Face detection architecture suitable for micropower integrated vision chip

Sophie Schönenberger Julius Gauckler

ETH Zürich Semester project Summer term 2004 with Tobi Delbruck, Institute of Neuroinformatics

Contents

1 Introduction **1**
1.1 Inspiration and Goal of the Work . . . 1

2 Theory **2**
2.1 Architecture . . . 2

3 Experimental Setup **3**
3.1 The Bear . . . 3
3.2 Test Video . . . 4
3.3 Signal for Interaction with the Bear . . . 4

4 Concept **5**
4.1 Correlation to a Mask . . . 5
4.2 Sum over the Rows and Columns . . . 6

5 Program **7**
5.1 How to use the Program . . . 7
5.2 Master.m . . . 7
5.2.1 Matlab Code of Master.m . . . 7
5.3 Source.m . . . 9
5.3.1 Code of Source.m . . . 9
5.4 Converter.m . . . 10
5.4.1 Matlab Code of Converter.m . . . 10
5.5 Masking.m . . . 11
5.5.1 Matlab Code of Masking.m . . . 12
5.6 Moviemaker.m . . . 12
5.6.1 Matlab Code of Moviemarker.m . . . 12
5.7 Timederivative.m . . . 13
5.7.1 Matlab Code of Timederivative.m . . . 13
5.8 Correlation1.m . . . 13
5.8.1 Matlab Code of Correlation1.m . . . 13
5.9 Correlation2.m . . . 14
5.9.1 Matlab Code of Correlation2.m . . . 14
5.10 Threshold.m . . . 14
5.10.1 Matlab Code of Threshold.m . . . 15
5.11 Illustrater.m . . . 15
5.11.1 Matlab Code of Illustrater.m . . . 15

6 Final Results **17**

Abstract

During the summer term 2004 we tried to search for a very simple model of face detection. Our approach was inspired by the receptor field of the color-selective ganglion cells in the retina. The setup with a camera in a doll simplified the task to detection of faces with a certain size and orientation. The goal of this work was to verify if this simple model can reach a satisfying precision of detection. The algorithm we developed wasn't very precise and couldn't recognize faces in all circumstances. However, by setting a threshold parameter, we could adjust the ratio of false positive to false negative responses over a wide range. Typically in an interactive environment users could usually get the doll to recognize them and false recognition rates were at the same time minimized.

Chapter 1

Introduction

We had the opportunity to make this project between March and July 2004 at the Institute of Neuroinformatics at the University Zurich and the Swiss Federal Institute of Technology Zurich, supervised by Tobi Delbruck.

1.1 Inspiration and Goal of the Work

At the beginning we had been inspired by Tobi Delbruck to find a simple but precise algorithm for face detection. He was thinking of a special dolly for his little daughter. The intention was the development of an interactive doll which can respond and react when a child plays with it. Consequently we should find a algorithm which could be implemented on a Chip.
To study this problem we mounted a USB camera (Logitech Quickcam, 320X240 at 15fps with 24 bit color) in the head of a toy bear, bought at a used items store. To get as many inputs as possible we took our bear on the street to let both, children and adults play with it.
After data capturing we started to develop the program in Matlab.

We tried two approaches with different success. Both will be named and explained in the following chapters.

Chapter 2

Theory

2.1 Architecture

An imaging chip consists of a 2-dimensional matrix of light sensitive elements - the so called pixels. Photons cause a change of charge in the pixels. After the integration time the pixel charge is transferred to a read out buffer. Based on the content of this buffer, the output unit generates the video signal.

The idea of our approach to face detection is to find an algorithm which can be implemented on the chip itself. When the charge is transferred to the buffer in each pixel it is multiplied by either 1 or -1 corresponding to large value of correlation. This correlation is interpreted in terms of probability whether there is a face or not.

Chapter 3

Experimental Setup

3.1 The Bear

Because of the part of the experiment we had to do on the street, we were looking for teddy. We opened his head to install a web camera in his head with the lens of the camera in his nose. The nose seemed to be the best place for the camera as the bear would still look nice and because of the symmetry.

Figure 3.1: Picture of our bear

Why a teddy bear and not any webcam?
First of all it's much easier to get a child play with a bear than with a simple web cam. Further the reason was the symmetry. Because of the length of the arms of the child the distance from the teddy to the head of the child stand in relation. The child's face would have a good size and orientation on the input signal. To get some data for testing the program we visited the kindergarten of the University of Zurich at the Irchel. Some of the video clips were taken at the ETH. With a laptop connected to our teddy bear we tried to get as many scenes as possible with different illumination and angles of view.

