

# Detecting Head Orientation in Low Resolution Surveillance Video

Technical Report CVMT-06-02 ISSN 1601-3646

Thomas Baltzer Moeslund,  
Bjarne Kondrup Mortensen & Dennis Mølholm Hansen

2nd November 2006



# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>Figure-ground Segmentation</b>	<b>7</b>
2.1	Background Subtraction . . . . .	7
2.2	Codebook Method . . . . .	8
2.3	Summery . . . . .	11
<b>3</b>	<b>Tracking</b>	<b>12</b>
3.1	Blob Merging . . . . .	13
3.2	Object Classification . . . . .	15
3.3	Tracking over Time . . . . .	16
3.4	Summery . . . . .	16
<b>4</b>	<b>Head Extraction</b>	<b>18</b>
4.1	Tophat Transform . . . . .	18
4.2	Gradient . . . . .	19
4.3	Face Detect in OpenCV . . . . .	21
4.4	Template Matching . . . . .	22
4.5	Summery . . . . .	31
<b>5</b>	<b>Head Orientation</b>	<b>32</b>
5.1	Symmetry/Correlation . . . . .	32
5.2	Feature Tracking . . . . .	36
5.3	Moment . . . . .	38
5.4	Summery . . . . .	47
<b>6</b>	<b>Conclusion</b>	<b>48</b>

<b>Bibliography</b>	<b>50</b>
<b>A Code Review</b>	<b>52</b>
<b>B Codebook Parameters</b>	<b>58</b>
<b>C Tracking Parameters</b>	<b>60</b>

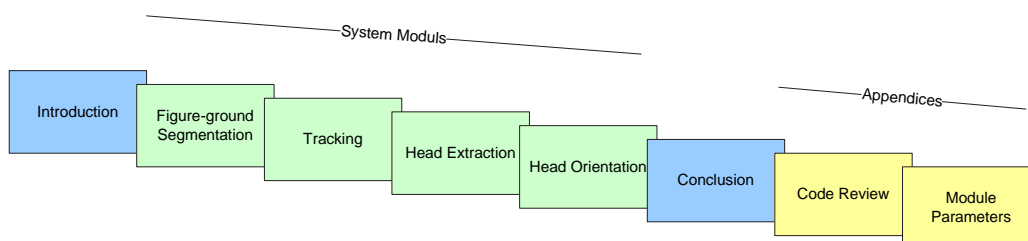
# CHAPTER 1

## Introduction

The work described in the technical report is part of the HERMES project [HERMES, 2006]. The main objective in the HERMES project is to develop a cognitive artificial system based in a framework model which allows both recognition and description of a particular set of human behaviors arising from real-world events.

The purpose of this work is to detect where the attention of humans in video-sequences are. The meaning of attention corresponds to where the person is looking.

The developed system is divided into four main modules; Figure-ground segmentation, tracking, head extraction, head orientation. A chapter is devoted to each module as seen in Figure 1.1, which illustrates the structure of the



**Figure 1.1:** Overview of the structure in the report.

report.

For further information about this work, please contact:

Thomas Baltzer Moeslund  
Aalborg University  
Faculty of Engineering, Science and Medicine  
Laboratory of Computer Vision and Media Technology (CVMT)  
Niels Jernes Vej 14 (3-109)  
DK-9220 Aalborg East, Denmark  
Phone: +45 96 35 87 87  
Fax: +45 98 15 24 44  
<http://www.vision.auc.dk/~tbm>  
[tbm@cvmt.dk](mailto:tbm@cvmt.dk)

## CHAPTER 2

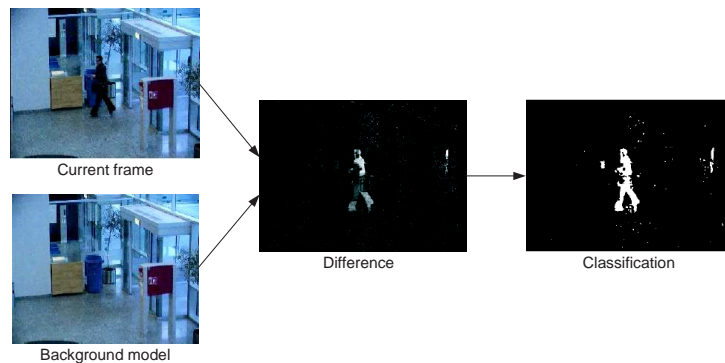
# Figure-ground Segmentation

The purpose of the figure-ground segmentation module is to handle the input from the camera, and segment the persons from the background in the video. Having removed the background, the focus of the later modules can be on tracking and finding the head orientation of the persons.

## 2.1 Background Subtraction

Different methods exist for figure-ground segmentation; i.e. background subtraction, temporal differencing and optical flow. In this work the background subtraction method is utilized. The concept of background subtraction is outlined in Figure 2.1. The frame is pixel-wise compared to a background model. If the difference is large enough the pixel is deemed foreground.

With a static background model, this method is extremely sensitive towards changes in the scene such as illumination. Therefore, it is very common to update the background model. The continuous maintenance of the background model is the backbone of a good background subtraction method. There exist many different methods for creating and maintaining the background model. Among the most popular are the Mixture of Gaussians and Code-



**Figure 2.1:** The concept of background subtraction. The pixel level difference between the current frame and the background model is found. A threshold determines whether or not the pixel belongs to the foreground.

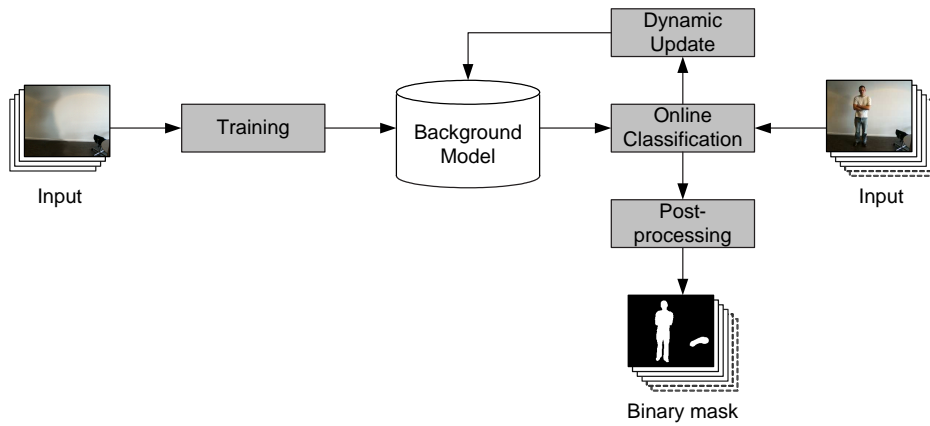
book method. In this work the codebook method is applied. This is chosen based on a perturbation test performed in [Chalidabhongse et al., 2003].

## 2.2 Codebook Method

The figure-ground segmentation is outlined in Figure 2.2. The background model is obtained through training with an empty scene. When the background model is trained, it can be used to classify frames that might contain persons. After classification the result is postprocessed to remove small errors.

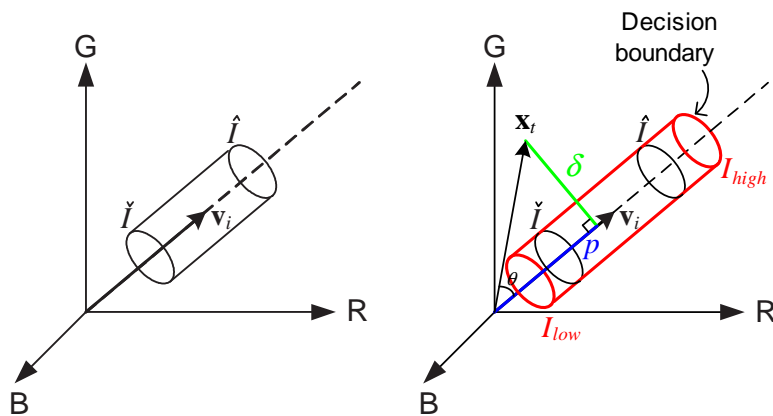
In the codebook method the background model is represented by a codebook for each pixel in the frame. A codebook contains a number of codewords, which describes the part of RGB-space that are background. A codeword is illustrated in Figure 2.3(a). Each codeword spans a volume in the RGB-space. The volume is shaped as a cylinder. A sampled pixel within the volume of a codeword is classified as background. When a pixel is in this volume, it is said to activate the codeword.

To determine if a sampled pixel  $\mathbf{x}_t$  lies within a codeword, the color distortion  $\delta$  and brightness  $\rho$  are calculated. These are illustrated in Figure 2.3(b). The



**Figure 2.2:** The outline of the conceptual design of the figure-ground segmentation module.

color distortion must be below a predefined threshold,  $\varepsilon$ , which corresponds to the radius of the codeword. The brightness must be between  $I_{low}$  and  $I_{high}$ , which are calculated from the minimum and maximum intensities of the codeword.



(a) Illustration of a codeword. The codeword spans a volume in the RGB-space.

(b) A codeword and a pixel  $\mathbf{x}_t$ . If  $\mathbf{x}_t$  is within the volume (the red cylinder) it activates the codeword.

**Figure 2.3:** A codeword in RGB-space.

### 2.2.1 Training

In the following the construction of a single codebook is described. The model is trained using a training sequence of  $N_{train}$  frames. The pixel sampled at time  $t$  in the training period,  $\mathbf{x}_t$ , is compared to the current codebook,  $\mathcal{C}$ , to determine which codeword  $c_m$  (if any) it matches.  $m$  is the matching index of the codeword in the codebook  $\mathcal{C}$ . The algorithm for constructing a codebook is shown in Listing 2.1.

---

```

1  $L = 0$  ,  $\mathcal{C} = \emptyset$ 
   For  $t = 1$  to  $N_{train}$  do
3    $\mathbf{x}_t = (x_R, x_G, x_B)^T$  ,  $I = \sqrt{x_R^2 + x_G^2 + x_B^2}$ 
     Seek matching codeword  $c_m$  in  $\mathcal{C}$  based on the two conditions:
5      $\text{colordist}(\mathbf{x}_t, \mathbf{v}_m) \leq \varepsilon_1$ 
      $\text{brighness}(I, (\check{I}_m, \hat{I}_m)) = \text{true}$ 
7   If  $\mathcal{C} = \emptyset$  or there is no match, then create a new codeword with index  $L$ :
      $L = L + 1$ 
9   Create a new codeword  $c_L = (\mathbf{x}_t, I, I, 1, t - 1, t, t)$  and add to  $\mathcal{C}$ 
     Otherwise, update the matching codeword  $c_m$ :
11     $\mathbf{v}_m = \left( \frac{f_m \cdot v_{R,m} + x_R}{f_m + 1} , \frac{f_m \cdot v_{G,m} + x_G}{f_m + 1} , \frac{f_m \cdot v_{B,m} + x_B}{f_m + 1} \right)^T$ 
      $c_m = \left( \mathbf{v}_m , \min(I, \check{I}_m) , \max(I, \hat{I}_m) , f_m + 1 , \max(t - q_m, \lambda_m) , p_m , t \right)$ 
13 End for

```

---

**Listing 2.1:** Algorithm for constructing a codebook

### 2.2.2 Online Classification

After the training is completed the pixels in the frame are classified as either foreground or background. To classify the pixel sampled at time  $t$ ,  $\mathbf{x}_t$ , the algorithm in Listing 2.2 is used.

