which we try to distinguish.

For our application, this means that we would have to divide our continuous

output space, the rotation angles between $-90°$ and $+90°$, into angle ranges. Consequently, an output unit corresponding to a certain angle range would have a high activation, if the real rotation angle is contained in this angle range. All other output units might also have an activation, however, their activation should be smaller.

In experiments, however, a function approximation approach proved superior results. Unlike neural network classification, in this case only a single output unit is used. The activation of this output unit isn't interpreted as a probability measure for classification, but directly as the desired rotation angle. Hence, during training the neural network was provided by an input pattern containing gray value and depth information and the target angle for the output unit normalized between 0 and 1.

Another performance improvement has been obtained by training separate networks for each degree of freedom. Thus, in the final network for estimating one of the rotation angles, the network contained 1536 input units and a single output unit.

What still needs to be decided is the number of hidden layers and hidden units the neural network should contain. As mentioned in appendix A feed-forward networks with two-layers of weights and sigmoidal activation functions can approximate any decision boundary to arbitrary accuracy (see [Krei1991]), if the number of hidden units is sufficiently high.

Consequently, a neural network containing the 1536 input units, one output unit and a hidden unit layer with a so far undefined number of units should be able to estimate the angles accurately. A pre-condition, however, is that there is a sufficiently large training data.

Our experiments showed that an amount of 60 to 80 units in the hidden layer of the neural network was suited best for estimating the head orientation accurately. In the experiments each unit layer was fully-connected with the successive layer. This means that each unit of the input layer was connected to each unit of the hidden layer and each unit of the hidden layer was connected with the output unit. Surprisingly, even with a much higher and smaller number for the hidden units, the network still achieved rather good results.

To sum it up, our neural network contained three layers of units:

- 1536 units in the input layer

- 60-80 units in the hidden layer
- 1 unit in the output layer

These layers were fully connected and contained only forward connections, meaning that there are no cycles of weights in the network.

Figure 2.19 shows a picture of the network.

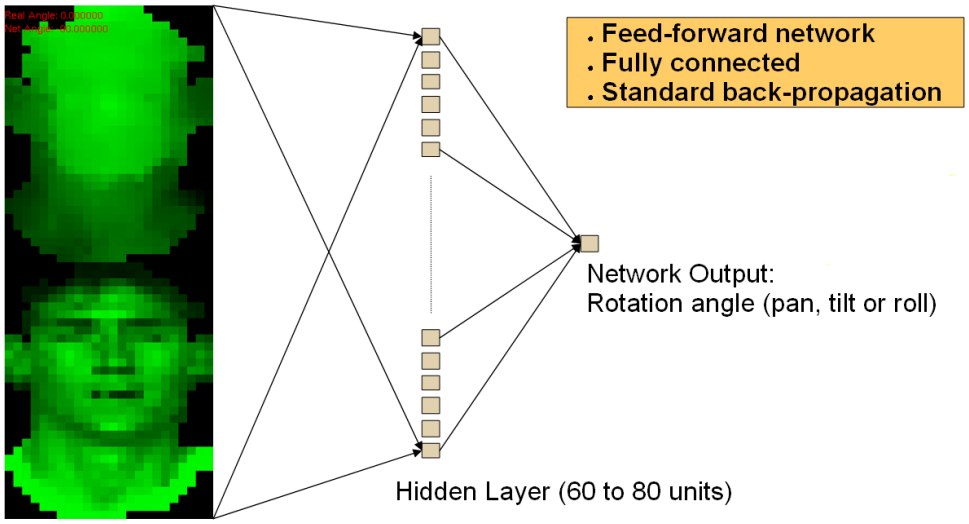


Figure 2.19: The topology of the neural network

2.5.2 Advantages Of This Approach

There are several advantages of this approach. Firstly, there exist a powerful and computational efficient algorithm for neural network learning: error-backpropagation. Next to the standard back-propagation algorithm which was used in this work, there are even more sophisticated learning algorithm like, for example *QuickProp* or *RProp*. These algorithms improve the convergence speed during neural network training.

Unlike other head pose estimation techniques (see section 1.4), the neural networks do not estimate the relative head rotation from one frame to another, but directly compute the orientation from a single image frame. That is why the estimation errors aren't accumulated over an image sequence. This

effect is called drift and is a significant drawback of approaches using, for example, optical flow. Furthermore when only relative rotations are estimated, the initial head orientation of a person has to be known. Thus, a manual initialization is required. For neural networks no manual initialization is necessary.

Another advantage is, that the above network topology does not divide the estimation in angle ranges or classes. Consequently, the real head orientations can be approximated very precisely.

Once neural networks are trained, they are extremely fast in computation. The activation levels of the input patterns have simply to be propagated through the three layers of the network. Hence, they are well suited for a real-time head pose estimation technique.

Chapter 3

Experimental Results

For evaluating the system described in the previous chapter, video data sets in different environments have been recorded. On the acquired data, a number of experiments have been run and the system's performance has been analysed. In the following we'll describe the results of these experiments and the conditions under which the video data has been recorded.

3.1 Data Collection "Portrait View"

The "Portrait View" data collection has been recorded in a relatively restricted environment. People were sitting in about 2 meter distance in front of the stereo camera. Therefore the position of the head in the image did not change considerably.

However, the people's movement wasn't restricted in any way. They were free to move their heads in pan, tilt and roll direction (see figure 3.1).

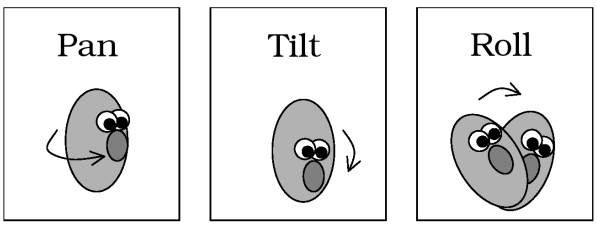


Figure 3.1: Pan, tilt and roll angles (image based on [Fitz2001])

Since one of our main goals was to improve robustness under changing lighting conditions. The data was recorded under two different illuminations. One consisted of a room illuminated by day light, the other was illuminated artificially with neon lamps. In order to obtain an even stronger effect of the illumination change, we tried to place an additional lamp next to the person. This was done to intensify the shadows in the face. Shadows shouldn't have an effect on the stereo reconstruction, but certainly affect the angle estimation with a conventional image-based technique.

Figure 3.2 shows some sample pictures from the data collection. One can easily see the difference in lighting conditions.

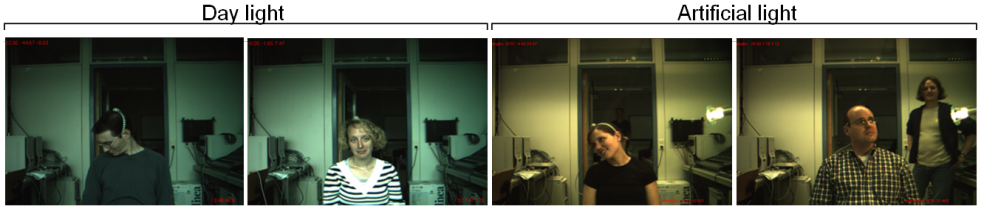


Figure 3.2: Sample images from the “Portrait View” data collection

In total we recorded image sequences of 10 persons looking around freely. The image sequences consisted of 250-500 pictures and were recorded under both of the lighting conditions described above. The image resolution was 640x480 pixels.

In order to be able to train the neural networks, the real head orientations for each image had to be recorded somehow. To accomplish this, we used a magnetic sensor, which was mounted on the person's head. It recorded reference angles for pan, tilt and roll direction.

For the evaluation we mainly focused on the pan angle. Pan direction is on the one hand the direction where the most movement occurs, on the other hand the pan angle seems to be the most useful angle for identifying the object a person is focusing on.

Figure 3.3 shows the histogram of the recorded pan angles in the image sequences. As you can see, more training data is available for angles ranges close to 0.

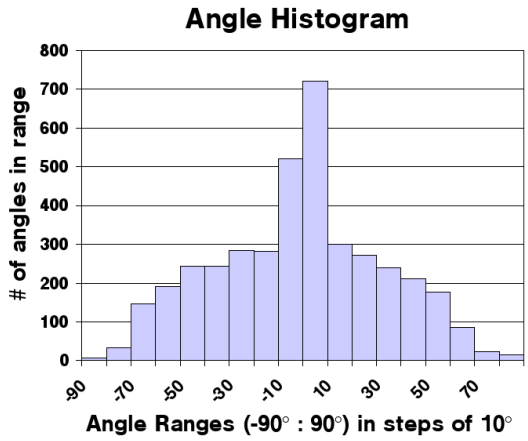


Figure 3.3: Angle histogram for pan direction in the ”Portrait View” data collection

3.2 Experiments “Portrait View”

Three experiments have been performed for the “Portrait View” data collection. We evaluated the system’s performance on known users, unknown users and under changed lighting conditions. The experiments focus on the estimation of the pan angle, however, we provide the results for the tilt angles as well.

Furthermore we compared the results to the work of Stiefelhagen [Stie2002]. He developed a head pose estimation technique which is based on neural networks, as well. Gray value images of similar size as in our application serve as neural network input. However, instead of depth information obtained from a stereo camera, he uses edge information to further improve his results.

Even though Stiefelhagen’s results were obtained with another data set, we argue that they are comparable with ours. Firstly, this is due to the fact, that they were recorded under similar conditions and achieved with a similar neural network. Secondly, on gray value images, Stiefelhagen’s results are almost equivalent to ours. It therefore can be assumed that the data set he used in his system, is equally difficult for head pose estimation.

3.2.1 Test 1 - Known Users

For the known user test, we divided the whole data set from the data collection into three different parts:

	% of data
Training set	80%
Cross evaluation set	10%
Test set	10%

The division was done by choosing the cross evaluation and test set randomly from the data set.

In order to have results, which are comparable to the system of [Stie2002] and to see the difference of performance with and without the calculated stereo information, we ran the known user test three times with different input patterns:

1. Patterns consisting of histogram normalized gray value images (24x32 pixels)
2. Patterns consisting of depth images (24x32 pixels)
3. Patterns consisting of both gray value and depth images (2x24x32 pixels)

Possibility 1 corresponds to taking just the lower part of the input pattern described in section 2.4. Possibility 2 corresponds to taking the upper part.