3.2 Test Video

To get test video we cut all our clips. With the function *moviemaker.m* single frames are marked with two little squares in one corner to check the quality of the detection later on. It is very difficult to decide which of the 1200 frames has enough of a face on it. This markings are subjective. The value of the calculated detection quality also depends on the way the frames had been selected.

3.3 Signal for Interaction with the Bear

To run the program real time we wanted to have an interaction between the bear and the person playing with the bear. The program should give a feedback when a face is detected so a threshold for the correlation with the mask is set. As the value gets higher than the threshold the program plays a sound. The one playing with the bear can try to place his face in front of the bear until it gives the signal that a face is detected.

Figure 3.2: Picture of a face captured by the webcam

Chapter 4

Concept

To get a observable for whether there is a face or not we need a mask of a face shape. With this mask the correlation of the input can be computed. As the correltion is an integer we have to set a threshold.

4.1 Correlation to a Mask

To compute the correlation we tried different types of masks. The input signal can come either from a testing video or can be captured real time by the camera. This signals get then compared with mask.
The input image is converted to the same size and color space as the mask image. The information is reduced to one channel and represented by matrices of the same size. The matrix of the image has entries in then range of 0 to 255.To get the entries of the mask we set the dark pixels to -1 and the bright ones to +1. Now the matrix of the mask is multiplied element-wise with the matrix of the input image. The sum over all pixels gives us a value for the correlation.
To visualize the result we plot the correlation over the length of the testing video or a predefined time. To filter out the most significant information for face detection we took only the color channel with the strongest contrast between face and background. So we tried different channels of the color spaces which are supported by Matlab. We gained the best results when we used the red channel in the RGB space or the Br channel in the YCbCr space. This is because the most significant difference between the background and the face is the high red value according to the color of the skin. We decided to use the Br channel of the YCbCr Space. The YCbCr color space is widely used for digital video. In this format, luminance information is stored as a single component (Y), and chrominance information is stored as two color-difference components (Cb and Cr).
Cb represents the difference between the blue component and a reference value. **Cr** represents the difference between the red component and a reference value. YCbCr data can be double precision, but the color space is particularly well suited to uint8 data. For uint8 images, the data range for Y is [16, 235], and the range for Cb and Cr is [16, 240].
To have a less sensitive result on small details, we decided to reduce the resolution of the image from 160×120 to 80×60 . Thus the image was blurred and structures like eyes mouth and teeth got lost. Due to movement of the head those were not good markers anyway.
As the quality of detection was unsatisfying we tried to modify the mask. Beside the areas with +1 and -1 a small area of zeros should make the correlation more shift invariant. Therefore we calculated local value of derivative of intensity in x and y direction. We toke the difference of the mask and a copy of it shifted by one pixel. As this did not help at all we cut it out again.
A hard light from the right side can give a bright right part of the face with the left part of the face laying in the shadow. As a hard light from the left side may cause just the negative image - a bright left part of the face with a dark right part- one idea was to split the image into several squares for which the correlation is calculated beside with independent masks for each square. Regarding the sum of the absolute values of the calculated correlations it would be more sensitive on shape rather than bad illumination. As the the YCbCr channel is hardly effected by the kind of illumination the feature to split the image was not included.
We tried to use the time derivative to get more information of the face. This approach led to very interesting results when we converted the input signal so that fast changing pixel were given as bright values. The eye were two well positioned bright spots when the person in the input images was moving the face a little and especially when the person blinked with the eye. The mouth was also notable when the person was talking. Unfortunately the quality of the signal depended very on the speed some one was moving and the frame rate. Also to much noise from the background when the bear was moved or the background had a lot of contrast made it difficult