See [Andersen et al., 2006] for more details about the figure-ground segmentation module. Here different algorithms for the dynamic updating is shown. The parameters used to control the codebook are shown in Appendix B.

---

```

1  $\mathbf{x}_t = (x_R, x_G, x_B)^T$ ,  $I = \sqrt{x_R^2 + x_G^2 + x_B^2}$ 
  Seek matching codeword  $c_m$  in  $\mathcal{C}$  based on the two conditions:
3    $\text{colordist}(\mathbf{x}_t, \mathbf{v}_m) \leq \varepsilon_2$ 
    $\text{brightness}(I, (\hat{I}_m, \hat{I}_m)) = \text{true}$ 
5 If match found, then
   Update  $c_m$ 
7   Classify  $\mathbf{x}_t$  as background
  Otherwise
9   Classify  $\mathbf{x}_t$  as foreground

```

---

**Listing 2.2:** Algorithm for background subtraction

## 2.3 Summery

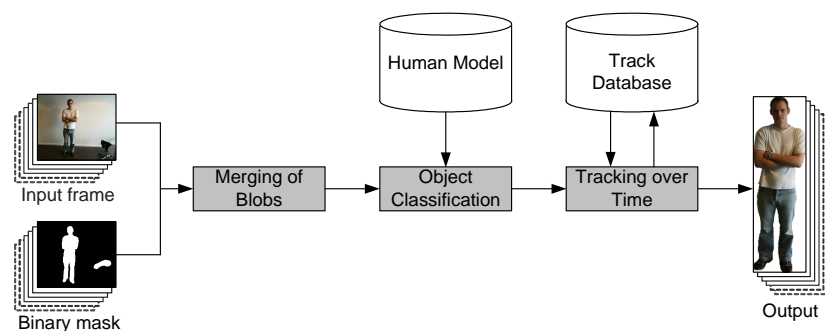
In this chapter an overview of the used figure-ground segmentation module has been given. First it was decided to use a background subtraction method. Then, the codebook method was chosen, and the concepts in the method was described. Finally, it was shown how a background model can be trained, and later be used for classification. For further details and test see [Andersen et al., 2006].

# CHAPTER 3

---

## Tracking

The tracking module serves several purposes. Its main objective is to track the persons from frame to frame. However, there might be other moving objects than humans in the scene. Therefore, the objects are classified as either humans or “noise”. Only the humans are tracked. Due to segmentation errors in the figure-ground segmentation, a person might be composed of several separated parts. These parts needs to be merged together to form a coherent person. The steps in the tracking module are illustrated in Figure 3.1.



**Figure 3.1:** The outline of the conceptual design of the tracking module.

The input to the tracking module is the original input frame and the corresponding binary mask produced by the figure-ground segmentation module.

The following sections describe the three different tasks performed by the tracking module.

## 3.1 Blob Merging

As seen in Figure 3.2, the extracted binary mask of one person can consist of many separated blobs. To merge these blobs into one object the following criteria must be fulfilled:

**Blob size** The size of the blobs must be larger than a predefined threshold set to 5 pixels. This is not scale invariant.

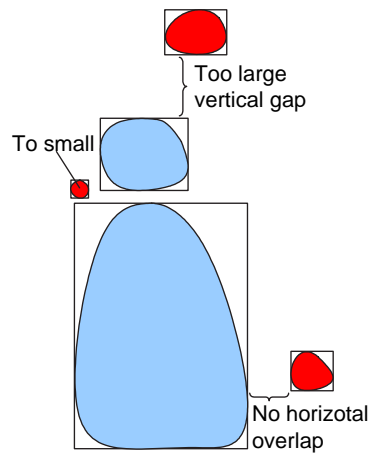
**Horizontal overlap** Two blobs are only merged if they overlap on the horizontal axis.

**Vertical distance** The vertical distance between the bounding boxes of the blobs must be lower than a predefined threshold set to 10 pixels.



**Figure 3.2:** Errors made in the figure-ground segmentation

The three criteria are illustrated in Figure 3.3. The two blue blobs are the only ones that are merged. The reds have either no overlap on the horizontal axis, a too large vertical gap or is too small.



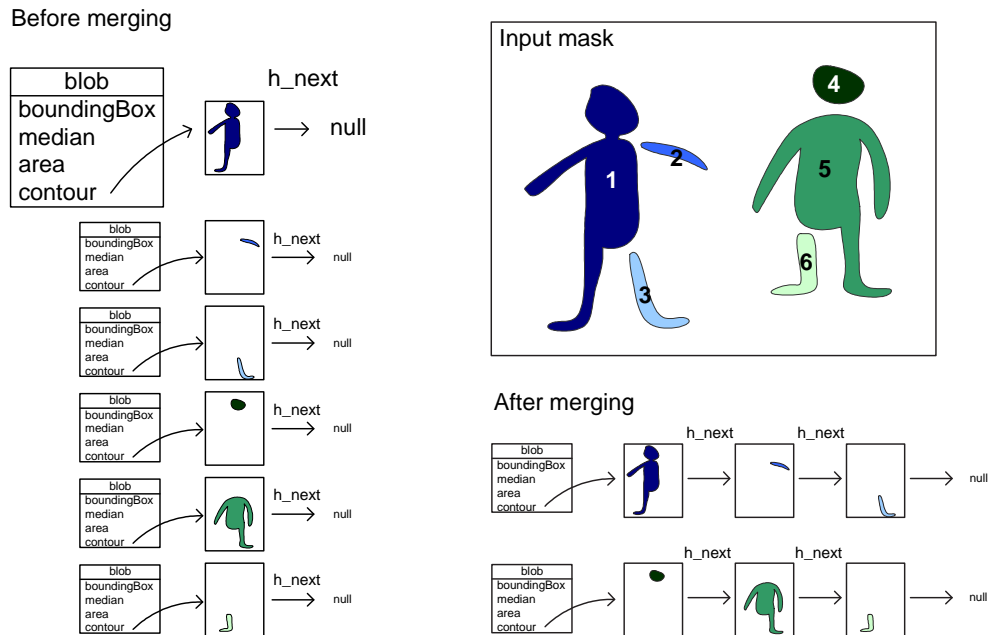
**Figure 3.3:** The different criteria used to determine if two blobs should be merged. Only the two blue blobs are merged.

Using the above defined criteria the blob merging is performed as described in Figure 3.4. The input mask contains two persons each comprised of three separated blobs. First the blobs are found in the mask using the `cvFindContours()` function in the OpenCV library. The result from this is a linked list of `CvContours`, which describes the boundary of a blob.

The first step is to create a blob object for each contour in the linked list. This is seen in the left of the figure, where the `contour` pointer in the blob object points to the contour. The `h_next` in the `CvCountour` is set to point to `null`. Normally `h_next` points to the next element in the linked list.

Next, all blobs are compared to see if the criteria are fulfilled. If this is the case, the two blobs are collapsed into one. One of the blobs are deleted and the other are updated to account for both blobs. This is done by letting the contour of the saved blob point to the contour of the deleted blob. The order which the blobs are compared have no influence on the final merging. Whenever two blobs are merged, the joined blob is compared to all other blobs once more.

After the merging is done, the blobs are stored as seen in the lower right part of the figure. For each person in the input mask a single blob object remains. The `contour` points to a linked list containing all the contours comprising the person.



**Figure 3.4:** The top right figure shows a fragmented input mask. The blobs are first stored as seen in the left figure. All `h_next` pointers in the `CvContour` structs are set to `null`. After the merging the blobs are stored as seen in the lower right figure. Blobs composing a single person are stored in one blob object. All the contours of the different blobs are stored in a linked list using the `h_next` pointer in the `CvContour` struct.

## 3.2 Object Classification

The objects found during the blob merging can be both persons or “noise”. Only the persons need to be tracked. A simple box-classifier is used to distinguish between persons and noise. Two sets of thresholds are used to classify the objects:

**Aspect Ratio** The aspect ratio between the width and height of the bounding box ( $AR = \frac{width}{height}$ ) must be between 0.2 and 0.8 for the object to be classified as human. These thresholds are applied under the assumption that the persons are walking upright.

**Area** The area of the object must be between 300 and 5000 pixels for the

object to be classified as human. The minimum and maximum areas correspond the person being between 5-15 meters from the camera.

### 3.3 Tracking over Time

When a person is tracked over time, a track object representing the person is created. The track object stores the history of the median point of the person in each frame. When a person is found in a frame his median point is compared to all the tracks. The closest track is updated to contain the newly found person. This is an acceptable method since it is assumed that no occlusion are present. This is illustrated in Figure 3.5



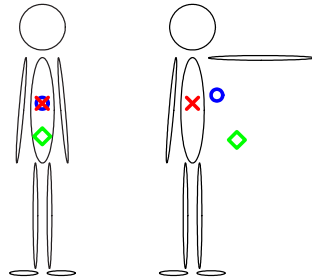
**Figure 3.5:** Tracking performed over two frames on two persons. The squares mark the median point at time  $t$  and the circle at time  $t+1$ .

The reason why the median points of the person are used in the tracking is, that they are more stable than e.i. the center of mass. Figure 3.6 shows this.

For more information about the tracking module please see [Andersen et al., 2006]. In Appendix C the parameters used in the tracking module are listed.

### 3.4 Summery

In this chapter the three tasks handled in the tracking module have been described. First, blobs in the binary output from the figure-ground segmentation module are found. Foreground objects may be split into numerous



**Figure 3.6:** The green square is the center of the bounding box. The blue circle is the center of mass. The red cross is the median point. The median point is the most stable when the person hold up his arm.

smaller blobs. These are therefore merged. Afterwards the merged blobs are classified as either persons or noise by using a box-classifier. Lastly, the persons are tracked over time using their median point. The tracking module works well as long as no occlusion is present. Test and further details can be found in [Andersen et al., 2006].

# CHAPTER 4

---

## Head Extraction

This chapter discusses several different methods for extracting the head location from a binary mask of a person. The first two methods use a horizontal projection of the binary mask to find the most narrow place which should correspond to the neck. The third method uses a build-in OpenCV function based on [Viola and Jones, 2004]. The last method uses an appearance-based approach, where a template of the head and shoulders is matched to the binary mask.

### 4.1 Tophat Transform

The idea is to produce a vertical histogram of the person's mask and locate the neck as a local minimum using the tophat transformation. However, this is not a feasible solution since the local minimum often is practically non-existing or covered in noise, as depicted in Figure 4.1 on page 20. It does not work.

## 4.2 Gradient

A couple of ideas are based on the gradient. The gradient is calculated from the person's mask in the following steps:

- Make a vertical histogram of the person.
- Smooth the histogram (necessary in order to calculate staple derivatives).
- Calculate the first order derivatives of the histogram (Second order derivatives do not make more sense).