Figure 3.4 shows the results obtained with the different input patterns.

The mean deviation from the reference angles using patterns which contain solely gray values is 4.2°. The system of [Stie2002] achieves slightly better results. This is due to the fact, that these results were obtained with a different data set, which seems to be easier for the neural networks.

When only depth information is used in the input patterns, the neural networks are less accurate. We obtain only a mean deviation of 5.1° from the reference angles.

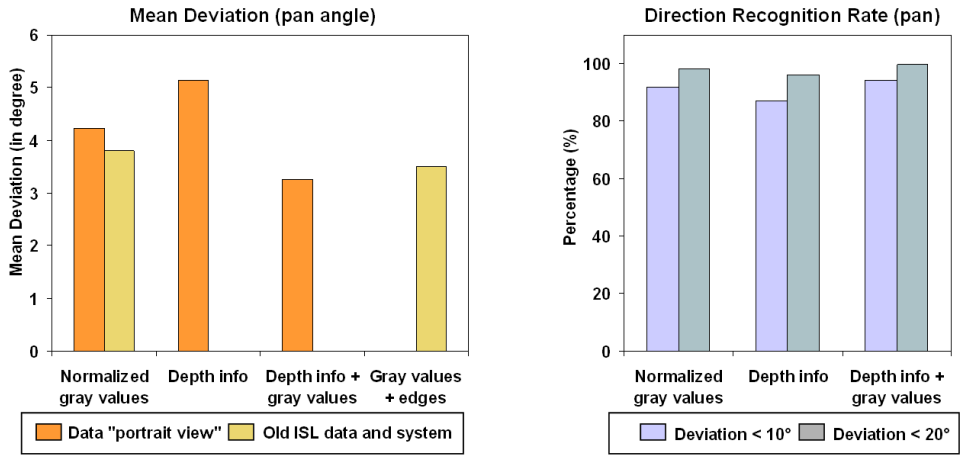


Figure 3.4: Mean deviation from the reference angle for head pan and direction recognition rates for known users

	mean deviation
gray values	4.2° / 2.9°
depth info	5.1° / 3.8°
depth+gray	3.2° / 2.6°

Table 3.1: Mean deviation for the pan/tilt angle

Now, if we combine both gray value and depth information, the performance of the system is further improved and a deviation of 3.2° is achieved.

Stiefelhagen also managed to improve his results by adding images of the horizontal and vertical images to the input patterns. But even though his data set seems to be slightly easier for the neural networks, he achieved only a mean deviation of 3.5°.

For the the tilt angles, we obtain even better results (see table 3.1). This is mainly due to the fact that there was less head movement in tilt direction. Thus, there are no large errors during estimation.

Consequently, we can deduce that the addition of depth information is better able of improving the performance of head pose estimation than the addition of edge information.

Next to the mean deviation from the reference angle, we analysed a value called direction recognition rate. The direction recognition rate was defined in the following manner. If the estimated angle differs by less than 10°

respectively 20° (see figure 3.4) from the reference angle, we consider the direction to be recognized by the neural network.

Under these circumstances we obtain recognition rates of 91.8% and 98.1% (depending on the allowed deviation) with patterns consisting of gray value images. The depth information alone leads to recognition rates of 87% and 95.9%. Neural networks that use patterns with a combination of gray value and depth information recognize 94.3% and 99.7%.

As you can see the estimation of the pan angle for known users is rather accurate. However, for practical applications, we do not want the system to depend on the user. This would imply to retrain the network for every new user. Since neural network training is computational expensive we want to avoid retraining.

In the next subsection we are evaluating the system's performance on new users.

3.2.2 Test 2 - Unknown Users

In order to evaluate the performance of the neural networks on new users, we trained them using the "Leave-One-Out method". This means that we trained the neural network on 9 persons and tested it on the remaining person. Since performance fluctuates depending on the choice of the remaining person, the training was done for every combination possible. The final results presented in figure 3.5 represent the average of the 10 training and test cycles that have been performed.

For new users the system's mean deviation from the reference angle is 9.6° with gray value input patterns. Patterns containing the depth information achieve a mean deviation of 11°, whereas combined patterns consisting of both depth and gray value information achieve only 7.2° mean deviation.

Stiefelhagen achieved with his system a mean deviation of 7.5°, which is close to what we achieved with our system. However, if we compare the result achieved with gray value patterns again, it seems that his data is slightly easier for the neural networks.

As for the known user case, the estimates for the tilt angle provide better results. The mean deviations are printed in table 3.2.

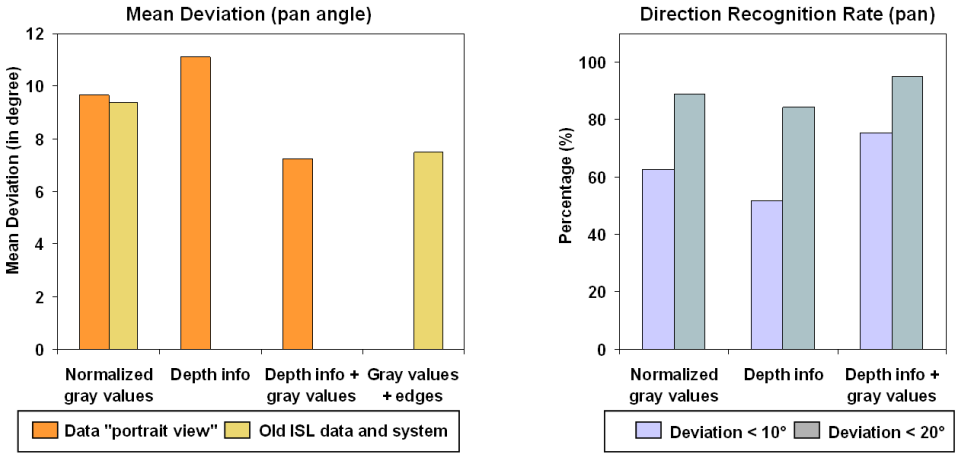


Figure 3.5: Mean deviation from the reference angle for head pan and direction recognition rates for unknown users

	mean deviation
gray values	9.6° / 8.8°
depth info	11.0° / 7.6°
depth+gray	7.5° / 6.7°

Table 3.2: Mean deviation for the pan/tilt angle for unknown users

These values are considerably worse than the results for known users. This is due to a number of circumstances. On the one hand the heads of person differ in appearance and aspect ratio. For example, some people's heads are rather longish, whereas others have heads which are quite broad. On the other hand, the algorithm who extracts the heads from the image (see section 2.3) might consider a person's hair to belong or not to belong to the head. Especially for people with long hair this is an issue.

As mentioned above, the presented values are average values. The range of the mean deviation was rather high. We observed values form 5° mean deviation up to 10.5° mean deviation depending on the person (with depth-gray input patterns).

As a matter of course the accuracy of the direction recognition rate decreased also when compared to the known user case. With gray value patterns recognition rates of 62.3% and 88.9% have been achieved. Depth information alone led to a recognition rate of 51.8% and 84.3%. Depth and gray value information together resulted in an direction recognition accuracy of 75.2% and

95.1%.

It can be concluded that the use of depth information improves head pose estimation for unknown users, as well. Depth information alone, however, is not sufficient for an accurate estimation.

3.2.3 Test 3 - Changed Lighting Conditions

Changing lighting conditions are one of the main problems of image-based techniques and particularly neural networks. Neural networks approximate functions, which map the neural network input to the output. Now, if the input changes due to a different illumination, the learned function isn't the same any more.

As Stiefelhagen [Stie2002] has shown, histogram normalization helps compensating changing lighting conditions to a certain amount. However, the results are still rather bad.

As we argued in the introduction of this work, depth information should be greatly invariant towards illumination changes. Therefore, we trained a neural network under some lighting conditions and afterwards tested the already trained network under the new illumination conditions.

Figure 3.6 shows the results obtained in this case.

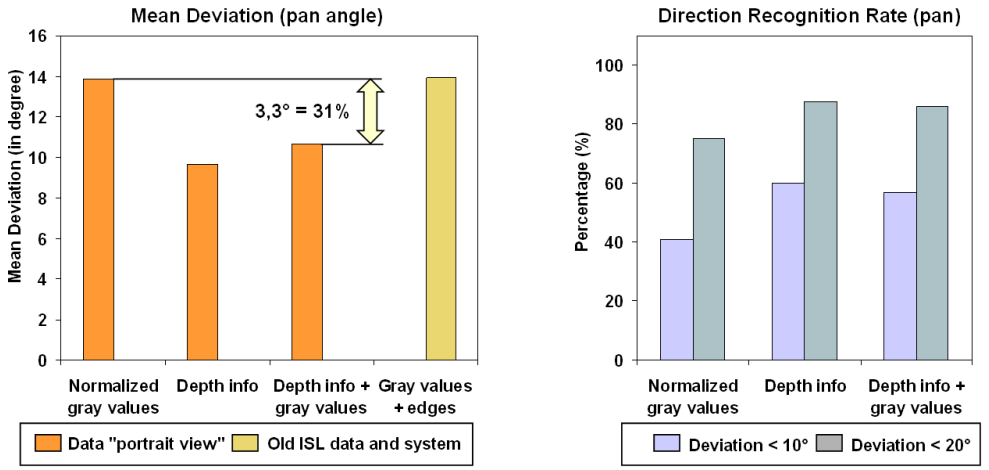


Figure 3.6: Mean deviation from the reference angle and direction recognition rates for unknown users

The mean deviation from the reference angle with gray value patterns increases to 13.9°. The combination of gray value and depth information leads to a mean deviation of 10.6°, whereas under these circumstances depth informations is with 9.6° mean deviation the most stable.

This is exactly what we expected. Depth information is indeed rather invariant to illumination change and improves head pose estimation under the new conditions considerably. In fact, when we compare this result to ”Test 2” with unknown users from above, it can be seen, that the mean deviation is the same.

Stiefelhagen achieved with his system a mean deviation of 13.8°. Edge information seems to be less suited for changing lighting conditions. This seems to be reasonable, since edge information is strongly influenced by the illumination conditions. Shadows, for example, produce edges in the face image, that are not really existent in the three dimensional scene.