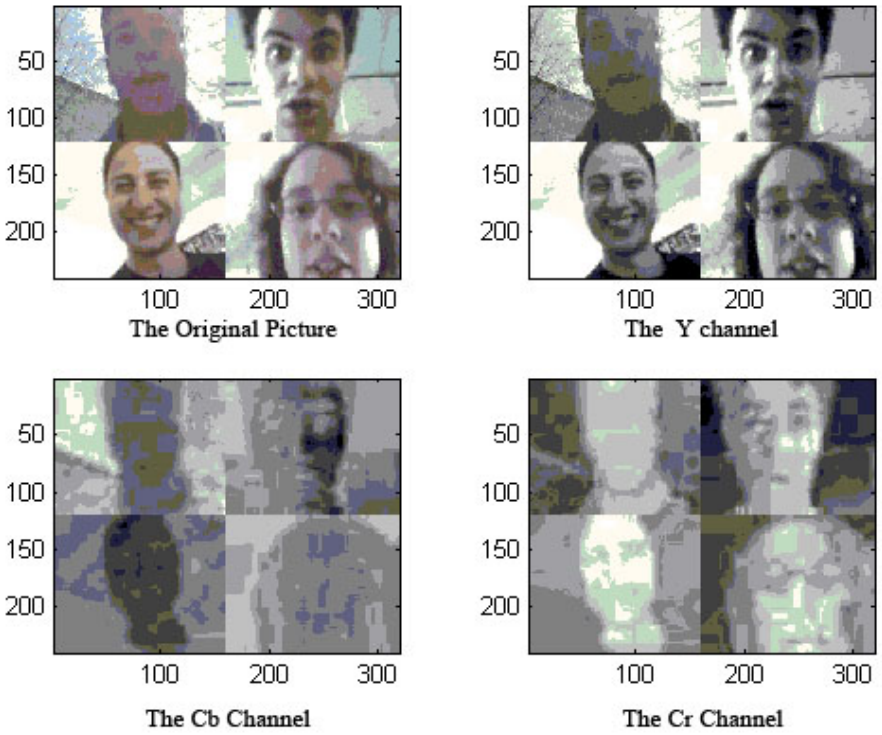

Figure 4.1: The different Channels of the YCbCr Space

to get useful information out of the time derivative image. With a different approach which uses more masks or which moves a mask over a region it might have been successful to detect a characteristic distance between the bright spots of the eyes.

4.2 Sum over the Rows and Columns

In our second attempt we reduced the input information to sums over the rows and the columns. All the intensity values of each row in the image were summed. As result we got a vector with the size of the number of columns. As well we summed each columns to get a vector the size of the number of rows. We compared the plot of the vectors with the according image but in the end we could not find any significant characteristics in the plot of the resulting vectors. So we stopped research on this method.

Chapter 5

Program

We wrote the program in Matlab. Though we had hardly any experience in programming before we were able to realize our ideas in this easy environment. We decided to split the program into small functions which are called by the *Master.m*. This way both of us were able to make changes in the source code to improve the program further the program *Master.m* became relatively short and easy to understand. Changes like to convert the input signal to a color space can be changed very quick which was important when we were trying out different settings.

5.1 How to use the Program

For changing the settings of the program open the file *master.m*. The important values like size, mode of correlation and use color space are defined in the first paragraph. To run the program real time you need a connected camera which can be detected by the Vcapg2.dll Mex-file. Start the program by running "master.m" (Just type "master" to the console when Matlab is in the appropriate directory.)

5.2 Master.m

Master.m is the main part of our program.
First the settings can be defined like color space and image size. Depending on the settings a image is loaded or a picture is taken by the camera. It gets then converted to the selected channel of the color space and a mask is rendered.
After this the following steps are repeated in a loop as many times as said in the settings or given by the length of the testing video. If the program is running with the testing video a frame is picked and it's checked if it's marked or not. In the real time mode an image is taken by the input image with *Converter.m*. The correlation to the mask is calculated either by *Correlation*1*.m* or by *Correlation*2*.m*. Then the input signal, the mask and the correlation are visualized with the function *Illustrater.m*.In the real time mode a wav file is played as a acoustical signal for face detection. After ending the loop the quality of face detection is shown and the camera is turned of.