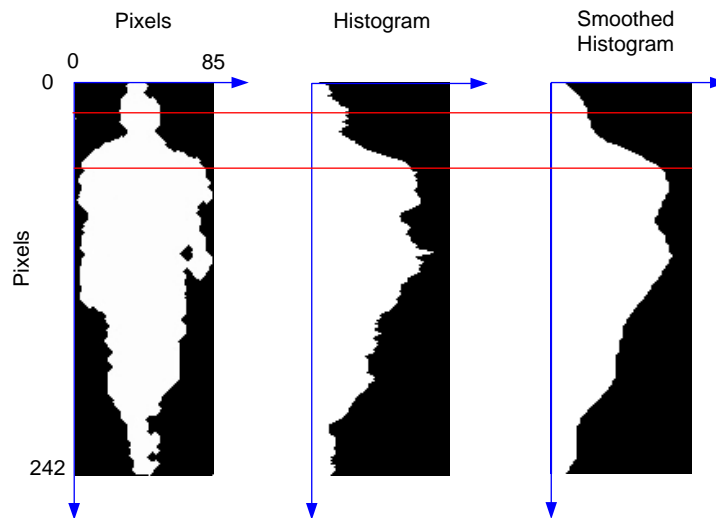
The result is a diagram of the derivatives of the histogram. To different solutions are then applied.

### 4.2.1 Maxfilter

The first solution is based on running a maxfilter on the histogram. The head and shoulders are then identified as the first and second maximum (top down). However, this solution fails because the two maximums are poorly separated, there is no minimum between the two, as shown in Figure 4.1 on the next page.

### 4.2.2 Logic

The second solution is based on logic and designed to accommodate the weak presence of the neck in the histogram, as the example in Figure 4.2 depicts. The approach is based on moving average, i.e. moving an interval over the image and continuously determining the average increase in the interval. Moving from the top down, the first goal is to identify the first increase above a certain threshold (the forehead). When the first goal is reached the interval length is reduced and the motion continues from the same offset. The next goal is to identify a zero or negative increase in the interval (face/chin).

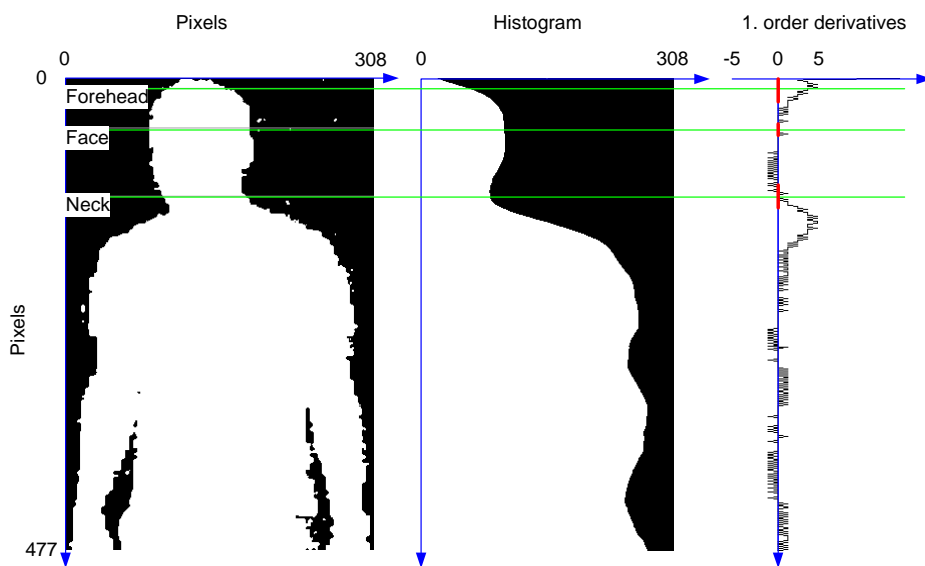


**Figure 4.1:** Example of a histogram of a segmented person. The red lines show the where the maxfilter should have located the two local maximums, but as seen histograms this is not a feasible solution. Though the neck appears in the first histogram as a weak local minimum, it is not visible on the smoothed histogram.

Again, the interval length is increased and a new threshold is set in order to local the the next increase above a certain threshold (neck/shoulder).

The method preforms well when the diagram contains a zero or negative average increase for an interval of at least 10 pixels at the location of the neck. However, this is not always the case, especially at low resolutions, e.g. as in Figure 4.1. A disadvantage of this solution is, that it involves many thresholds (increase and interval length) which need to be adjusted dynamically since the method is not natively scale invariant. Furthermore, the solution is sensitive to segmentation errors. In the example in Figure 4.2, the three interval lengths are set to 10, 7 and 12 pixels with a increase threshold of 0.3, 0.0 and 0.3, respectively.

This solution is not chosen for the final implementation.



**Figure 4.2:** Method that locates the neck as an average increase over a certain threshold in an interval (forehead) followed by a zero or negative increase in a second interval (face/chin) followed by another increase above a threshold in a third interval (neck/shoulders). Intervals are depicted as red lines, and the derived positions are depicted with green lines.

### 4.3 Face Detect in OpenCV

The OpenCV library offers a build-in face detector based on [Viola and Jones, 2004] (Haar-like features, AdaBoost, Cascade classifier). The face detector is very accurate on images of good quality and resolution. The face detector has been tested with a few videos used in this study, and it has been observed that the face detector recognizes faces down to  $15 \times 15$  pixels, as shown in the leftmost image of Figure 4.3 on the following page. However, at that resolution false negatives are frequent, as depicted in the following images. False negatives especially occur when the person is a little blurred e.g. when the person is moving. Furthermore, false positives also frequently occur in the test videos. A disadvantage of the face detector for this purpose, is that it is not made for recognizing several face positions simultaneously, e.g. front and profile. Finally, the alignment of the box around the detected face can



**Figure 4.3:** Examples of the OpenCV face detection, the first image show a correctly detected face while the second, third and fourth image show examples of false positives and negatives.

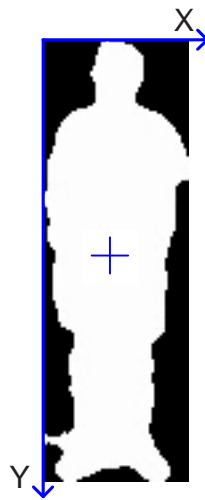
vary significantly even when the person is standing still. Though the face detector does an impressive job of recognizing faces directly from images, it is expected that a better result can be obtained by applying some assumptions like, background subtraction is possible and more that half of the person is visible.

## 4.4 Template Matching

Template matching is the method selected for the head extraction. The template matching is performed on an image of the person's mask, where the median of the image is known, as depicted in Figure 4.4. The template matching is based on matching the person's mask with a body model. The body model is build using an estimate of the person's size in order to make it scale invariant. Several steps of preprocessing are applied before the template matching is preformed.

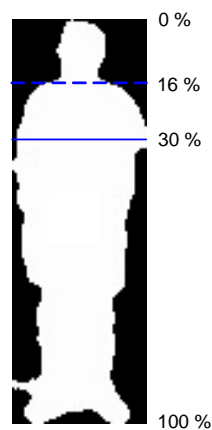
### 4.4.1 Trim Height

The height of the input image is reduced to the top 30 percent in order to limit the search area and reduce processing time. The top 30 percent should contain the head even if it is slightly misplaced, since the head normally only constitutes 16 percent of a person's height. The percentages are depicted on



**Figure 4.4:** The input mask with the median depicted as a cross, only the x-coordinate of the median is used in the template matching.

a test sample in Figure 4.5.

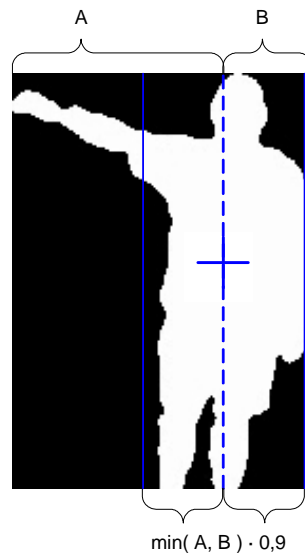


**Figure 4.5:** A test sample showing how percentages of the height can divide a person into areas of interest.

#### 4.4.2 Trim Width

In order to further reduce the search area, the width of the search area is trimmed to the person's estimated shoulder width. The shoulder width is

estimated as 90 percent of the minimum distance from the median to the edge of the image, as depicted in Figure 4.6. As shown, this also solves the issue of a person stretching an arm out to the side.



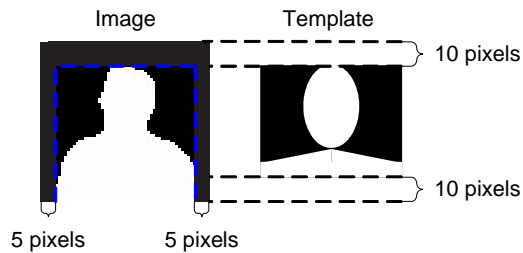
**Figure 4.6:** The solid lines shows how much is trimmed of each side. The punctured line shows the median of the person.

### 4.4.3 Padding

As part of the template matching the image is padded around the edge after it has been trimmed to allow for movement of the template. The image is padded with 5 pixels at each side and 10 pixels at the top. The image is not padded at the bottom. Instead the size of template is trimmed from the bottom. The padded image is shown leftmost in Figure 4.7.

### 4.4.4 The Template

The template consists of a head and torso and is of the same size as the image after it has been trimmed minus 10 pixels in the height, as depicted in the left part of Figure 4.7 on the next page.



**Figure 4.7:** The image padded with zeros is depicted to the left, while the template is shown to the right.

The template is build with the assumptions that the width of the head is 40% of the estimated shoulder width, and that the height of the head is 150% of the head's width. The 40% is based on visual inspection, and the 150% is based on [Department of Defence, 1991].

The template is drawn in the middle (equal to the median), and with the top of the head in  $y = 0$ . The torso is drawn as a rectangle with the shoulders sloping slightly downward. The sloping of the shoulders are used as a bias for fine tuning the match of the neck up and down. Currently the shoulders is set to fall 20 percent of the shoulder width, corresponding to a slope of  $11.3^\circ$ . The torso is extended to the bottom of the template. Note, that this might not be the optimum solution since the size-ratio between the head and torso is not locked..

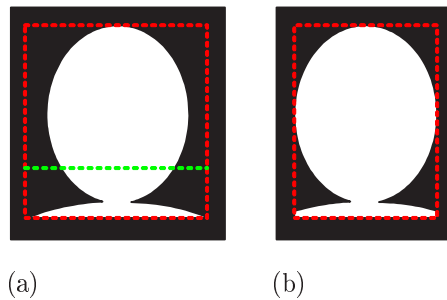
#### 4.4.5 XOR

The template is matched with the image using XOR, meaning that any mismatches between the two decreases the correlation. Because the image is not padded with zeros at the bottom, the template can move down without unnecessarily reducing the correlation.

The template is moved 5 pixels to each side and 10 pixels up and down, and the position with the highest correlation are saved as the best match.