For the direction recognition rates, the following results have been obtained. 40.7% and 75.1% accuracy for gray value input patterns. Depth and gray value information yielded an accuracy of 56.9% and 86.2%. As for the mean deviation the best results, however, have been obtained with depth information alone. Here the direction recognition rates were 60% and 87.6%.

We conclude that depth information is suited best for head pose estimation under changing lighting conditions. To have a versatile method working well under all conditions, we propose nevertheless to combine depth and gray value information for head pose estimation. With this configuration the conventional intensity image-based approach is still 31% worse.

3.2.4 Error Analysis

Error analysis cannot only help to understand where and why the errors occur, but also be useful to improve the performance of the estimation.

So, in order to better understand the head pose estimates, we analysed the pan estimation errors for different pan and tilt angle ranges. We expected the error of the estimation to be worse for large angles. On the one hand because less training data was available, on the other hand because less of the face is visible and therefore it might be more difficult for the neural network to find appropriate features.

Figure 3.7 shows the the pan estimation errors for known users divided in pan angle ranges.

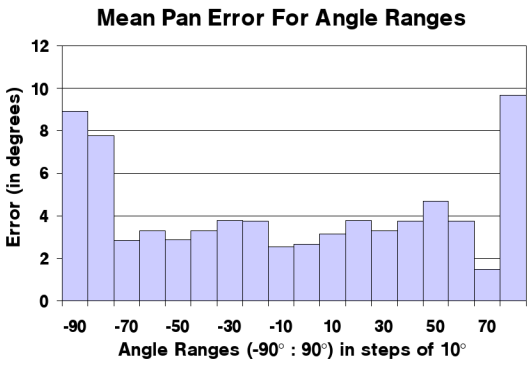


Figure 3.7: Pan estimation error for the different pan angle ranges

The expected effect is visible. The estimation errors are indeed slightly higher for large pan angles. So far, we don’t yet know whether this is due only to the lack of training data or not. We therefore postpone a further analysis to section 3.4.3 where we analyse the errors for the second data collection. There, the number of recorded images is distributed almost equally for all pan angles.

For unknown users, it can sometimes be observed that the estimation is displaced, meaning that estimated angles are, for example, to small most of the time. This is due to the fact that the shape of faces differ from person to person. A neural network which is trained on a number of persons can therefore commit the same error for every pan angle, if the face shape of the new person differs from the mean face of the training samples. Figure 3.8 shows an error histogram which shows this effect.

Another question was, whether the pan angle estimation errors differ for different tilt angles. In fact, the estimation might be expected to work better for small tilt angles than for large ones. However, this effect couldn’t be verified in our experiments. Obviously, for the neural network it is sufficient if the number of training samples is high for all tilt angle ranges.

Figure 3.9 shows the distribution of the pan estimation errors for the different tilt angle ranges for an unknown user.

For the “Portrait View” data collection it can be concluded, that the estimation angles of the neural network are accurate, if sufficient training samples are available. Further results have to be postponed to the analysis of the

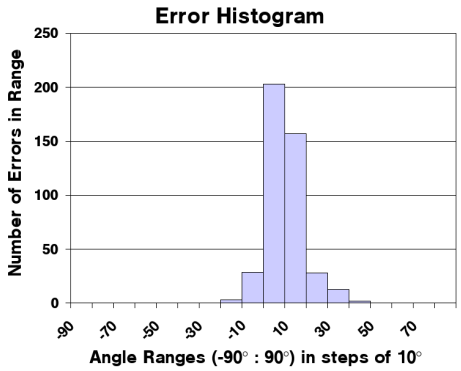


Figure 3.8: Error histogram for an unknown user

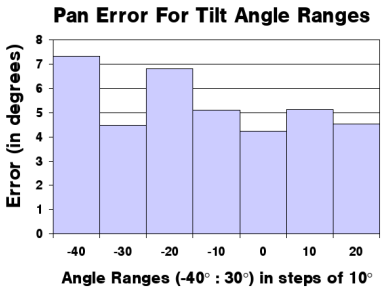


Figure 3.9: Pan estimation error for the different tilt angle ranges

“Robot Scenario” data collection.

3.3 Data Collection “Robot Scenario”

As mentioned above, in the “Portrait View” data collection, we have a relatively restricted environment. For a human-robot interaction scenario, however, it cannot be assumed that the user is sitting directly in front of the camera. That is why another data set has been collected under a more realistic environment.

During the “Robot Scenario” data collection the users were standing further away from the camera. Moreover, they were free to move around in the room. As image three in figure 3.10 illustrates, the user could, for example, move to the side or go closer to the camera. The head movement, of course, wasn't restricted in any way neither.

Since we later want to incorporate the recognition of hand gestures in the system. The users were additionally asked to execute pointing gestures on pre-defined targets in the room.

Figure 3.10: Sample images from the ”Robot Scenario” data collection

A total of six users have been recorded under these conditions. The data sequences consist of about 1000 images per person. Even though the heads in the images were smaller than for the ”Portrait View” data collection, the image resolution wasn't changed and remained at 640x480 pixels. Consequently the afterwards extracted head models consisted of fewer pixels. But since the data is downscaled for the neural network anyways, this shouldn't have an effect on the system's performance (as long as the heads are still sufficiently large).

As for the “Portrait View” data collection the reference angles were captured with a magnetic sensor, which was mounted on the user's head. Again we captured pan, tilt and roll angle for every frame in the data sequence. For the evaluation we focused on the pan angle.

Figure 3.11 shows the histogram of the recorded pan angles in the image sequences. Unlike for the “Portrait View” data collection the histogram values for the different angle ranges are pretty much the same. This is due to the pointing gestures the users executed. The targets the users had to point at, were spread equally in the room. Since people tend to look at the target they are pointing to, the head orientations were also distributed equally.

3.4 Experiments “Robot Scenario”

Under the new conditions in the “Robot Scenario” two experiments have been performed. One experiment evaluated the neural network's performance on known users, the other on unknown users.

The goal of these experiments was to compare the system's performance to

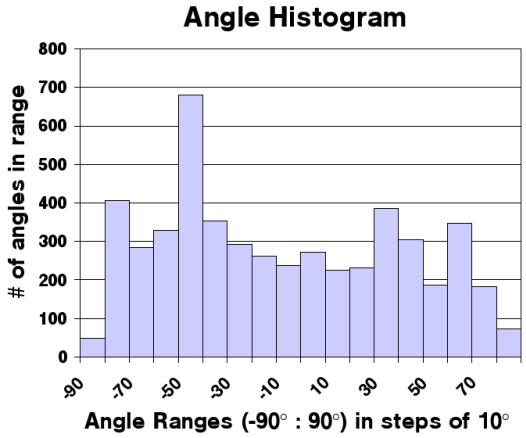


Figure 3.11: Angle histogram for pan direction in the “Portrait View” data collection

the results under the more restricted conditions in the “Portrait View” data collection. There, the system was doing well and it had yet to be shown, if the system was capable to operate under a realistic environment.

3.4.1 Test 1 - Known Users

The known user experiment was carried out with similar conditions as in section 3.2.1. Hence, 80% of the data served as training set, 10% as cross evaluation set and 10% as test set.

Figure 3.12 shows the results of the experiment.

The mean deviation from the reference angle is 4.6° for gray value patterns. With depth information alone, we achieve only a mean deviation of 8°. Obviously the 3D reconstruction of the head becomes worse, if the head is small in the stereo image. This is due to the fact that the depth resolution of the stereo algorithm becomes worse the farther away an object is. This effect is clearly visible when we look at the 3D reconstruction in an OpenGL window, where the head can be viewed under various perspectives. Consequently, we do not expect the depth information to improve the head pose estimation considerably. The obtained results with gray value and depth information are therefore only slightly better than gray value information alone. The mean deviation for this case is 4.3°.

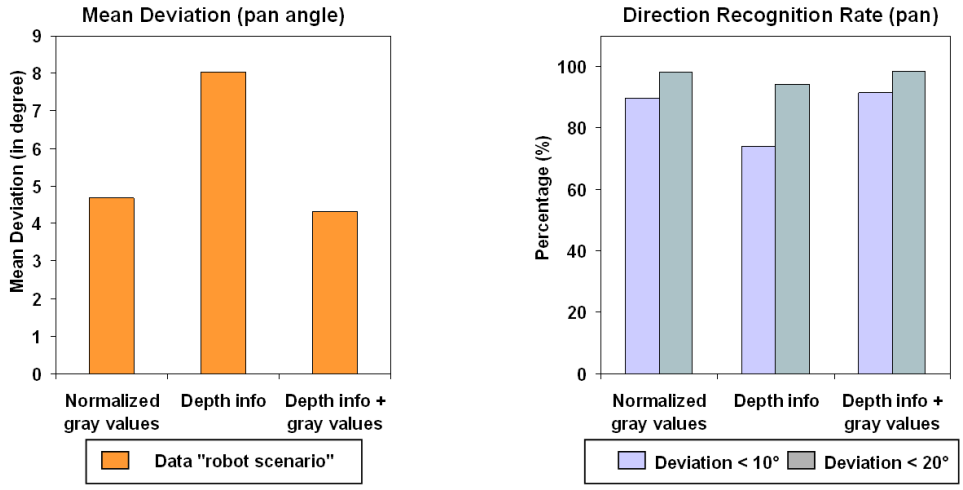


Figure 3.12: Mean deviation from the reference angle and direction recognition rates for known users

	mean deviation
gray values	4.6° / 2.4°
depth info	8.0° / 3.3°
depth+gray	4.3° / 2.1°

Table 3.3: Mean deviation for the pan/tilt angle for known users

The results for the tilt angle are shown in table 3.3.

The direction recognition rates with gray values as input for the neural network are 89.53% and 98.26%. Input patterns with depth information lead to recognition rates of 73.98% and 94.37%. The best result is achieved by combining depth and gray value information. For this case the neural networks recognize 91.47% and 98.64% of the directions.

Compared to the “Portrait View” data collection, we have seen that the results for the less restricted conditions get slightly worse. Particularly affected is the depth information whose quality is worse with the “Robot Scenario” data. However, even in the “Robot Scenario” the performance of the system is still quite good.