5.2.1 Matlab Code of Master.m

```
Videosignal=1;
Mask_form_image=1;
Correlation_mode=1;
Colour_space='YCbCr';
Colour_channel=3;
Image_size=[80 60];

if(Videosignal==1) % for playing real time
     clear vcapg2;
     Videosignal=vcapg2; %initializing Vidokamera, get number of suported USB cams
     Wavbear=wavread('bear.wav');
```

```
     I=150; % Number of calculated frames - determins how long the program will run
end

Correlation_vector=0;
Truth_vector=0;
Respond_vector=0;

if(Videosignal==0) %for playing the test video
     Mov=aviread('Face2marked.avi');
     I=size(Mov);
     I=I(2);

end

% get an image to make a mask

if(Mask_form_image)
     Picture_Mask=imread('mask.bmp');
     Picture_Mask=imresize(Picture_Mask,Image_size);
else
     pause(3);
     Picture_Mask=vcapg2;
     Picture_Mask = imresize(Picture_Mask,Image_size);
end


% convert to a Colour space and pick a channel

Picture_Mask_grayscale=Converter(Picture_Mask,Colour_space,Colour_channel);

%render the mask

Mask=Masking(Picture_Mask_grayscale);


for(i=1:I)

%get frame from the cam or avi object

     if(Videosignal==0)
          Picture=Source(Mov,i);

%check if the frame is marked in the left upper corner.

          if((Picture(3,3,1)<10)*(Picture(8,8,1)<10))
               FaceMarker=1;
          else
               FaceMarker=0;
          end
          Truth_vector=[Truth_vector FaceMarker];
          Picture = imresize(Picture,Image_size);
          else
          Picture=vcapg2;
          Picture = imresize(Picture,Image_size);
     end

Picture_grayscale=Converter(Picture,Colour_space,Colour_channel);
```

```
if(Correlation_mode)

     Correlation=Correlation1(Mask,Picture_grayscale);
     Correlation_vector=[Correlation_vector,Correlation];
     Picture_Mask_Matrix=uint8((Mask+ones(size(Mask))));
     Pictures= zeros([Image_size,2]);
     Pictures(:,:,1)=Picture_grayscale;
     Pictures(:,:,2)=Picture_Mask_Matrix;

     if(Videosignal==1)
          if(Correlation>65000)
               soundsc(Wavbear,44100)
               pause(2)
          end
     end

%shows the correlation and the actual Fame

     Illustrator(Correlation_vector,Pictures);

else

     Correlation=Correlation2(Picture_grayscale);
     Pictures=Picture_grayscale;
     Illustrator(Correlation,Pictures);

%shows the sum vectors of rows and columns

end

% adjusting the speed

pause(1/100)

end
% turn off the Cam

clear vcapg2;
if(Videosignal*Correlation_mode==0)

% get the Quality by comparing the correlation with the markers in the test video

     Respond_vector=Threshold(Truth_vector,Correlation_vector);
end
```

5.3 Source.m

The program *Source.m* returns a single frame from an avi object in a uint8 rgb image format.

5.3.1 Code of Source.m

```
function [Picture ] = Source( Movie,i )

%Source Summary of this function goes here
```

```
if (size(Movie)== [1 1])
     Picture=vcapg2(0);
else
     frame=Movie(i);
     Picture = frame2im(frame);
end
```

5.4 Converter.m

Converter.m converts the input into the different colour spaces of Matlab. The property of all those spaces was discribed in chapter 2.3.

5.4.1 Matlab Code of Converter.m

```
function [Mono_Uint8] = ConvertViewer( RGB_Uint8, mode,channel )

% CONVERTVIEWER mode=rgb, HSV, NTSC, YCbCr channel= 1,2,3
% RGB Channels takes the red green or blue channel

if(strcmp(mode,'RGB'))
     if(channel==1)
          Mono_Uint8=RGB_Uint8(:,:,1);
     end
     if(channel==2)
          Mono_Uint8=RGB_Uint8(:,:,2);
     end
     if(channel==3)
          Mono_Uint8=RGB_Uint8(:,:,3);
     end
end

% ----------------
% HSV

if(strcmp(mode,'HSV'))
     HSV_Array=rgb2hsv(RGB_Uint8);

% convert RGB picture to HSV colorspace, picks channel 1 2 or 3 (hue, saturation, value)

     if(channel==1)
          Mono_Array=HSV_Array(:,:,1);
     end
     if(channel==2)
          Mono_Array=HSV_Array(:,:,2);
     end
     if(channel==3)
          Mono_Array=HSV_Array(:,:,3);
     end

% Normalize matrix to values in [0,255] bevor converting back to a Uint8

     Mono_Array=-min(min(Mono_Array))*ones(size(Mono_Array))+Mono_Array;
     Mono_Array=Mono_Array/max(max(Mono_Array))*255;
     Mono_Uint8=uint8(Mono_Array);
end
```

```
% -------------------
% NTSC: luminance(Y) & chrominance (I and Q)

if(strcmp(mode,'NTSC'))
     YIQ_Array = rgb2ntsc(RGB_Uint8);
     if(channel==1)
          Mono_Array=YIQ_Array(:,:,1);
     end
     if(channel==2)
          Mono_Array=YIQ_Array(:,:,2);
     end
     if(channel==3)
          Mono_Array=YIQ_Array(:,:,3);
     end

% Normalize matrix to values in [0,255] bevor converting back to a Uint8

     Mono_Array=-min(min(Mono_Array))*ones(size(Mono_Array))+Mono_Array;
     Mono_Array=Mono_Array/max(max(Mono_Array))*255;
     Mono_Uint8=uint8(Mono_Array);
end

% -------------------
% YCbCr luminance (Y) and chrominance (Cb and Cr)

if(strcmp(mode,'YCbCr'))
     YCbCr_Array=rgb2ycbcr(RGB_Uint8);
     YCbCr_Array=double(YCbCr_Array);
     if(channel==1)
          Mono_Array=YCbCr_Array(:,:,1);
     end
     if(channel==2)
          Mono_Array=YCbCr_Array(:,:,2);
     end
     if(channel==3)
          Mono_Array=YCbCr_Array(:,:,3);
     end

% Normalize matrix to values in [0,255] bevor converting back to a Uint8

     Mono_Array=-min(min(Mono_Array))*ones(size(Mono_Array))+Mono_Array;
     Mono_Array=Mono_Array/max(max(Mono_Array))*255;
     Mono_Uint8=uint8(Mono_Array);
end
```