### 4.4.6 Postprocessing

Using the template matching the location of the neck is estimated with high accuracy. However, the sides of the head are not. This is illustrated in Figure 4.8(a) where the bounding box is too wide. To compensate for this, the bounding box is narrowed. The input mask, that is covered by the initial bounding box, is searched for the left- and rightmost non-zero pixel. Only the top 66 percent is searched, because these do not include the shoulders. The 66 percent line is depicted in Figure 4.8(a). The bounding box is altered according to the found pixels. The result is seen in Figure 4.8(b).

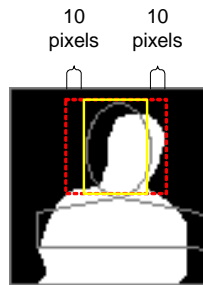


**Figure 4.8:** Before and after the postprocessing. a) Initial estimate. b) After Postprocessing.

This simple method has proven to be very reliable in locating the head edges. To take advantage of this, the head box found by the template matching is extended with ten pixels to each side before trimming it again, as depicted in Figure 4.9. This solves the situations as in the figure where the person is tilted.

### 4.4.7 Fall Back

If the input image is too small the top 16 percent of the person's height is used as a fall back estimate. At present the fall back option is only used when the input image is less than 6x11 pixels, meaning that it is practically never used. The fall back generally provide a good estimate of the head (in many situation almost as good as the template matching), but fail if the



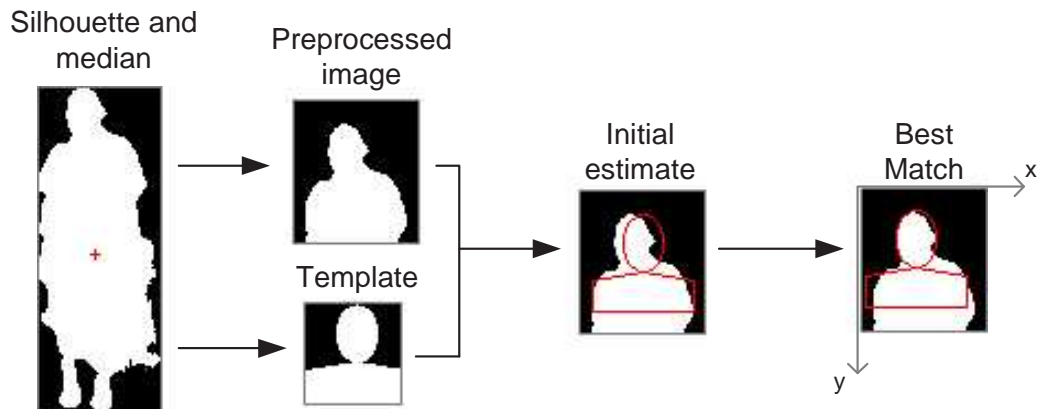
**Figure 4.9:** The yellow depicts the box extracted by the template matching, and red lines depicts the extension of this head box before the postprocessing is applied.

person's bounding box for some reason do not match the person's height. An advantage of the fall back is that it always work, no matter distance and angle of the person or quality of the image. Figure 4.5 shows the 16 percent line on one of the test samples.

#### 4.4.8 Results

The initial estimate of the head size and position from the shoulder width usually provides a good estimate in it self, therefore the main function of the template match is to provide additional fitting and compensate for minor errors rather that locating the head in a large search area. This does however not neglect the importance of the template matching since the later head orientation module require a tight and stable match of the head. Figure 4.10 depicts the overall template matching procedure, and provide an example of the difference between the initial estimate and the best match found by the template matching. Experimental tests show that the method provide a very good estimate of the head location at all resolutions, and with very little between-frame variation.

This method does not work well when the person is seen from the side, because the shoulder width is estimated to small. Furthermore, the opposite situation can also make the method fail, this happens when the shoulder width is estimated to large e.g. when a person is stretching out both arms



**Figure 4.10:** The states of the overall template matching procedure from input to output. In the two final steps, from the initial estimate to the best match, the position of the template is corrected with  $(x, y) = (-4, -1)$ .

or due to segmentation errors. Figure 4.11 show examples of the limitations of the method.

#### 4.4.9 Additional Results

The template matching was tested without drawing the torso, meaning the template was reduced so that it only contained the head. This performed almost equally well and reduced the search area. However, the full head and torso was kept because of its slightly higher stability. Depending on the precision needed in the steps following the head extraction, the head-only template could be a durable solution.

If the initial estimate of the head size and position for some reason is hard to make, the search area can be widened, the movement of the template increased and scaling of the template can be added as an additional degree of freedom. The template matching has been tested with up to 70 percent of the person's height instead of 30 percent, with padding up to 20 and 80 pixels instead of 5 and 10 pixels, and with scaling of the template between 1.4 and 0.7 times the initial estimate (in steps of 0.1). These enhancements solves some cases where the initial estimate is poor. However, these cases are

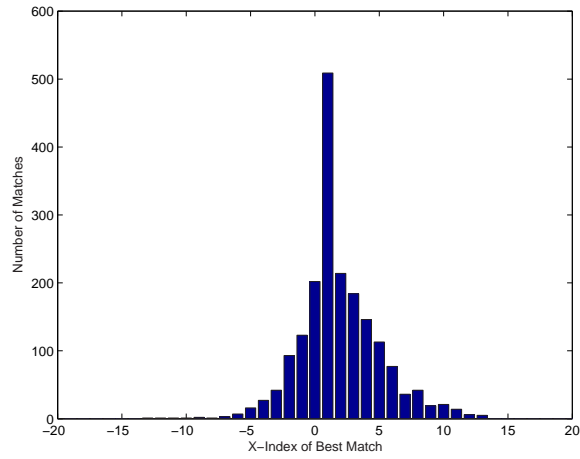


**Figure 4.11:** Three examples of the limitations of the head extraction. First column: nearly correct initial estimation, but mismatch due to segmentation errors. Second column: too small initial estimation (person seen from the side) causing erroneous head extraction. Third column: too large initial estimate, but since the head box is not allowed to exceed the original bounding box at the top of the person (not depicted), the head is extracted correct.

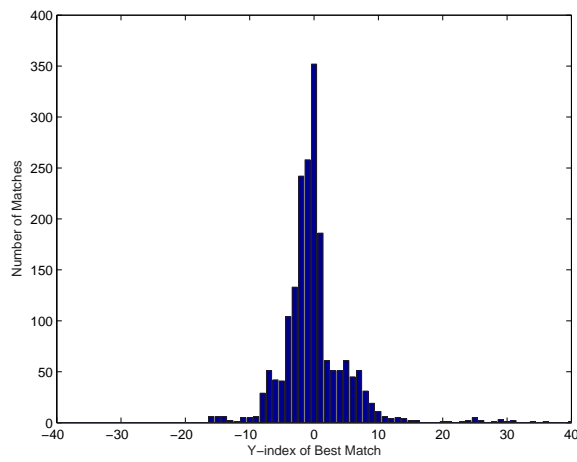
rare and the enhancements are heavy to process and are therefore not part of the final solution.

Figure 4.12 show the indices of the best matches relative to the initial estimates. This gives an idea of how much it affects precision to choose a template movement of  $\pm 5$  and  $\pm 10$  pixels. As seen from the graphs in Figure 4.12 the majority of the best matched is detected within a range of  $\pm 5$  pixels for the x-movement and  $\pm 10$  pixels for the y-movement. Additionally, more than half of the best matches outside this range is caused by erroneous frames, e.g. a person with the side to the camera.

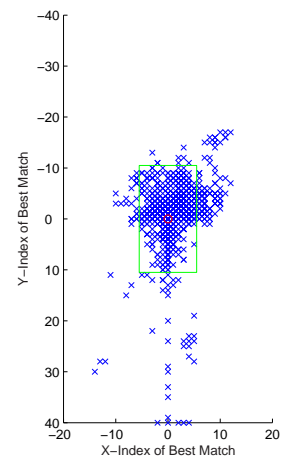
Figure 4.13 show an example of a match that could be improved by allowing additional sideways movement of the template. However, in this case the  $\pm 5$  pixel sideways movement is sufficient to get a correct result due to the postprocessing described in Section 4.4.6.



(a)

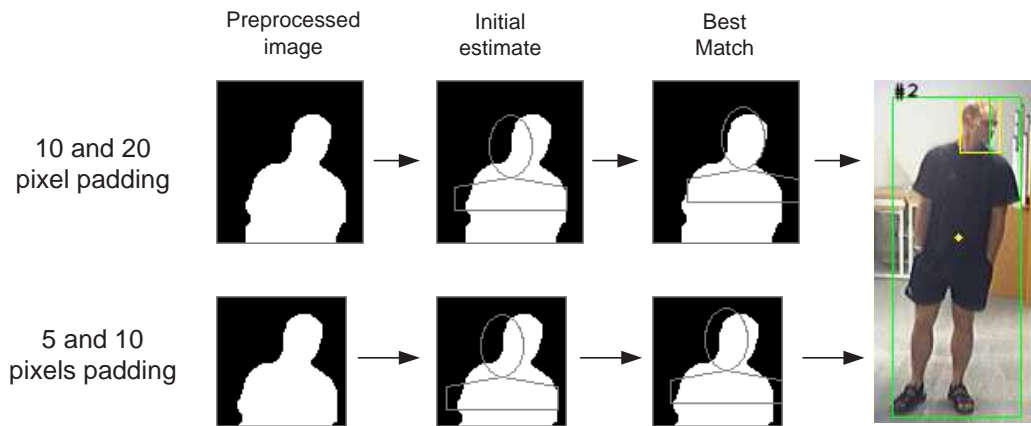


(b)



(c)

**Figure 4.12:** The three figures show the index of the best match relative to the initial estimate for 1925 frames (including erroneous frames), with an x-movement of  $\pm 20$  and y-movement of  $\pm 40$  pixels. Figure (a) and (b) show distribution of the indices for the x and y axis, while Figure (c) shows the distribution in both dimensions. The green box in Figure (c) corresponds to x and y padding of 5 and 10 pixels.



**Figure 4.13:** Example of head extraction of a tilted person where the precision of the template matching is improved by allowing the template to move  $\pm 10$  pixels sideways instead of  $\pm 5$  pixels.

## 4.5 Summery

In this chapter different approaches for extracting the head region were analyzed. First, methods based on the horizontal projection of the binary mask was reviewed. The idea was that the neck can be found as a local minimum. However, tests showed that the method was very unstable.

Next, the face detector developed by [Viola and Jones, 2004] was applied. The face detector was able to find the head very well under strict conditions. However, the fact that the method is not orientation invariant mean that it can not be used in this work.

The chosen method is based on template matching. A template is build from the binary mask using assumptions about the person's shape. The template is matched against the binary mask, and the best correlation is used to extract the head region. Tests show that the method performs well as long as the person's body face the camera.

# CHAPTER 5

---

## Head Orientation

This chapter describes three approaches for determining the head orientation. The first method looks for symmetry in the image. The face is assumed to be more symmetric when looking straight forward. The second method finds a lot of features in the face and track these for a short period of time. The change in orientation can when be found as the average change in feature positions. The last method uses a moment-based approach inspired by [Park and Aggarwal, 2000].