3.4.2 Test 2 - Unknown Users

For the test on unknown users, we applied the “Leave-one-out” method again. Hence, in this case we trained the neural nets on 5 persons and tested them on the remaining one.

Figure 3.13 shows the obtained results.

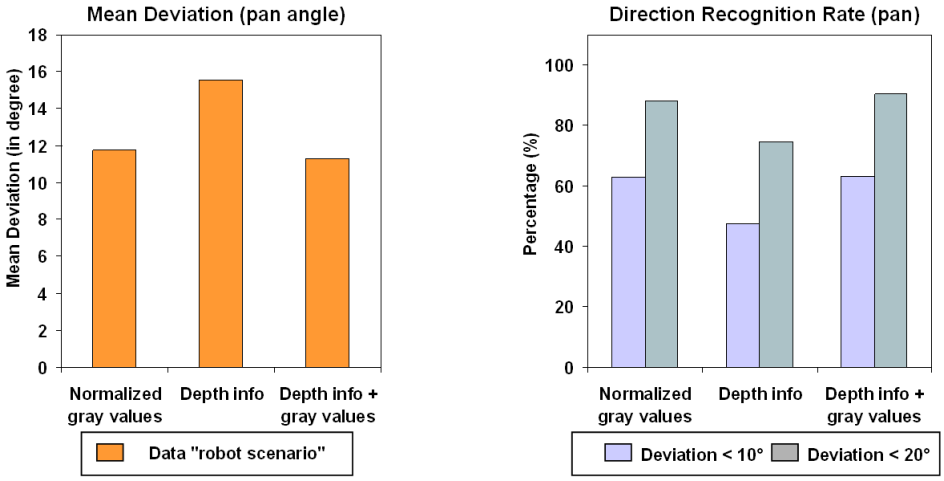


Figure 3.13: Mean deviation from the reference angle and direction recognition rates for unknown users

As expected the performance on unknown users was considerably worse. The mean deviation for neural networks which use gray value images as input patterns was 9.9°. Input patterns containing the depth information achieved a mean deviation of only 15.5°. As mentioned above, this is due to the fact that the stereo reconstruction becomes worse for objects which are far away from the camera. Consequently, combining depth information with gray value information did only improve the mean deviation a little to a value of 9.7°.

The results for the tilt angle are shown in table 3.4.

The direction recognition rates were 62.75% and 88.29% for gray value patterns. With depth information alone 47.39% and 74.66% recognition rate were achieved. The results for patterns containing depth information and gray value information have been 63.12% and 90.44%.

Even though the mean deviation from the reference angle is considerably

	mean deviation
gray values	15.5° / 6.3°
depth info	11.0° / 5.7°
depth+gray	9.7° / 5.6°

Table 3.4: Mean deviation for the pan/tilt angle for unknown users

higher. The estimated angles can still give a robot a good hint on where a person is looking. Recognition rates of up to 90% seem to make the method applicable in practice (see also chapter 4.1).

3.4.3 Error Analysis

As for the “Portrait View” data collection, we analysed the pan estimation errors for the different pan and tilt angle ranges.

Figure 3.14 shows the pan estimation errors for the various pan angle ranges.

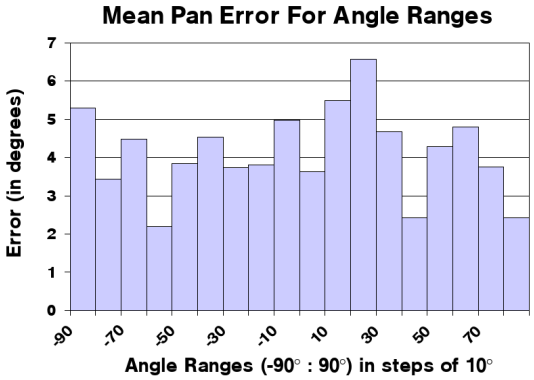


Figure 3.14: Pan estimation error for the different pan angle ranges

Unlike for the “Portrait View” data collection, in this case, we do not observe higher estimation errors for large angles. Obviously, this time there is sufficient training data available to estimate all pan angles accurately.

For the different tilt angle ranges, we obtained the similar results as for the “Portrait View” data collection. The estimation error obviously doesn't depend on the tilt angle. However, it could nevertheless be useful to train separate neural networks for different tilt angle ranges. Thus, we would have more specialized neural networks, which might achieve better results.

We conclude that the neural network's head pose estimation is almost equally accurate for all pan and tilt angle ranges. Obviously the neural network is able learn the head orientations well even if the rotation angles are large.

3.4.4 Filtering

During the analysis of the estimated rotation angles, we observed that the neural networks pose estimates are rather noisy (see figure 3.15).

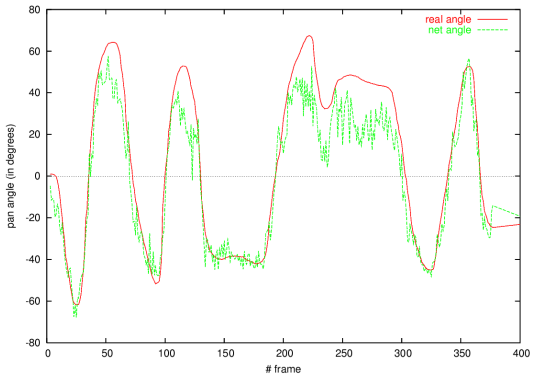


Figure 3.15: Estimation of the rotation angles is rather noisy

In order to further improve the estimation results, it therefore seemed to be useful to filter the neural network output with some filter technique. Kalman filters are widely used for such tasks and have proven excellent performance. Given the nature of our application, the smooth movement of head in time, Kalman filters should perform well on our application data, too.

The Kalman filter estimates the state $x_k \in \mathbf{R}^n$ of process, that is governed by a linear stochastic difference equation:

$$x_k = Ax_{k-1} + w_{k-1} \tag{3.1}$$

with a measurement $z \in \mathbf{R}^m$ that is

$$z_k = Hx_k + v_k \tag{3.2}$$

The w_k and v_k represent the process and measurement noise. For a more detailed description of the Kalman filter, please refer to appendix C.

Hence, in order to implement a Kalman filter for our application, we have to define a state vector x_k, a process matrix A, a measurement vector z_k and a measurement matrix H. Moreover, we have to know something about the nature of the measurement and process noise w_k and v_k.

Obviously, in our application the pan angle at time step k can be calculated from the pan angle at time step $k-1$ and the rotation velocity at time step $k-1$. Consequently, we can define the state of the process by a two-dimensional vector consisting of the pan angle n_k and the rotation velocity l_k.

Equation 3.1 yields in this case:

$$x_k = \begin{pmatrix} n_k \\ l_k \end{pmatrix} = Ax_{k-1} + w_{k-1} = \begin{pmatrix} 1 & dt \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} n_{k-1} \\ l_{k-1} \end{pmatrix} + w_{k-1} \tag{3.3}$$

with dt the time difference between time step k and time step $k-1$.

With the values for a_{21} and a_{22} we could model additional velocity changes. However, since we know nothing about a person's behavior a modeling of these parameters isn't possible. It is therefore assumed that the velocity is constant. We obtain: 1

$$A = \begin{pmatrix} 1 & dt \\ 0 & 1 \end{pmatrix} \tag{3.4}$$

A measurement in our application consists solely of the angle output of the neural network. H is therefore the simple 1×2 matrix $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$.

For applying the predictor-corrector algorithm of the Kalman filter (see appendix C.2), what still needs to be determined are the covariance matrices Q and R of the process and measurement noise.

The measurement error covariance may be calculated by taking some sample measurements. In our application, the measurements correspond to the output of the neural network. Since, we also have the real head orientations from our magnetic sensor, we can calculate the measurement error covariance easily from our data.

The process noise is somewhat more complicated, since the user may move his head arbitrarily fast or slow. However, we can deduce a mean process noise from our recorded data. In this case, the filter achieves worse performance, if a user moves for example too fast.

In our tests the Kalman filter was able to improve the performance of the head pose estimation. In the “Robot Scenario” we achieved the following results for the mean error on new users:

Without Kalman	9.7°
With Kalman	9.1°

This is a relative improvement of 6.2%.

Figure 3.16 displays the result of the Kalman filter graphically.

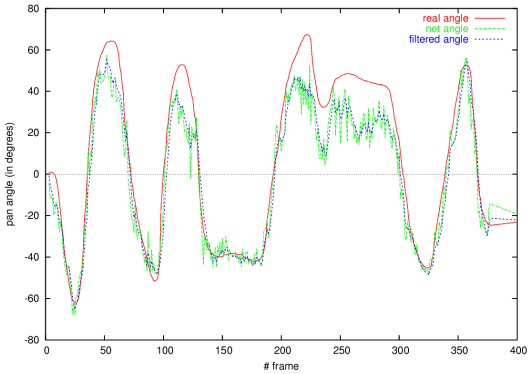


Figure 3.16: The Kalman Filter smooths the angle estimates

The result of the Kalman filter still isn't very smooth. If we adjusted the process and measurement covariance of the filter to make it change slower, we would obtain a smooth curve. However, in this case, the Kalman filter wouldn't be flexible enough and the overall error would increase considerably.

Chapter 4

Head Pose Estimation in Applications

4.1 The Real-Time System

Real-time capability is one of the crucial points in computer vision. In human-computer interaction, the estimation of head pose only makes sense, if the robot can immediately respond to it and thus if the estimation can be done in real-time.

In order to prove the practicability of our approach, we implemented a real-time system of for the head pose estimation technique. In our tests, we achieved calculating 10 frames per second with a resolution of 320x240 pixels (Pentium 4, 2, 8 GHz). This is due to the fact, that once neural networks are trained, they are extremely fast in computation. The activation levels of the input patterns have simply to be propagated through the three layers of the network. Face tracking with the color-based technique presented in section 2.3 is rather fast as well. Taking skin color as selection criterion restricts the search space for algorithm considerably. The only issue for our system is the calculation of the depth information. The stereo algorithm is computational expensive and restricts the frame rate considerably.

Figure 4.1 shows a screenshot of the real-time system. The two small windows in the upper right corner display one of the current camera images and the skin color regions found in it. The windows below and left of these windows show the 3D reconstruction of the found head and the subsequently calculated

input pattern for the neural network. The slider at the bottom displays the current head orientation (pan angle). The control panel on the left may be used to load different trained neural networks, change the camera exposure etc.