### 5.1 Symmetry/Correlation

A possible approach to finding the head orientation, is to look for symmetry in the face. Whenever a person is looking straight forward the two halves of his face are more similar than when he is looking to the side. One way to exploit this, is to find the correlation between the two halves.

#### 5.1.1 Method

The basic idea is to take the image of the face and flip it horizontally. Then the correlation between the unflipped and flipped image can be found. If the

correlation is high, the person is looking forward. If it is low, his is looking to the side. It is not possible to determine which side the person is looking to.

The above mentioned method is too simple to be directly applied due to a couple of issues. The face is not necessarily centered. Furthermore, the face might be slightly tilted. The following is the outline of the applied method, which solves these issues:

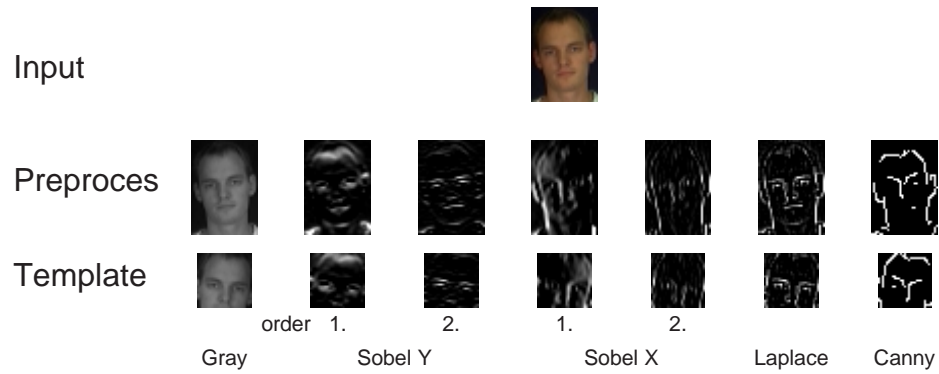
1. Make a template by flipping the input image and reducing the size 10% on each side. This solves the issue of non-centered faces.
2. Rotate the template from  $-2^\circ$  to  $2^\circ$  to compensate for a possible tilted face.
3. For each rotated template perform a template matching.
4. Save the best found correlation.

It was discovered that the method did not perform very well. It was suspected that the unevenly illumination of the person was the reason for this. To help suppress the unevenly illumination, the input image is first preprocessed to enhance more significant features such as the eyes and edges. The tested preprocessing methods are illustrated in Figure 5.1. Different sobel-operators have been tried. The best results are obtained with the 1st order operator that finds edges in the horizontal direction.

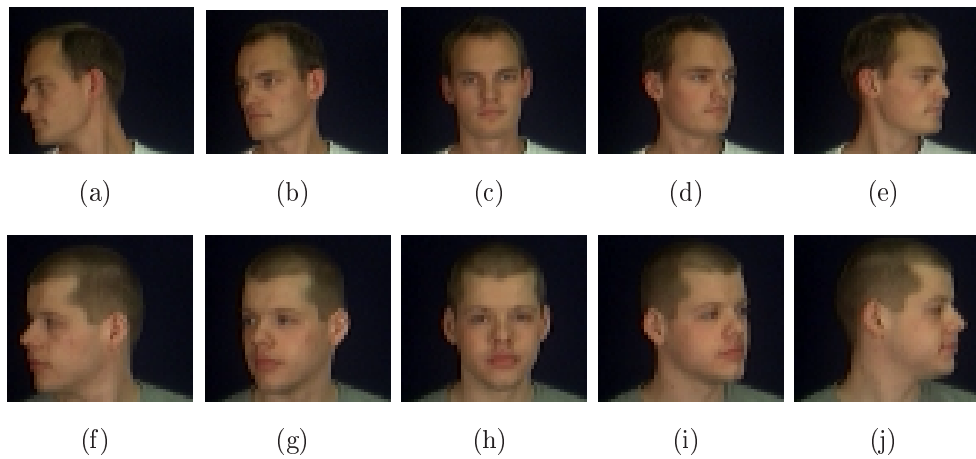
### 5.1.2 Results

The method has been tested using two recording. Two different persons have been recorded under fairly controlled lighting conditions. The persons are standing still looking in five directions as seen in Figure 5.2. Each head position is hold for approximately 100 frames.

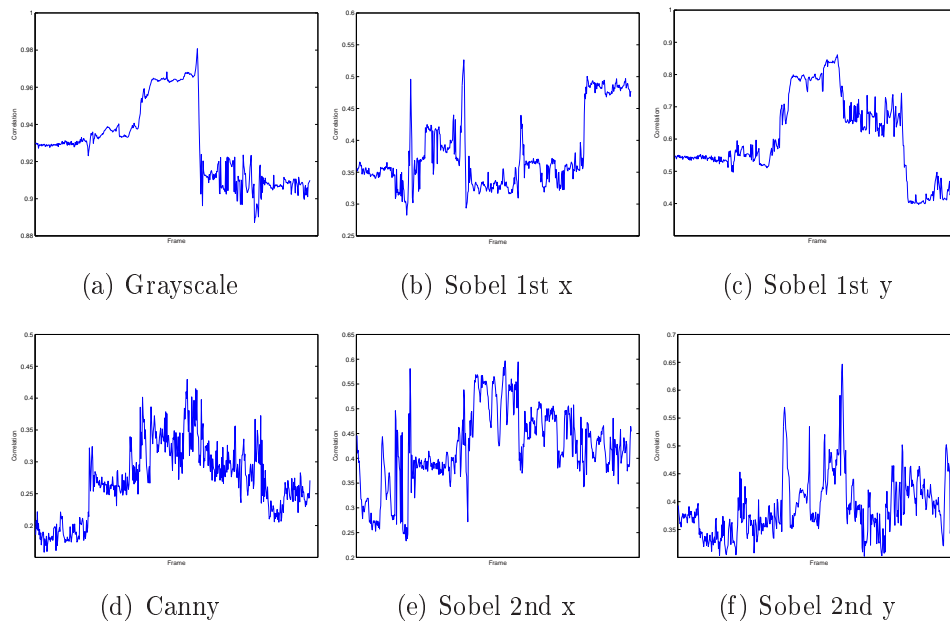
Each of the mentioned preprocessing methods have been applied to the test recordings. The optimal result would be three different correlation levels - one for forward looking, one for  $45^\circ$  for both sides and one for  $90^\circ$  for both



**Figure 5.1:** Several different preprocessing approaches have been analyzed. At first the grayscale image was used. Later different edge detectors were tried.



**Figure 5.2:** Row 1: Head orientation for test subject 1. Row 2: Head orientation for test subject 2. Each orientation is hold for approximately 100 frames



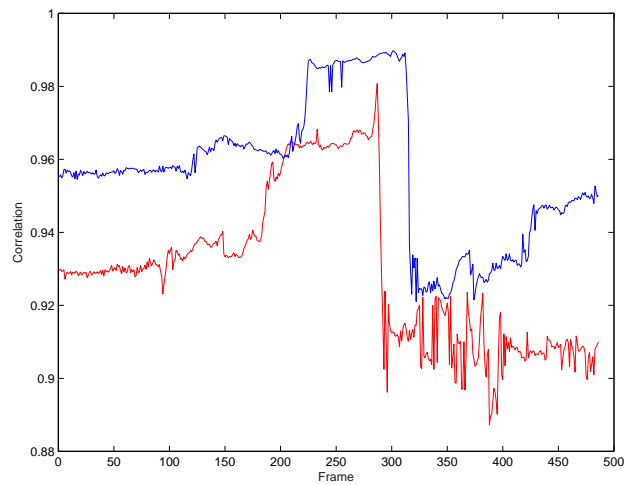
**Figure 5.3:** Results of applying different preprocessing methods.

sides. The actual results are shown in the six graphs in Figure 5.3. Each graph represent a different preprocessing method. It is not possible to clearly distinguish three different correlation levels in any of the graphs.

Using grayscale as preprocessing it is possible to determine when the person is looking forward. However it is not possible to determine how much to the side the person is looking.

Still, it seems possible to at least detect when the person is looking forward by choosing a appropriate threshold. However, one person might have a higher correlation when looking to the side than another person looking straight forward. This situation is shown in Figure 5.4.

Based on the tests it is concluded, that a simple symmetry approach is not robust enough for estimating the orientation of a person's head. The method is therefore abandoned.



**Figure 5.4:** The blue graph is the correlation for test person 1 and the red for test person 2. Test person 1 has a higher correlation looking sideways, than person 2 looking forward.

## 5.2 Feature Tracking

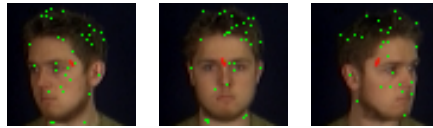
A method for estimating the head orientation is to calculate the optical flow of the person's head. This method does not find the absolute head orientation, but rather the change in orientation. To calculate the absolute orientation the method could be used in conjunction with e.g. the symmetry approach described in the previously section.

The idea is that facial features are found in the person's head. These features are then tracked for short period of time, after which the average displacement is found. The feature displacement tells which way the person turns his head. To allow the person to move around in the scene, the person's global translation must be compensated for.

### 5.2.1 Find Facial Features

Facial features in the person's head are found using the OpenCV function `cvGoodFeaturesToTrack()`. The function finds strong corners in an image using the principle described in [Shi and Tomasi, 1994]. Examples of features

found using the algorithm is shown in the images in Figure 5.5.



**Figure 5.5:** Features found using `cvGoodFeaturesToTrack()` is shown in green.

### 5.2.2 Feature Tracking

The found features are tracked for a consecutive number of frames. This is implemented using the OpenCV function `cvCalcOpticalFlowPyrLK()`, which calculates optical flow for a sparse feature set using iterative Lucas-Kanade method in pyramids [Bouguet, 1999].

The length of the period, that the features are tracked, is set to 15 frames which corresponds to one second. If the period is too short, the features will not move enough and it will be hard to precisely estimate the displacement of the features. If the period is too long, small variations might not be detected. This could happen if the person turns his head in several directions during the tracking period.

### 5.2.3 Compensation for Head Translation

The person is not necessarily standing still. The displacement of the feature will therefore be a combination of the global translation of the person and the displacement of the person's head. Only the displacement of the person's head is wanted. The global translation is expressed in the median point of the person (see Section 3.3 on page 16). The median point of person is not always following the movement of the head. Instead the median point of the head is therefore used to compensate for the head translation.

### 5.2.4 Result

A short video showing the result of the feature tracking method can be found on the enclosed DVD (© /Video\_Result/featuretracking\_direction10.avi). In the video the red line indicates which direction the person is turning the head. The line is showing the displacement for the last second. In this small video the algorithm works well. However, in a scene where the person is moving around, see (© /Video\_Result/featuretracking20060718\_1.avi), it is impossible to say which way the head is turning. The displacement describing the turning of the person's head is very small compared to the global translation of the head. Small errors in the estimation of global translation leads to inaccurate estimation of the change in head posture.

The conclusion is that the method can not be used to estimate the head orientation when the person is moving.

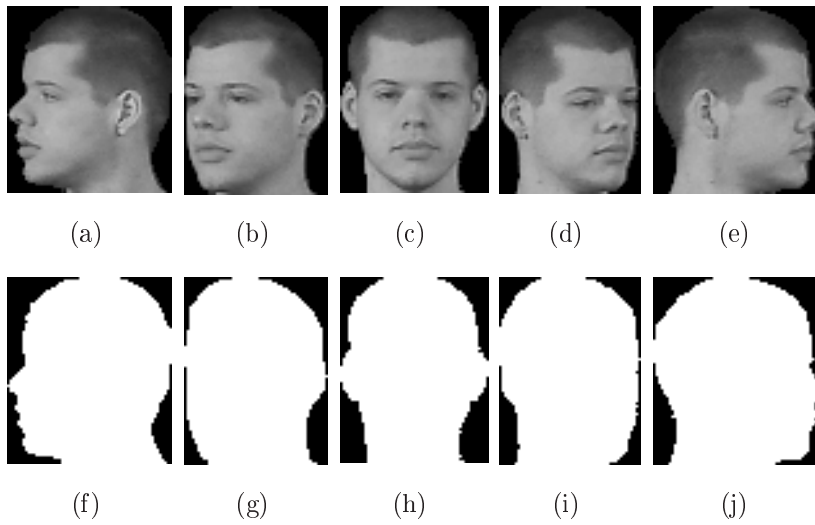
## 5.3 Moment

This section presents a moment-based approach for finding the head orientation. The approach is based on a method presented in [Park and Aggarwal, 2000]. The method calculates moments of subimages in the input image. Each calculated moment constitutes a feature. The classification is done using k-nearest-neighbor.

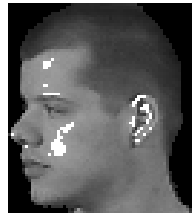
The input for this module is a gray scale image of the head, and a corresponding mask of the head, as seen in Figure 5.6. The head images is divided into five classes corresponding to an orientation of  $-90^\circ$  ,  $-45^\circ$  ,  $0^\circ$  ,  $45^\circ$  and  $90^\circ$  .

### 5.3.1 Highlights

Before any calculations are made, highlights are removed from the input image. Highlights can appear due to face items such as glassed or earrings, but skin e.g. on the nose also often create highlights. By removing them the classification is made more robust and insensitive to difficult lighting conditions. The highlights are defined as:



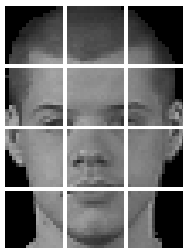
**Figure 5.6:** Examples of input/training images for the moment-based head orientation classifier. Each head image is shown with its corresponding mask. From left to right the images show an orientation of  $-90^\circ$ ,  $-45^\circ$ ,  $0^\circ$ ,  $45^\circ$  and  $90^\circ$  degrees.



**Figure 5.7:** Shows removed highlights as white.

$$(intensity(pixel) - mean(image)) / std(image) > 1.5$$

Figure 5.7 provide an example of removed highlights. Highlights are removed by zeroing them in the mask.



**Figure 5.8:** The grid with 12 subimages.

### 5.3.2 Feature Calculation

In [Park and Aggarwal, 2000], the image is divided into twelve subimages using a  $3 \times 4$  grid as depicted in Figure 5.8.

For each subimage a feature is calculated using the measure:

$$W = \frac{\text{mean}(\text{subimage}) - \text{mean}(\text{image})}{\text{std}(\text{image})}$$

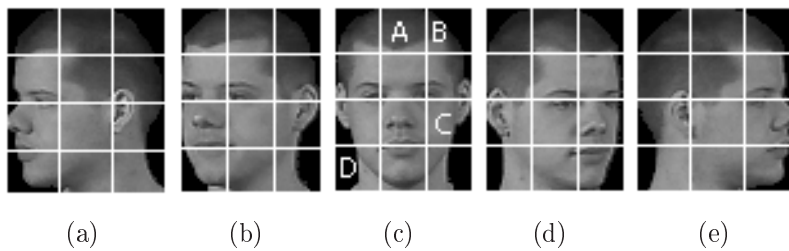
where  $\text{mean}(\text{image})$  and  $\text{std}(\text{image})$  refer to the mean and standard deviation of the full image, and  $\text{mean}(\text{subimage})$  is the mean intensity of a subimage. In both the *image* and *subimage* only those pixels that are valid in the mask are part of the calculation.  $W$  is the feature value and with twelve subimages a 12 dimensional feature vector is calculated for each image:

$$X = [W_1 \ W_2 \ W_3 \ W_4 \ W_5 \ W_6 \ W_7 \ W_8 \ W_9 \ W_{10} \ W_{11} \ W_{12}]$$

The calculated features are scale invariant and insensitive to imperfections in the image [Park and Aggarwal, 2000].

### 5.3.3 Modified Feature Calculation

By inspecting the values of the 12 features in the training data, it seems that some features provide very little or no information, especially the corners have a poorly separated mean value and a high variance. Furthermore, many of the features are correlated and sensitive to the same errors. In Figure 5.9



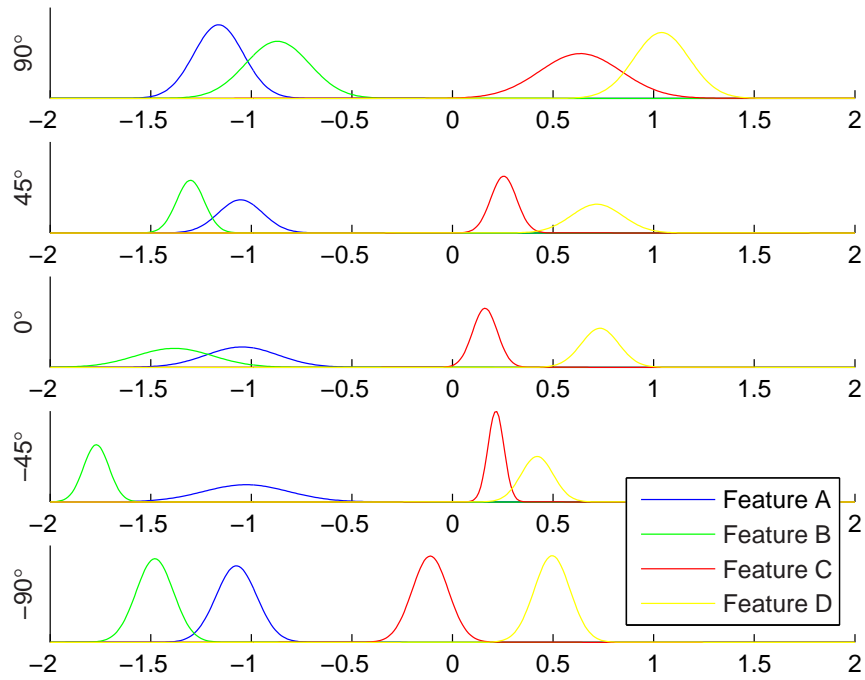
**Figure 5.9:** The features of the four subimages with letters provide examples of subimages with limited information. From left to right the images show an orientation of  $-90^\circ$ ,  $-45^\circ$ ,  $0^\circ$ ,  $45^\circ$  and  $90^\circ$  degrees.

four of the 12 subimages are selected to provide examples of these issues. The feature values of the subimages are plotted in Figure 5.10. As shown, none of the features clearly separate the orientations; especially feature A is a poor choice since it provide no information at all.

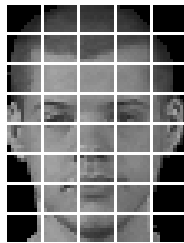
In an attempt to improve this, the grid is changed to a  $5 \times 8$  grid with 40 subimages as shown in 5.11. By inspecting the values of the new grid, the majority of the new features were discarded and the four most discriminative were selected. Those selected features are depicted in Figure 5.12. The numbers in the subimages corresponds to the numbering of the features, and the plot in Figure 5.13 depicts their mean value and variance. The plot is based on the training data consisting of images from two persons, five images at each of the five head orientations per person, which results in 50 pictures totally.

The plot shows that feature 1 are good at separating head orientations  $90^\circ$  and  $45^\circ$  from the rest, and feature 3 separates feature  $-90^\circ$  and  $-45^\circ$  from the rest. Feature 2 separates the  $\pm 90^\circ$  from the rest. Feature 1, 2 and 3 thus provide sufficient information to make a classification. Feature 4 provides additional information for separating head orientation  $0^\circ$  from the remaining orientations.

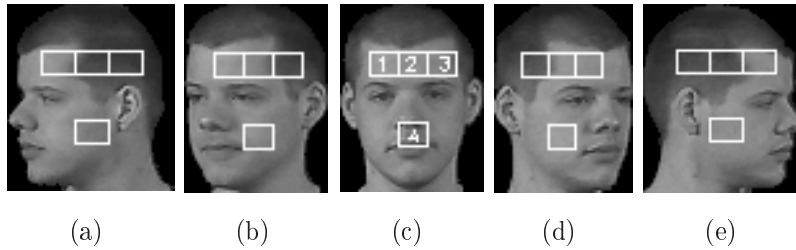
The reason why the features in the forehead (1, 2, and 3) provide a good measure of orientation, is that the hair covers some of the subimages when the head turns. The subimages are placed with a small margin to the hair and eyebrows at the top and bottom in order to avoid misleading values due



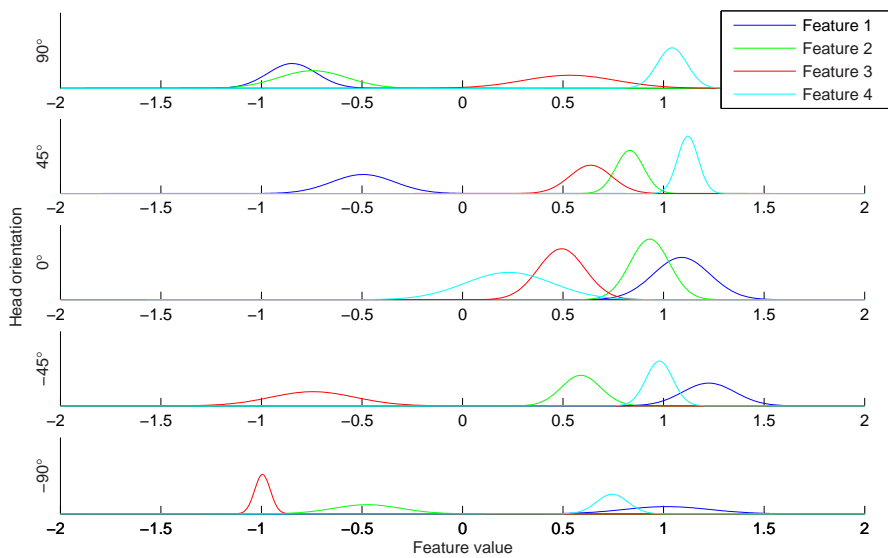
**Figure 5.10:** The plot shows the mean and variance of four features. The plots are calculated from the training images.



**Figure 5.11:** The grid with 40 subimages.



**Figure 5.12:** The grid with 4 subimages. From left to right the images show an orientation of  $-90^\circ$ ,  $-45^\circ$ ,  $0^\circ$ ,  $45^\circ$  and  $90^\circ$  degrees.