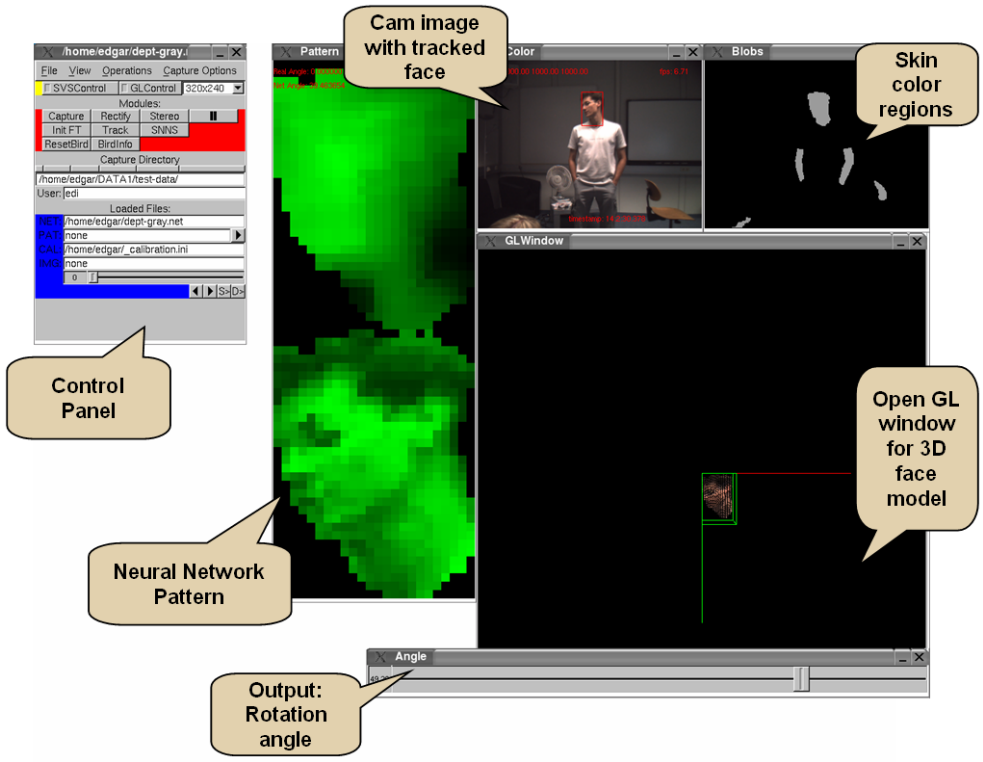


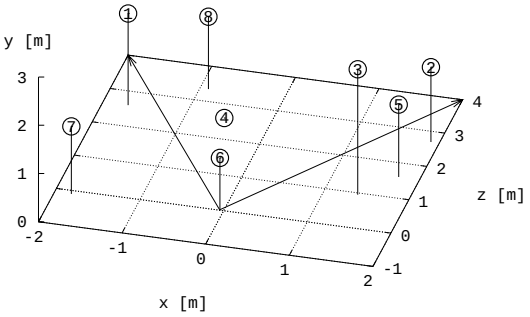
Figure 4.1: Screenshot of the real-time system

4.2 Tracking Of Pointing Gestures With The Help Of Head Orientation

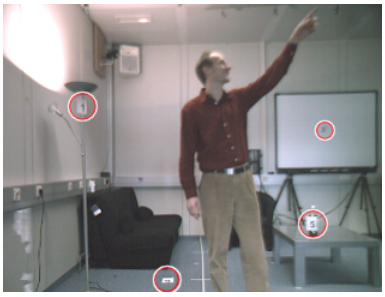
As argued in the introduction of this work, for intuitive human-robot interaction the robot needs to know a person's attention and intention. Pointing gestures are often used by people to indicate directions or to refer to objects.

It has been observed that pointing gestures are often accompanied by a head movement in the same direction. Combining pointing gesture recognition and head orientation estimation should therefore lead to a more accurate and robust system.

Nickel and Stiefelhagen [NS2003] proposed a framework to track pointing gestures with hidden markov models. In their setup, they marked several targets, at which a user had to point (see figure 4.2). They computed the number of pointing gestures, which have been recognized. Moreover they calculated the difference of the target angle to the angle of the recognized gesture (angle error).



Targets



Setup

Figure 4.2: Targets and setup of the gesture recognition system (images taken from [NS2003])

In order to improve their recognition results, head orientation information was added to the feature vector. Table 4.1 shows the obtained results.

	Recall	Precision	angle error
No Head-Orientation	79.8%	73.6%	19.4°
Sensor Head-Orientation	78.3%	86.3%	16.8°
Estimated Head-Orientation	78.3%	87.1%	16.9°

Table 4.1: Performance of gesture recognition with and without including head-orientation to the feature vector

The pointing gesture recognition without any head orientation information achieved a recall of 79.8%. The precision was 73.6% and the average error from the target angle to the estimated pointing angle was 19.4°.

When head orientation information from a magnetic sensor was added, the recall decreased by 1.5%. However, the precision increased significantly by an absolute 12.6%. Moreover the angle error was only 16.8°. Obviously, the head orientation information does improve the recognition performance of the pointing gesture system.

Surprisingly, even though the head orientations from the magnetic sensor are more accurate than the ones estimated with the neural network, the results improve slightly. The precision increases by 0.8%.

It can be concluded that the visual estimation of head orientation is capable of improving the results of the pointing gesture recognition.

Chapter 5

Conclusion and Future Work

In this work, a new approach to head pose estimation was presented. It is based on neural networks and uses depth information to improve estimation results. The approach was evaluated under various conditions including the performance on known, unknown user and under lighting fluctuations. Furthermore different scenarios have been considered: A "Portrait View" scenario where the user only moves his head and a "Robot Scenario" which is more realistic and allows the user to move around freely.

The results obtained with the proposed approach are rather promising. Depending on the environments, a mean deviation between 3.2° and 9.7° has been achieved. Direction recognition rates range between 90.44% and 99.7% for the various scenarios, if a deviation of 20° is allowed. The accuracy seems to be suitable for pratical applications.

We have seen, that stereo information helps improving the estimation of the head orientation angles considerably. Moreover it improves the robustness of the system under changing lighting conditions. Kalman Filtering can further improve the obtained results.

We proved the practicability of the approach by implementing a real-time head pose estimation system which operates at 10 frames per second. The implemented system has proven to be enhance human-robot interaction. It has, for example, been used to improve the results of a pointing gesture recognition system by adding the estimated head pose to the feature vector [NSS2004].

Even though adding stereo information has improved the robustness of the system significantly, the performance still degrades noticeably under new lighting conditions. Here further work is necessary.

Further improvements of this approach could be obtained incorporating information about the tilt angle into the neural network for estimating the pan angle. This could, for example, be accomplished by training three different pan estimation networks for large, average and small tilt angles. A neural network trained on the tilt angle could be used to determine which network to use.

Other enhancements could be made in the preprocessing stage. By applying techniques like principal component analysis, the dimensionality of the input patterns could be reduced and important information extracted. This should lead to better recognition results especially for unknown users.

Speed improvements could be achieved by calculating depth information only on a small window around the current position of the head. Only in this area depth information is necessary for head pose estimation. This could lead to real-time capability even on higher resolutions.

Appendix A

Neural Networks

A.1 Introduction To Neural Networks

Artificial Neural Networks are inspired by the way biological nervous systems, such as the brain, work. They are composed of a large number of highly interconnected processing elements (neurones) working together to solve a specific problem.

Until now, neural networks have been applied to a great number of problems. This is due to the fact that neural networks are a very versatile technique. They have been particularly popular in the area of pattern recognition and image analysis.

A.2 Advantages Of Neural Networks

Neural networks have the remarkable ability to derive meaning from complicated or imprecise data. They may be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can generalize, meaning that it is not only able to reproduce what it has been trained, but also to interpolate new results.

Other advantages include [SS1996]:

- **Adaptive learning**: An ability to learn how to do tasks based on the data given for training or initial experience.
- **Self-Organization**: A neural network can create its own organisation or representation of the information it receives during learning time
- **Real Time Operation**: Neural network computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.
- **Fault Tolerance via Redundant Information Coding:** Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

A.3 Building Blocks Of Neural Networks

A neural network consists of units (neurones) and links or connections between these units. In analogy to activation passing in biological neurons, each unit receives the output of its prior units as input.

Figure A.1 shows a small sample network.

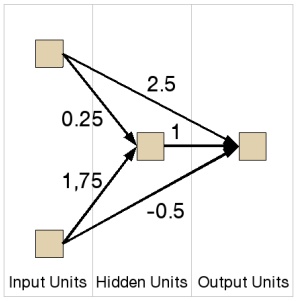


Figure A.1: A simple neural network

The direction of a connection shows the direction in which the activations are passed. Each connection has an associated weight. The effect of a unit on its successor is influenced by this weight. Negative values decrease the activity of the succeeding units, positive values enhance it.

Depending on the function of a unit, we distinguish three different types. Units which have no predecessors are called input units. Their activation

levels are set by the application/problem data itself or are fixed (for example, special "ON" units). Units whose output represents a part of the result of a neural network computation are called output units. For classification problems there is usually one output unit for each possible class. All remaining units are called hidden units, since their inputs can not be directly set and their outputs not directly observed.

The actual information processing within the units is accomplished by the activation and the output function which are associated with each unit.

First, the activation function computes the net input from the weighted outputs of the prior units. Then, the new activation level of the unit is calculated. The general formula for an activation function is:

$$a_j(t+1) = f_{act}(net_j(t), a_j(t), \theta_j) \tag{A.1}$$

where:

$a_j(t)$	activation level of unit j at time/step t
$net_j(t)$	net input in unit j at time/step t
θ_j	threshold (bias) of unit j

The net input $net_j(t)$ is computed with

$$net_j(t) = \sum_i w_{ij} o_i(t) \tag{A.2}$$

where:

$o_i(t)$	output of unit i at time/step t
w_{ij}	weight of the connection between unit i and j

The output function takes the activation value as input and subsequently computes the output of the unit. The general formula is:

$$o_j(t) = f_{out}(a_j(t)) \tag{A.3}$$

with $a_j(t)$ and $o_j(t)$ as defined above.