**Figure 5.13:** The plot shows the mean and variance of the four features, calculated from the four selected subimages. The plots are calculated from the training images.

to poor alignment. The limitations of this approach is that persons with hair lighter than the skin or no hair are unlikely to be classified correctly.

The last feature (4) is useful since the mouth/nose area often is darker than the cheek. In case the person has a full beard this feature may also be misleading. However, this is not a problem in the test and training videos, and therefore the feature is used.

It is not possible to utilize finer details from areas such as the eye and nose region with this method due to the low resolution and coarse feature measure.

### 5.3.4 k-Nearest-Neighbor

The classification is made using k-nearest-neighbor and the previously mentioned 50 training images. The classification is based on the 7 nearest.

### 5.3.5 Results

The resulting solution work down to a head size of  $10 \times 15$  pixels (width  $\times$  height).

The reduction from 12 to 4 features improve the results on our test videos, but it is possible that it makes the solution less general. More test data would be needed in order to confirm or reject this claim.

The method is tested on an indoor video (`© /Video_Results/20060718_1x_output.avi`) and an outdoor video (`© /Video_Results/20060817_1x_output.avi`). The confusion matrix of the indoor scene is presented in Table 5.1 and the confusion matrix of the outdoor in Table 5.2. Frames containing sever segmentation errors, persons entering or leaving the scene or standing with the side to the camera are not included in the confusion matrices. In general, it is difficult to define ground truth for the person's head orientation, especially in the transition from one orientation to another, these situations are resolved by empirical decisions.

As can be seen from Table 5.1 the classification has a significant number of misclassification. The misclassifications in the adjacent classes mainly occur

in the transition from one orientation to another. What happens in the transitions period is that the classifier alternately classifies the two adjacent orientations in 1 or 2 frames, this behavior of the classifier could be detected, and these misclassification is therefore considered less serious than the rest.

The misclassification of the non-adjacent classes in the indoor scene is mainly due to two issues. The first issue is the light; the video is recorded in a cellar with very little ambient light and with lamps that creates non-uniform shadowing of the face. This causes a significant number of the non-adjacent misclassification, e.g. all of the 16 misclassification in the upper right corner of the confusion matrix is due to this issue.

A particular noticeable property of the matrix is that the  $0^\circ$  column has an over-representation of misclassification, this is due to the second issue which is motion blur. In the scene the test persons is moving sideways between different positions, facing the camera as they walk, which creates a heavy blur of the face causing misclassification. The occurrence of these two issue may be detected and removed by temporal filtering as these misclassifications rarely are stable for more than 1 or 2 frame at a time and is alternating between both adjacent and non-adjacent classes, while the correct classification typically is stable for 5+ frames. Furthermore, a better camera than the webcam used (Philips ToUCam PRO II), may reduce or remove the blurring issue.

	$-90^\circ$	$-45^\circ$	$0^\circ$	$45^\circ$	$90^\circ$
$-90^\circ$	220	13	51	5	16
$-45^\circ$	23	100	61	3	1
$0^\circ$	31	45	469	31	11
$45^\circ$	0	19	44	106	21
$90^\circ$	7	3	39	6	207

**Table 5.1:** Confusion table for an indoor scene with difficult lighting conditions from fluorescent tubes.

The outdoor scene is recorded when the sky is overcast and the lighting conditions is therefore much better than in the indoor scene. Furthermore, there is no sideways movement of the person, since he is walking directly

towards the camera from a faraway distance. Therefore, the two major issues in the indoor scene do not occur in this video. As a result there is less misclassifications in this scene as shown in Table 5.2. The transition between orientations are also a problem in the outdoor scene which account for the adjacent misclassifications. The misclassifications in the last row is caused by quantization due to the persons distance from the camera. At a head resolution of  $10 \times 15$  pixels it starts to be difficult for the human eye to determine the correct head orientation, and misclassifications occur. For some reason the  $90^\circ$  class is most receptive to these misclassifications.

	$-90^\circ$	$-45^\circ$	$0^\circ$	$45^\circ$	$90^\circ$
$-90^\circ$	34	2	0	0	0
$-45^\circ$	2	23	0	0	0
$0^\circ$	0	11	67	1	0
$45^\circ$	0	0	0	17	2
$90^\circ$	2	1	4	11	89

**Table 5.2:** Confusion table for an dynamic outdoor scene with good lighting conditions and trees, water and cars.

### 5.3.6 Additional results

A couple of variants of this solution is tested. Instead of calculating the features with the measure presented in [Park and Aggarwal, 2000], a uniformity measure is used. From observing the 40 subimages the difference in uniformity seems significant in some subimages. The measure used is found in [Gonzalez and Woods, 2001] and is given as:

$$U = \sum_{i=0}^{255} p(z_i)^2$$

where  $z$  is a random variable denoting the gray levels, and  $p(z_i)$  is the value of each bin in the gray scale histogram. However, experimental tests showed that the uniformity measure is too sensitive (large feature variance) to be used as a reliable feature, especially when used on different persons.

The second idea that is tested is moment calculations based on the entire silhouette of the head. The idea is to use the seventh Hu-moment as a feature. The reason for using this moment is that it is invariant towards rotation, translation and scaling, but not mirroring. Thereby, the method could potentially separate all five head orientations using only a one feature value. The test showed that it does separate the orientations of a single person, however, as other higher order moments this method is very sensitive to any variation, and it performs poor when different persons are used.

Both the uniformity measure and Hu-moment is discarded because the feature variance is to large compared to the separation of the classes.

## 5.4 Summery

Three methods for estimating the head orientation have been investigated. The idea in the correlation method is that the face is more symmetric when looking forward. The method is not able to distinguish between left and right. Tests show that it is not possible to use a single threshold across all persons to determine the head orientation.

The feature tracking method finds the change in head orientation. It does not give the absolute orientation. Features are tracked for one second, whereafter the average displacement of the features are found. The global displacement of the person is compensated for using the median point of the head. Tests show that the method is very unreliable, because the displacement of the features are lower than the noise of the global displacement.

The chosen method is a moment-based approach inspired by [Park and Aggarwal, 2000]. In the method the head region is divided into 5 x 8 grid. The three regions covering the forehead and the one covering the mouth are chosen. For each region a feature is extracted, resulting in a 4-dimensional feature space. Using these four features it is possible to distinguish between 5 different head orientations. Tests show that most of misclassifications are classified as an adjacent orientation.

## CHAPTER 6

## Conclusion

In this study a functional system for detecting human head orientation from low-resolution video recording has been developed. The detection of head orientation form basis for determining the attention of a person, which is the primary objective for this work. The four major parts of the system are the figure-ground segmentation, person tracking, head extraction and the head orientation. The solutions for the figure-ground segmentation and parts of the tracking were selected based on pervious studies at the CVMT Laboratory. Thus, these two parts of the system contains no major research contributions, but rather two proven and functioning solutions.

In developing the head extraction several approaches were tested, and the best results were obtained using template matching. The template matching solution is made scale-invariant by constructing the template using an estimate of the person's size. The solution performs very well at all resolutions.

Like in the head extraction, several approaches for finding the head orientation has been tested. The chosen solution inspired by a moment approach, was modified to increase performance in the test video. In order to fully understand the validity of the improvements further work needs to be done. The performance and limitations of the head orientation has been documented, and suggestion for improvements has been presented. The solution perform

reasonable in detecting the head orientation, and tests show that temporal filtering would improve the performance considerable.

The full system is capable of running in real time or near real time on a 1.3 GHz machine at a resolution of  $320 \times 240$  pixels.

### 6.0.1 Future Work

There are four main areas which need to be addressed to further analyze the developed system.

First of all it would be interesting to perform a temporal filtration on the estimated head orientations. This would remove many of the wrong estimates. A way to perform the filtering could be to use the most occurring estimate in a sliding window, or require the orientation estimate to be stable for at least 5 frames in order to be valid. A similar approach have been used successfully in [Andersen et al., 2006].

The next step would then be to extract where a person's attention is directed at in a scene. By analyzing the temporal filtered head orientation estimates, it can be determined if the attention is to the left, right or straight ahead.

A comparison between the original method by [Park and Aggarwal, 2000] and the derived method used in this work is also needed. It is interesting to analyze if any improvement has been achieved.

Finally, more test person is required. In this work the system has been trained using recordings of only two different persons, and the system has been tested using the same two person. The test persons should be more dislike than the two used in the current system. Person of different gender with different hairstyles etc. is preferable.

## BIBLIOGRAPHY

- [Andersen et al., 2006] Andersen, J. R., Duizer, P. T., Hansen, D. M., and Mortensen, B. K. (2006). Automatic annotation of humans in surveillance video recordings. Student report, Aalborg University - Laboratory of Computer Vision and Media Technology.
- [Bouguet, 1999] Bouguet, J.-Y. (1999). Pyramidal implementation of the lucas kanade feature tracker. Technical report, Intel Corporation, Microprocessor Research Labs.
- [Chalidabhongse et al., 2003] Chalidabhongse, T. H., Kim, K., Harwood, D., and Davis, L. (2003). A perturbation method for evaluating background subtraction algorithms. *Joint IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance*.
- [Department of Defence, 1991] Department of Defence (1991). Anthropometry of us military personnel. <http://assist.daps.dla.mil/docimages/0000/40/29/54083.PD0>. DOD-HDBK-743A.
- [Gonzalez and Woods, 2001] Gonzalez, R. C. and Woods, R. E. (2001). *Digital Image Processing*. Prentice Hall.
- [HERMES, 2006] HERMES (2006). Human expressive representations of motion and their evaluation in sequences. <http://www.hermes-project.eu/>.

- 
- [Park and Aggarwal, 2000] Park, S. and Aggarwal, J. K. (2000). In *Head segmentation and head orientation in 3D space for poseestimation of multiple people*, Austin, USA.
- [Shi and Tomasi, 1994] Shi, J. and Tomasi, C. (1994). Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, Seattle.
- [Viola and Jones, 2004] Viola, P. and Jones, M. J. (2004). Robust real-time face detection. In *International Journal Computer Vision*, Hingham, MA, USA.

# APPENDIX A

## Code Review

The software is implemented in C++ in Visual Studio 2005 using OpenCV Beta 5. Each of the described modules in the technical report has its own class. (CodebookModel, Tracking, HeadExtraction and HeadOrientation)

The code shown in Listing A.1 is the main function of the program. First the properties of the input video are retrieved. These are used to set up the codebook model. When all variable have been set up, the program enters the main loop. Three different situations can occur; the system can be online, training or in transition between the two. In the last two the codebook is being trained with the `trainCodebook()` function. Furthermore, in the transition case, all non periodic codewords are removed with the `cleanupCodebook()` method. When the system is online the foreground is extracted with the `subtractBackground()` function. The result is given to the tracking module via `DoTracking()`.