In the literature, there often doesn't exist both an activation and an output function for a neural network unit. In this case there is only one process-

ing function per unit defined, which is also called activation function. This function receives the net input as input parameter.

A.4 Learning With Neural Networks

A.4.1 Linear Discriminant Functions and Single-Layer Networks

Single-layer neural networks consist of a single layer of adaptive weights. They can be used to distinguish two or more classes with linear decision boundaries.

Consider a linear discriminant function y of the form of A.4, which separates two classes C_1 and C_2.

$$y(x) = w^T x + w_0, x \in \mathbf{R}^d \tag{A.4}$$

$y(x) = 0$ represents a hyperplane in the d-dimensional space $\mathbf{R}^d$. The vector w is chosen to fulfill the following formula:

$$y(x) = \begin{cases} < 0 & \text{if } x \in C_1 \\ > 0 & \text{if } x \in C_2 \end{cases}$$

Such a decision boundary can be modeled with a simple neural network as shown in figure A.2 (left) in the following manner. The weights of the network are defined as the values of the vector w, the activation function f_{act} of the output unit is defined as y and the output function as the identity.

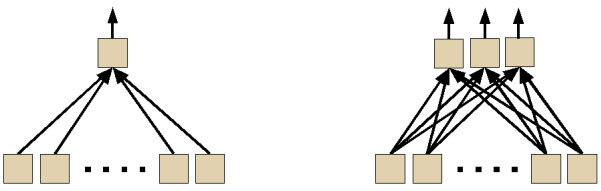


Figure A.2: Single-layer neural networks

To distinguish more than two classes we can extend the above formula. We define a discriminant function y_k for each class C_k with:

$$y_k(x) = w_k^T x + w_{k0}, x \in \mathbf{R}^d \tag{A.5}$$

Then, a new point x is assigned to a class C_k if $y_k(x) > y_j(x), \forall j, j \neq k$. The decision boundary separating C_j from C_k is given by $y_k(x) = y_j(x)$. Figure A.3 shows an example of a multi-class decision boundary.

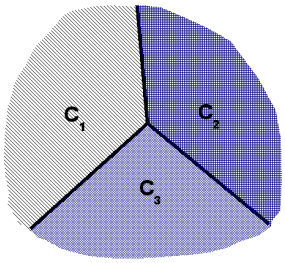


Figure A.3: Example decision boundary produced by a multi-class linear discriminant

Again, the decision boundaries can be modeled with a single-layer neural network as we can see in figure A.2 (right). The weights leading to output unit j are set to the values of w_j. The activation functions of the output units are the discrimination functions y_k.

So far, we have constructed our neural networks from known decision boundaries of classes. However, what we are really interested in, is to have the neural network learn the decision boundaries automatically from sample data.

For this purpose we consider the sum-of-squares error function:

$$E(w) = \frac{1}{2} \sum_i \sum_k \{y_k(x_i, w) - t_{ik}\}^2 \tag{A.6}$$

where x_i are sample vectors and t_{ik} their target values for the different classes. Since y_k is linear, E is a quadratic function in w and hence, the derivatives of E are linear. The solution of the minimization of this function can therefore be found quite easily by setting the derivatives to 0. We even can find a closed form for the solution.

Consequently, for arbitrary linear decision boundaries in $\mathbf{R}^d$ we can find weights w such that a neural network with the activation functions y_k for the output units, can classify all vectors correctly.

When considering different error functions or a non-linear function as activation function, such an explicit solution is not possible any more. However, if the activation function is differentiable, we can still calculate the derivatives of the error function E. These derivatives can be used to perform a gradient descent to find a suitable solution.

We may adopt the following procedure. We begin with an initial guess of the weights w. Subsequently, we move a small distance in w-space in the direction where E decreases the most rapidly. By iterating this process, we create a sequence of weights w^r whose components may be calculated with the following formula:

$$w_{kj}^{r+1} = w_{kj}^r - \eta \frac{\partial E}{\partial w_{kj}}\bigg|_{w^r} \tag{A.7}$$

where η is a small positive number called learning rate. Under suitable conditions the values of w will converge to a point where E is minimal. The choice of η is rather important, since low learning rates lead to a slow convergence, whereas high learning rates may lead to oscillation and divergence.

A.4.2 The Perceptron

A perceptron is a single-layer neural network with one output unit. To improve the performance the perceptron has a layer of fixed processing elements to transform the raw data. These processing elements can be considered as the basis function of a generalized linear discriminant.

Consequently, the output of the perceptron is given by

$$y = g\Big(\sum_j w_j \phi_j(x)\Big) = g(w^T \phi) \tag{A.8}$$

where ϕ is the vector of activations. The activation function of the output is defined by a threshold function of the form:

$$g(a) = \begin{cases} -1 & \text{if } a < 0 \\ +1 & \text{if } a \geq 0 \end{cases} \tag{A.9}$$

For the perceptron often a continuous, piecewise-linear error function called *perceptron criterion* is considered. It is a measure for the number of misclassifications the perceptron has performed. Suppose that with each activation vector ϕ^n we obtain an associated target value t^n. With $t^n = -1$ if ϕ^n belongs to class C_1 and $t_n = +1$ if phi_n belongs to class C_2. Then the perceptron criterion E^{perc} can be written as:

$$E^{perc}(w) = -\sum_{\phi^n \in M} w^T(\phi^n t^n) \tag{A.10}$$

with M the set of vectors ϕ^n which are misclassified by the current weight vector w.

Now, if we apply the pattern-by-pattern gradient descent rule from equation A.7, we obtain the following learning rule:

$$w_j^{r+1} = w_j^r + \eta \phi_j^n t^n \tag{A.11}$$

Hence, we have a very simple learning algorithm, that can be summarized as follows. Cycle through all input patterns in the training set with the current weights. If a pattern is misclassified simply add/subtract the pattern vector multiplied by η to the weights.

There is an interesting result which states that, for any data set which is linearly separable, the learning rule is guaranteed to find a solution in a finite number of steps. This property is known as the *perceptron convergence theorem*.

A.4.3 Multi-Layer Networks

As we have seen, single-layer networks have the limitation, that classes have to be linearly separable. Any problem requiring non-linear decision boundaries can not be classified by these networks. This is for example true for the XOR problem A.4.

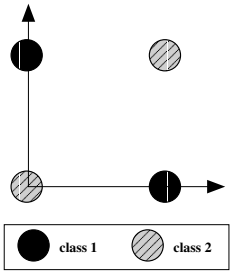


Figure A.4: The classes in the XOR problem are not linearly separable and thus a single-layer network cannot classify the elements correctly

Now, we want to have a look at the capabilities of multi-layer networks. We concentrate on networks consisting of successive layers of adaptive weights. These networks are easier to analyze theoretically than more general topologies. Moreover they can be implemented more efficiently in software or hardware. This kind of network can be viewed as a transformation of the input data by successive single-layer networks. Meaning that the output of one single-layer network is processed by the succeeding single-layer network. Therefore these networks are also called feed-forward networks. Figure [REFERENCE], for example, shows a two-layer network, which can solve the XOR problem.

Figure

It can be shown that even neural networks with two-layer of adaptive weights are capable of approximating an arbitrary continuous mapping from one finite-dimensional space to another, provided the number of hidden units is sufficiently large. For the classification problem, we can obtain an important corollary of this property. Two-layer networks with sigmoidal activation functions can approximate any decision boundary to arbitrary accuracy (see [Krei1991]). Thus, such networks also provide universal non-linear discrimination functions.

Next to this theoretical result, feed-forward networks are used because there exists a powerful and computational efficient method for finding the derivatives of an error function with respect to the weights. Provided that, the activation function are differentiable. This method can be used to train the neural network and is called error back-propagation.

Error-Backpropagation

For the error-backpropagation, we consider each training pattern separately. The error function E is defined as:

$$E = \sum_n E^n \tag{A.12}$$

where E^n is the error function for the pattern with index n. We suppose that E^n is differentiable for all n.

The unit input is in the following denoted by a_j and computed as we already have seen above:

$$a_j = \sum_i w_{ji} z_i \tag{A.13}$$

with z_i the activations of the predecessing units, which are calculated from their own unit inputs as follows:

$$z_j = g(a_j) \tag{A.14}$$

where g may be a non-linear activation function.

For each pattern, we now suppose that we have supplied the corresponding input vector to the neural network and calculated the activations of all hidden and output units by successive application of equation A.13 and A.14. This process is called forward propagation, since the activations are calculated from one layer to the next.

Our goal is to find a procedure for evaluating the derivatives of E^n with respect to the weights. Since E^n depends on the weight w_{ji} only via the unit input a_j, we can apply the chain rule for partial derivatives:

$$\frac{\partial E^n}{\partial w_{ji}} = \frac{\partial E^n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} \tag{A.15}$$

Using A.13, we can write:

$$\frac{\partial a_j}{\partial w_{ji}} = z_i \tag{A.16}$$

Thus, in order to evaluate the derivative of E^n we only have to calculate $\frac{\partial E^n}{\partial a_j}$ for each hidden and output unit in the network.

For the output units the evaluation can be done easily using equation A.14:

$$\frac{\partial E^n}{\partial a_k} = g'(a_k)\frac{\partial E^n}{\partial y_k} \tag{A.17}$$

The values on the right hand side of the equation are known from the definition of the activation and error function as well as from the forward propagation phase.

For the hidden units we again make use of the chain rule for partial derivatives:

$$\frac{\partial E^n}{\partial a_j} = \sum_k \frac{\partial E^n}{\partial a_k}\frac{\partial a_k}{\partial a_j} \tag{A.18}$$

where the sum runs over all units with index k to which unit j sends a connection.

By making use of the equations A.13 and A.14 we obtain the following *back-propagation* formula:

$$\delta_j = \frac{\partial E^n}{\partial a_j} = g'(a_j)\sum_k w_{kj}\frac{\partial E^n}{\partial a_k} \tag{A.19}$$

Consequently, now we can evaluate the derivatives of E^n with respect to the weights with the following algorithm:

1. Apply an input pattern to the network and forward propagate the activations through all layers
2. Evaluate the δ_k for all output units

3. Back-propagate the δ's using the backpropagation formula
4. Use the resulting δ's to calculate the derivatives of E^n and E

To perform a learning step, what still needs to be done is alter the weights with a gradient descent technique. For example:

$$\Delta w_{ji} = -\eta \frac{\partial E^n}{\partial w_{ji}} \tag{A.20}$$

with η the learning rate.

There exist a number of other learning techniques for neural networks. Quickprop and Rprop, for example, try to improve convergence speed for the learning algorithm. It is not intended to cover these techniques in this work. For a detailed description see [SNNSRef].

A.5 Generalization Of Neural Networks

Neural networks have the ability to generalize. This means that a trained neural network can classify data that has never been seen before. In real world applications developers normally have only a small part of all possible patterns. To reach best generalization, the data set is generally split into three parts:

- Training set
- Cross evaluation set
- Test set

the training set is used to train the neural network. Backpropagation or other learning techniques is used to minimize the error during training. The cross evaluation set is used to determine the performance of the neural network on patterns, which are unknown to the network. This is a measure of whether the network generalizes well. Finally, the test set is used for checking the over all performance of the neural network.

Usually during training the performance of the network is continuously tested on the cross evaluation set. If the error on the cross evaluation set reaches a minimum the training is stopped. The error on the training set may still decrease if further training is applied. However, this is the point where the neural network is considered to generalize best. Remember, our goal hasn't been to obtain the best possible results on the training set, but to obtain a neural network which performs well on all possible data. If we would continue to train the neural network, we would see an effect called over-fitting, meaning that the general performance of the network decreases.

Figure A.5 shows an example of a typical error development during neural network training.

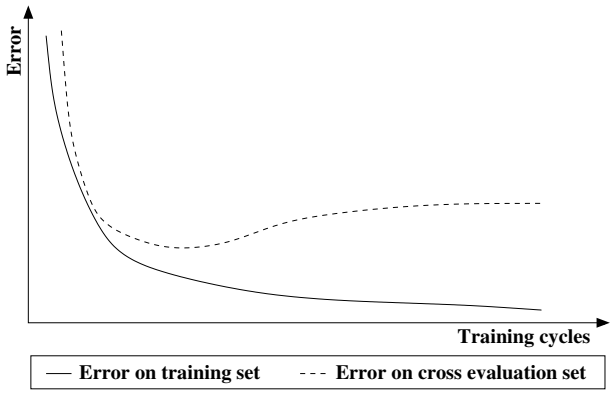


Figure A.5: Development of the error during neural network training

A crucial point when dealing with neural network is the availability of training data. The more units the neural network contains, the more training data is needed to achieve good performance. Next to the quantity of training data, it is important that there are enough training patterns for each class we want to classify.

Appendix B

Pattern-Based Face Detection

This chapter describes the face detection technique developed by Viola and Jones [VJ2001], which is used in our system to find a skin color region.

The technique classifies images based on the value of simple features. Features have the advantage that the can act to encode ad-hoc domain knowledge, which might be difficult to learn from the image data itself. Specifically, four kinds of features are used in this algorithm: *horizontal and vertical two-rectangle features, a vertical three-rectangle feature and a four-rectangle feature* (see figure B.1).

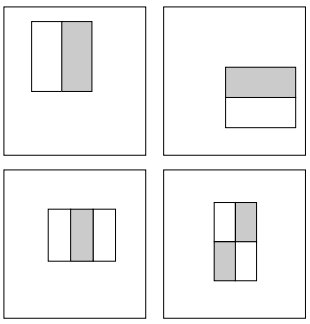


Figure B.1: Example rectangle features shown relative to a detection window

The value of a feature is computed by summing up all pixels in the gray areas and calculating the difference to the pixels in the white areas.

In order to compute these features efficiently, a so-called integral image is used. For an image $i(x, y)$ the integral image ii is defined by:

$$ii(x, y) = \sum_{x<x_1, y_1<y} i(x_1, y_1) \tag{B.1}$$

The integral image itself can be computed with a single pass over the image by the following pair of recurrences:

$$s(x, y) = s(x, y - 1) + i(x, y) \tag{B.2}$$
$$ii(x, y) = ii(x - 1, y) + s(x, y) \tag{B.3}$$

where $s(x, y)$ denotes the pixel sum of column x till the row y is reached.

Consequently the sum of the pixels in a rectangle can be calculated by taking the integral image values at its corners. Figure B.2 shows an example.

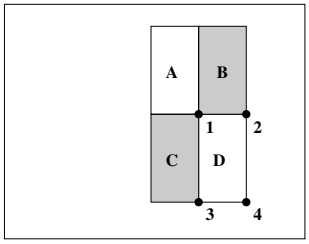


Figure B.2: The sum of the pixels in rectangle D can be computed by adding the integral image values at point 1 and 4 and subtracting the values at point 2 and 3

B.1 Learning Classification Functions

Given the feature set and a training set with positive and negative images, in their system AdaBoost is used to both select the appropriate features and to actually train the classifier. AdaBoost helps to improve simple learning algorithms called weak learner, which might only have recognition rates of 51%.

AdaBoost provides rather strong formal guarantees. It has been proved that the training error of the resulting classifier approaches to zero exponentially. Moreover AdaBoost provides good generalization performance. AdaBoost

helps finding a small set of good classification functions which have significant variety nevertheless.

In their face detection algorithm these classification functions or weak learners correspond to finding a rectangle features and a threshold, which can perform the classification task with low error. The weak learners h_j therefore can be written in the following manner:

$$h_j(x) = \begin{cases} 1 & \pm f_j(x) < \sigma_j \\ 0 & otherwise \end{cases} \tag{B.4}$$

with x a detection window, f_j a rectangle feature, and σ_j a threshold value.

The boosting algorithm based on these weak learners works as follows (see [VJ2001]):

- Given example images $(x_1, y_1), \ldots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives
- For $t = 1, \ldots, T$:
  1. Normalize the weights,
  $$w_{t,i} = \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}} \tag{B.5}$$
  so that w_t is a probability distribution.
  2. For each feature j train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to $w_t, e_j = \sum_i w_i |h_j(x_i) - y_i|$.
  3. Choose the classifier h_t with the lowest error e_t.
  4. Update the weights:
  $$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i} \tag{B.6}$$
  where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{e_t}{1-e_t}$.

- The final strong classifier is:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & otherwise \end{cases} \tag{B.7}$$

  where $\alpha_t = \log \frac{1}{\beta_t}$.

Figure B.3 shows the first and the second features selected by the boosting algorithm. The two-rectangle feature represents the fact, that eyes are darker then the upper cheeks. The three-rectangle feature compares the intensities in the eye regions to the intensity across the bridge of the nose.

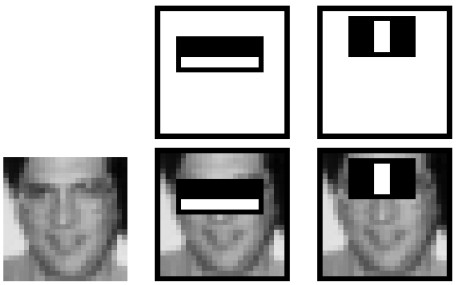


Figure B.3: First and second features selected by AdaBoost (images taken from [VJ2001])

In order to improve detection performance, Viola and Jones implemented a so-called *attentional cascade*. The image windows pass through some kind of degenerated decision tree with a classifier at each node. The cascade helps filtering out a great number of images with the computation of a single or several simple classifiers. Only the image windows which pass all the cascade nodes are considered to be a face. For more details and experimental results please refer to [VJ2001].

Appendix C

Discrete Kalman Filter

The Kalman filter has been subject of extensive research. It has been used for all kinds of applications where measurements of a continuous process was noisy and results had to be filtered.

The discrete Kalman filter addresses the general problem of estimating a state $x \in \mathbf{R}^n$ of a discrete-time controlled process. The underlying stochastic equation has the following form:

$$x_k = Ax_{k-1} + w_{k-1} \tag{C.1}$$

where x_k is the state of the process and at time k and w_k the process noise. Apart from the process noise, the matrix A defines how a state x_k is calculated from its previous state.

A measurement $z \in \mathbf{R}^m$ is derived from a state k with:

$$z_k = Hx_k + v_k \tag{C.2}$$

where v_k is the measurement noise. The matrix H computes the measurement vector from a state x_k.

We assume the random variables w_k, v_k to be distributed normally with covariance matrices Q and R:

$$p(w) \sim N(0, Q) \tag{C.3}$$
$$p(v) \sim N(0, R) \tag{C.4}$$

The measurement noise covariance R and the process noise covariance Q might change with each time step of measurement. However we assume the matrices to be constant.

C.1 Basic Kalman Filter Equations

We define x_k^{prior} to be the *a priori* state estimate given the knowledge of the process states prior to step k. Moreover, we define x_k^{poster} to be the *a posteriori* state estimate given the knowledge of the process states prior to step k and the measurement vector z_k. The corresponding errors e_k^{prior} and e_k^{poster} are consequently:

$$e_k^{prior} = x_k - x_k^{prior} \tag{C.5}$$
$$e_k^{poster} = x_k - x_k^{poster} \tag{C.6}$$

The *a priori* and *a posteriori* covariance matrices follow from e_k^{prior} and e_k^{poster}:

$$P_k^{prior} = E[e_k^{prior} e_k^{prior^T}] \tag{C.7}$$
$$P_k^{poster} = E[e_k^{poster} e_k^{poster^T}] \tag{C.8}$$

Now the Kalman filter equation can be derived from the above equations. In this equation, the *a posteriori* state x_k^{poster} is computed from its corresponding *a priori* state x_k^{prior}, as well as the weighted difference of the measurement vector z_k and a measurement prediction Hx_k^{prior}. We obtain the following formula:

$$x_k^{poster} = x_k^{prior} + K(z_k - Hx_k^{prior}) \tag{C.9}$$

The justification for this equation comes from the probabilistic origin of the filter. For further details see [Jaco1993]. The difference $(z_k - Hx_k^{prior})$ is called measurement innovation and denotes the discrepancy between the predicted measurement and the actual measurement value. A difference value of 0 means that the actual measurement is conform with the predicted one. The *a priori* state estimate therefore has not to be updated.