---

```
1 int main(int argc, char* argv[]) {  
  
    CvCapture* capture;  
    static CvMemStorage* storage = cvCreateMemStorage(0);  
    IplImage* f; // Image captured from video  
6    IplImage* frame; // Copy of the captured image  
    IplImage* result; // Result from the codebook  
    IplImage* filtered; // The binary mask from the codebook after  
        it has been post-processed  
    unsigned int framePos = 1;
```

```

11 //Codebook:
CodebookModel* codebookModel;

//Tracking:
Tracking* track = new Tracking(TRACK_TIMEOUT);

16 //Windows:
cvNamedWindow("Video", 1);

//Choose capture source:
21 capture = cvCaptureFromAVI(INPUT_FILE);

if(capture) {
//Setup parameters according to input video:
CvSize imageSize = cvSize((int)cvGetCaptureProperty(capture,
CV_CAP_PROP_FRAME_WIDTH), (int)cvGetCaptureProperty(capture,
CV_CAP_PROP_FRAME_HEIGHT));
26 int numOfFrames = (int)cvGetCaptureProperty(capture,
CV_CAP_PROP_FRAME_COUNT);
int framesPerSecond = (int)cvGetCaptureProperty(capture,
CV_CAP_PROP_FPS);

//Construct the codebook:
codebookModel = new CodebookModel(imageSize, E1, E2, ALPHA, BETA,
LEARN, STABLE_BG, ONLINE_TRAINING_TIME, ONLINE_EXPIRATION_TIME,
ONLINE_MNRL_THRESHOLD, TRAIN_MNRL_THRESHOLD);

31 //Main loop:
while(cvGrabFrame(capture)) {
//Retrieve frame
f = cvRetrieveFrame(capture);
36 if(!f) //Did retrieval go OK?
break;

//Make a copy of the captured frame – this should be done
according to opencv doc:
frame = cvCreateImage(cvGetSize(f), IPL_DEPTH_8U, f->nChannels);

41 // The system is online (the codebook has been trained):
if (framePos > TRAIN_PERIOD+1) {
//Initialize the result:
result = cvCreateImage(cvGetSize(frame), IPL_DEPTH_8U, 1);
46 //Initialize the filtered result:
filtered = cvCreateImage(cvGetSize(frame), IPL_DEPTH_8U, 1);

//Subtract background and store in result:
codebookModel->subtractBackground(frame, result, framePos);

51 //Post-processing, 5x5 median filter:
cvSmooth(result, filtered, CV_MEDIAN, 5, 0, 0);

```

---

```

56         //Tracking:
        track->DoTracking(filtered , frame , framePos);

        //Show result:
        cvShowImage("Filtered", filtered);

61         //Release images
        cvReleaseImage(&result);
        cvReleaseImage(&filtered);

        //The training period is done (TRAINING -> ONLINE):
66     } else if (framePos == TRAIN_PERIOD) {
        //Train on the last training frame
        codebookModel->trainCodebook(frame, framePos);

        //Clean up the codebook - remove non-periodic codewords
71     codebookModel->cleanupCodebook(framePos);

        //Create a window to show the filtered result from the
        //segmentation
        cvNamedWindow("Filtered", 1);

76     //The codebook is being trained:
    } else {
        codebookModel->trainCodebook(frame, framePos);
    }

81     framePos++;
    cvShowImage("Video", frame);
    cvReleaseImage(&frame);
}

86 //Destroy codebook model:
codebookModel->~CodebookModel();

//Destroy tracking:
track->~Tracking();

91 //Destroy windows:
cvDestroyAllWindows();

//Release:
96 cvReleaseCapture(&capture);

    return 0;
}
}

```

---

**Listing A.1:** The main loop in the program.

The `DoTracking()` function is outlined in Listing A.2. First the blobs in the binary input mask are found. These are merged using the criteria described in Section 3.1. Then the merged blobs are being classified as either humans or “noise”. The human blobs are being tracked. The head orientation of each human are then being found with the `findDirection()` function.

---

```

void Tracking::DoTracking(IplImage *mask, IplImage *org, unsigned int
    frameNumber) {

    IplImage* inputcontour = cvCreateImage(cvGetSize(mask), IPL_DEPTH_8U,
        mask->nChannels);
    cvCopyImage(mask, inputcontour);

5
    //Find blobs:
    int numBlobs = cvFindContours(inputcontour, contourstorage, &contours,
        sizeof(CvContour), CV_RETR_EXTERNAL, CV_CHAIN_APPROX_NONE);

    if (numBlobs > 0) {
10
        //Merge blobs:
        vector<Blob> *blobs = mergeBlobs(contours, cvGetSize(inputcontour));
        int numMergedBlobs = blobs->size();

        for (int i = 0; i < numMergedBlobs; i++) {
15
            if (objectClassification((*blobs)[i])) {

                //Do tracking after object classification:
                Track* track = FindMatchingTrackID((*blobs)[i]);
                if (track == NULL) { //No matching track
20
                    track = AddTrack((*blobs)[i], frameNumber);
                } else { //Matching track
                    UpdateTrack((*blobs)[i], track, frameNumber);
                }

                //Find head orientation:
25
                findDirection(org, mask, track);
            }
        }

30
        delete blobs;
        cvClearMemStorage(contourstorage);
    }
    //Clean up expired tracks:
    CleanupTracks(frameNumber);

35
    //Free the temp contours-image:
    cvReleaseImage(&inputcontour);
}

```

---

**Listing A.2:** The `doTracking` function.

The purpose of the `findDirection()` (outlined in Listing A.3) is to construct the necessary images for finding the head orientation. First the location of the head is found using the mask of the person. This is done as described in Section 4.4 using `findHead()`. Two binary masks are constructed, which can be used for determining the orientation of the head using one of the methods described in Chapter 5.

---

```

void Tracking::findDirection(IplImage* frame, IplImage* mask, Track* track){
2
    //bodymask:    binary mask containing only the body
    //faceMaskFull: binary mask the size of the original mask, but only
                    containing the face
    //faceMask:    binary mask containing only the face
    IplImage* bodyMask, *faceMaskFull, *faceMask;

7
    //Construct bodyMask image:
    cvSetImageROI(mask, track->boundingBox);
    bodyMask = cvCreateImage(cvSize(track->boundingBox.width, track->
        boundingBox.height), IPL_DEPTH_8U, 1);
    cvCopyImage(mask, bodyMask);
12
    cvResetImageROI(mask);

    //Find headbox:
    CvRect headinabox = headext->findHead(bodyMask, track->centroid.x-track
        ->boundingBox.x);
    cvReleaseImage(&bodyMask);

17
    //Create a mask the size of the image, but only containing the head:
    faceMaskFull = cvCreateImage(cvGetSize(mask), IPL_DEPTH_8U, 1);
    cvZero(faceMaskFull);

22
    //Create a mask the size of the head:
    faceMask = cvCreateImage(cvSize(headinabox.width, headinabox.height),
        IPL_DEPTH_8U, 1);

    cvSetImageROI(mask, headinabox);
    cvSetImageROI(faceMaskFull, headinabox);

27
    cvCopy(mask, faceMaskFull);
    cvCopy(mask, faceMask);

    //Make the entire image the ROI again
32
    cvResetImageROI(mask);
    cvResetImageROI(faceMaskFull);

    /*****
    APPLY SOME METHOD FOR DETERMINING THE ORIENTATION HERE!
37
    Use faceMask or faceMaskFull together with the original frame.
    *****/

```

```
//Free the images:  
cvReleaseImage(&faceMaskFull);  
42 cvReleaseImage(&faceMask);  
}
```

---

**Listing A.3:** The findDirection function.

## APPENDIX B

### Codebook Parameters

This appendix lists the parameters used by the figure-ground segmentation module. The parameters are divided into parameters used for training and parameters used for online classification.

#### Training Parameters

- $\varepsilon_1$  : Threshold that determines the radius of the codeword. It is the maximum allowed chromaticity difference. Value = [8.0 ; 15.0].
- $\alpha$  : Used along with  $\beta$  to determine the height of the codeword based on  $\check{I}$  and  $\hat{I}$ . Value = [0.4 ; 0.7].
- $\beta$  : Used along with  $\alpha$  to determine the height of the codeword based on  $\check{I}$  and  $\hat{I}$ . Value = [1.1 ; 1.5].
- $N_{train}$  : The number of frames used to train the background model. Value = [500 ; 1000].
- $T_\lambda$  : The threshold for Maximum Negative Run-Length ( $\lambda$ ) during temporal filtering. Codewords with  $\lambda$  values above this threshold are removed from the codebook. Value =  $0.5 \cdot N_{train}$ .

## Online Classification Parameters

- $\varepsilon_2$  : Threshold that determines the radius of the codeword. It is the maximum allowed chromaticity difference. Value = [8.0 ; 15.0].
- $\alpha$  : Used along with  $\beta$  to determine the height of the codeword based on  $\check{I}$  and  $\hat{I}$ . Value = [0.4 ; 0.7].
- $\beta$  : Used along with  $\alpha$  to determine the height of the codeword based on  $\check{I}$  and  $\hat{I}$ . Value = [1.1 ; 1.5].
- $\gamma$  : The learning used by the adaptive filter to update the activated codewords. Value = [0.005 ; 0.020].
- $N_{stable}$  : The number of consecutive frames a pixel must be classified as background in order to be considered as stable background. Only stable background is updated during online classification. Value = [5 ; 15].
- $N_{train, online}$  : The number of frames that a codeword must undergo online training before it can be considered a change to a nonpermanent codeword. Value = [500 ; 1000].
- $T_{\lambda, online}$  : The online MNRL threshold. The  $\lambda$  of the training codeword must be below this threshold before it can be changed to a nonpermanent codeword. Value =  $0.5 \cdot N_{train, online}$ .
- $N_{expiration}$  : The maximum allowed number of frames between activation of a nonpermanent codeword. If more frames occur between activation, the codeword is removed from the codebook. Value = 5000.
- $n_{median}$  : The size of the median filter used in post-processing. Value = [3, 5].

## APPENDIX C

### Tracking Parameters

This appendix lists the parameters used by the tracking module. The parameters are divided into parameters used for merging, parameters for tracking and parameters used for object classification.

#### Merging Parameters

$T_{overlap}$  : The minimum allowed overlap on the  $x$  axis between two blobs for the two blobs to be merged. The threshold is measured in percent. Value = 20.

$T_{area}$  : The minimum area or size of a blob. If the blob is below this threshold it is not merged with any other blobs. Value = 20.

#### Tracking Parameters

$T_{bbcenter}$  : The maximum allowed distance between the blob bounding box center and the track bounding box center for a blob to be assigned to a track. Value = 50.

$N_{timeout}$  : The number of frames that a track can pass between activation before a track is saved to disk and removed from the track database. Value = 50.

$N_{activation}$  : The number of times that a track must be activated, in order to be considered as describing human movement. Value = 20.

## Object Classification Parameters

$A_{max}$  : The maximum allowed collective area of the blobs that comprise an object for it to be classified as a human. Value = 5000.

$A_{min}$  : The minimum allowed collective area of the blobs that comprise an object for it to be classified as a human. Value = 400.

$AR_{max}$  : The maximum allowed aspect ratio ( $\frac{width}{height}$ ) for blobs to be classified as humans. Value = 0.8.

$AR_{min}$  : The minimum allowed aspect ratio ( $\frac{width}{height}$ ) for blobs to be classified as humans. Value = 0.2.