The $n \times m$ matrix K in equation C.9 is chosen to minimize the *a posteriori* error covariance P_k^{poster}. The minimization can be accomplished by substituting the Kalman filter equation into the definition of e_k^{poster} and substituting this into the definition of P_k^{poster}. Subsequently, the derivative of the trace of P_k^{poster} with respect to K can be calculated and set to 0. We obtain the resulting matrix K_k for step k of the process:

$$K_k = P_k^{prior} H^T (H P_k^{prior} H^T + R)^{-1} \tag{C.10}$$

Obviously, the more the measurement error covariance approaches 0 the larger is the weight K_k gives to the measurement innovation. In this case, the *a priori* state estimate has to be updated to conform to the measurement. We also might say that we trust the actual measurement more than the *a priori* estimation. On the other hand, as the *a priori* estimate error covariance P_k^{prior} decreases, the less weight is given to the measurement innovation. In that case, the *a priori* state estimation is rather good and can be trusted. Therefore the *a priori* estimation has only to be slightly changed.

C.2 The Predictor-Corrector Process

As might be imagined from the above section the Kalman filter estimates the process by using a form of feedback control also referred to as predictor-corrector approach. The filter predicts the state of the process at some step k and then obtains feedback in form of a possibly noisy measurement.

As such, we define two groups of equations:

- time update equations (prediction step)
- measurement update equations (correction step)

The specific equations for the time updates are:

$$x_k^{prior} = Ax_{k-1}^{poster} \tag{C.11}$$
$$P_k^{prior} = AP_{k-1}^{poster}A^T + Q \tag{C.12}$$

Notice again, that these equations project the state and the covariance estimates forward from time step $k-1$ to k.

For the measurement updates, we obtain:

$$K_k = P_k^{prior}H^T(HP_k^{prior}H^T + R)^{-1} \tag{C.13}$$
$$x_k^{poster} = x_k^{prior} + K_k(z_k - Hx_k^{prior}) \tag{C.14}$$
$$P_k^{poster} = (I - K_kH)P_k^{prior} \tag{C.15}$$

So, the first task during the measurement update is computing the matrix K_k which minimizes the *a posteriori* error covariance (see section C.1). Subsequently, the process is actually measured and the *a posteriori* state estimate is calculated with the help of this measurement. Finally, we need to compute the *a posteriori* covariance estimate for the next time update step.

Now, we can continue with another time update step. This two stage process is repeated from each time step $k-1$ to k (see figure C.1).

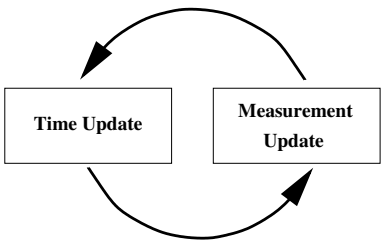


Figure C.1: Predictor-Corrector cycle of the Kalman filter

Bibliography

[Adam2000] B. Adams et al., *Humanoid robots: a new kind of tool*, IEEE Intelligent Systems, pages 25-31, 2000

[Agah2001] A. Agah, *Human interactions with intelligent systems: research taxonomy*, Computers and Electrical Engineering,pages 71-107, 2001

[AHP1993] A. Azarbayejani, B. Horowitz and A. Pentland, *Recursive estimation of structure and motion using the relative orientation constraint*, Proceedings of the Computer Vision and Patteern Recognition Conference, pages 70-75, 1993

[Bish2000] Christopher M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, 2000

[DTB2002] Darrell, T., Tollmar, K., Bentley, F., Checka, N., Morency, L.-P., Rahimi A., and Alice Oh, *Face-responsive Interfaces: from Direct Manipulation to Perceptive Presence*, International Conference of Ubiquitous Computing, 2002

[Fitz2001] Paul Fitzpatrick, *Head pose estimation without manual initialization*, AI Lab, MIT, Cambridge, USA, 2001

[HB1995] D. Heeger and J. Bergen, *Pyramid-based texture analysis/synthesis*, Proceedings of SIGGRAPH 1995, pages 229-238

[HH2002] Y. Huang, T. S. Huang, *Facial Tracking with Head Pose Estimation in Stereo Vision*, IEEE Int. Conference on Image Processing, Rochester, New York, US, September 22-25, 2002

[HRD1999] Michael Harville, Ali Rahimi, Trevor Darrell, Gaile G. Gordon, John Woodfill, *3D Pose Tracking with Linear Depth and Brightness Constraints*, ICCV 1999, pp. 206-213

[HS1980] B.K.P. Horn and B.G. Schunck, *Determining optical flow*, AI Memo 572, Massachusetts Institute of Technology, 1980

[Jaco1993] O. Jacobs, *Introduction to Control Theory, 2nd Edition*, Oxford University Press, 1993

[Jähn1997] B. Jähne, *Digitale Bildverarbeitung*, Springer-Verlag, Berlin-Heidelberg, 4. Auflage, 1997

[KBS2000] Volker Krüger, Sven Bruns, Gerald Sommer, *Efficient Head Pose Estimation with Gabor Wavelet Networks*, Proc. British Machine Vision Conference, Bristol, UK, Sept. 12-14, 2000.

[Koku2000] , A.B. Koku et al., *Towards socially acceptable robots*, Proceedings of 2000 IEEE International Conference on Systems, Man and Cybernetics, pages 894-899, 2000

[Kono1997] K. Konolige, *Small Vision System: Hardware and Implementation*, IEEE Conference on Computer Eighth International Symposium on Robotics Research, Hayama, Japan, 1997

[Krei1991] V. Y. Kreinovich, *Arbitrary nonlinearity is sufficient to represent all functions by neural networks: a theorem*, Neural Networks 4 (3), pages 381-383, 1991

[Mats1999] , Y. Matsusaka et al., *Multi-person conversation via multi-modal interface - A robot who communicates with multi-user*, Proc. Eurospeech 99,pages 1723-1726, 1999

[Matt2002] Matteo Matteucci, *ELeaRNT: Evolutionary Learning of Rich Neural Network Topologies*, Center for Automated Learning and Discovery, Technical Report, CMU-CALD-02-103, 2002

[MA2000] Yoshio Matsumoto, Tsukasa Ogasawara, Alexander Zelinsky, *Behavior Recognition Based on Head Pose and Gaze Direction Measurement*, Proceedings of 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'2000), pp.2127-2132, 2000

[MC1993] B.S. Manjunath and R. Chellappa, *A unified approach to boundary perception: edges, textures and illusory contours*, IEEE Trans. Neural Networks, 1993

[MRC2002] Louis-Philippe Morency, Ali Rahimi, Neal Checka, and Trevor Darrell, *Fast stereobased head tracking for interactive environment*, In

Proceedings of the Int. Conference on Automatic Face and Gesture Recognition, 2002.

[MZ2000] Yoshio Matsumoto, Alexander Zelinsky, *An Algorithm for Real-time Stereo Vision Implementation of Head Pose and Gaze Direction Measurement*, Proceedings of IEEE Fourth International Conference on Face and Gesture Recognition (FG'2000), pp.499-505, 2000

[NS2003] K. Nickel and R. Stiefelhagen, *Pointing Gesture Recognition based on 3D tracking of Face, Hands and Head Orientation*, Fifth International Conference on Multimodal Interfaces, Vancouver, Canada, Nov. 5-7, 2003

[Nick2003] Kai Nickel and Rainer Stiefelhagen, *Detection and Tracking of 3D-Pointing Gestures for Human-Robot-Interaction*, Proceedings of the Third IEEE International Conference on Humanoid Robots - Humanoids 2003, Karlsruhe, Germany, October 1-3, 2003

[NSS2004] Kai Nickel et al., *Tracking Head and Hands for Pointing Gesture Recognition in a Human-Robot Interaction Scenario*, submitted to International Conference on Automatic Face and Gesture Recognition (FG2004), Seoul, Korea, 2004

[Perz2001] D. Perzanowski et al., *Building a multimodal human-robot interface*, IEEE Intelligent Systems, pages 16-21,2001

[SA2000] Sangho Park, J.K. Aggarwal, *Head Segmentation and Head Orientation in 3D Space for Pose Estimation of Multiple People*, 4th IEEE Southwest Symposium on Image Analysis and Interpretation, p. 192, Austin, April 02-04, 2000

[SB2002] S. Srinivasan and K. L. Boyer, *Head Pose Estimation Using View Based Eigenspaces, Intl. Conf. on Pattern Recognition*, Quebec, 2002

[SM2002] Kenneth O. Stanley and Risto Miikkulainen, *Efficient Evolution of Neural Network Topologies*, Proceedings of the 2002 Congress on Evolutionary Computation (CEC '02). Piscataway, NJ: IEEE, 2002

[SNNSRef] *Stuttgart Neural Network Simulator User Manual*, Version 4.2

[SS1996] C. Stergiou and D. Siganos, *Neural Networks*, SURPRISE 96 Journal, Department of Computing, Imperial College of Science Technology and Medicine, London, 1996

[Stie2002] Rainer Stiefelhagen, *Tracking and Modeling Focus of Attention in Meetings*, Dissertation, Universität Karlsruhe, Fakultät für Informatik, 2002

[SYW2001] R. Stiefelhagen, J. Yang, A. Waibel, *Tracking Focus of Attention for Human-Robot Communication*, IEEE-RAS International Conference on Humanoid Robots - Humanoids 2001, November 22-24, 2001, Tokyo, Japan

[VJ2001] P. Viola and M. Jones *Robust real-time object detection*, Technical Report 2001/01, Compaq CRL, February 2001.

[WB2003] Greg Welch and Gary Bishop, *An Introduction to the Kalman Filter*, Technical Report TR 95-041 University of North Carolina at Chapel Hill, 2003

[YLW1997] J. Yang, W. Lu and A. Waibel, *Skin-Color Modeling and Adaption*, Technical Report, School of Computer Science CMU-CS-97-146, CMU, USA, 1997

[YZ2001] R. Yang and Z. Zhang, *Model-based Head Pose Tracking With Stereovision*, In Proc. Fifth IEEE International Conference on Automatic Face and Gesture Recognition (FG2002), pages 255-260, Washington, DC, May 20-21, 2002

[Zhan2000] Z. Zhang, *A flexible new technique for camera calibration*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(11):1330-1334, 2000