5.5 Masking.m

The *Masking.m* returns the kernel rendered from a one channel image. The entries of the kernel matrix are +1 for pixels with a higher value as the median and -1 for pixels below the median. By taking the median of all pixels the positive area is exactly as large as the negative. Thus an unicoloured image or a random pattern will lead to zero respond in average. Also the returned kernel is hardly affected by irritating light sources in the background.
In other words the *Masking.m* makes a mask of the picture with the values (-1,0,1).

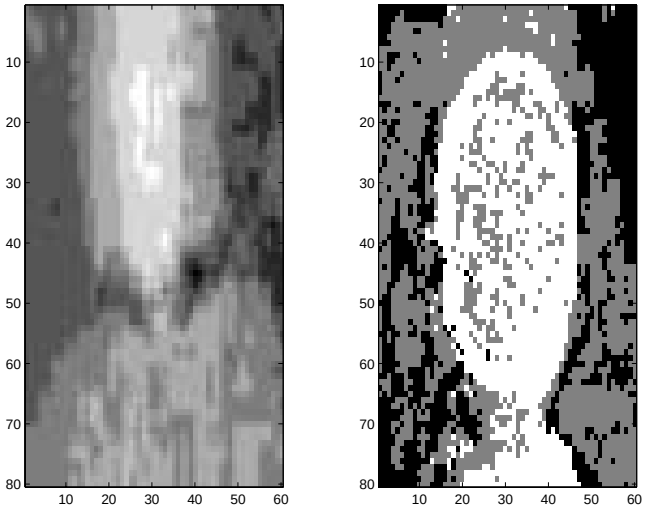

Figure 5.1: Picture and mask

5.5.1 Matlab Code of Masking.m

```
function [ kernal ] = Masking(picturemask)

mask=double(picturemask);
s=size(mask);
m=median(mask(:));

kernal=zeros(size(mask));
kernal(find(mask>m))=1;
kernal(find(mask<m))=-1;
```

5.6 Moviemaker.m

For testing different settings a movie was cut with many different faces in it. With algorithms can be tested under the same conditions. The program moviemarker.m is for marking the frames we expect the program to detect. First it opens an avi file and plays it frame by frame. For each frame one can decide whether the image shows a good visible faces or not. The result is written as a new avi file where all the good frames are marked with two black squares in the left upper corner. When the testing video is used it is simple to check whether there is a face in the actual frame or not by checking the value of a few pixels.

5.6.1 Matlab Code of Moviemarker.m

```
function [mov_out ] = Moviemarker( mov_in )
mov_out=avifile('markedMovie.avi');
S=size(mov_in);


for i=1:S(2)
     frame=frame2im(mov_in(i));
     image(frame)

     Eingabe=input('mark frame?  Enter for No');
     if isempty(Eingabe)
     frame=mov_in(i);
     mov_out = addframe(mov_out,frame);
```

```
    else      markframe=mov_in(i);
         markframe=frame2im(markframe);
         markframe=double(markframe);
         markframe(1:5,1:5,1:3)=zeros([5 5 3]);
         markframe(6:10,6:10,1:3)=ones([5 5 3]);

         markframe=uint8(markframe);
         markframe=im2frame(markframe);
         mov_out = addframe(mov_out,markframe);
    end
end

mov_out= close(mov_out);
```

5.7 Timederivative.m

The *Timederivative.m* makes, as you can guess because of his name, the timederivative of two pictures. It takes two pictures coming after each other and subtracts the second one from the first one. Because no negative numbers are accepted, you make the square of it. So you get *Picture*3 as a difference of the two.

5.7.1 Matlab Code of Timederivative.m

```
function [ picture3] = timederivative( picture1,picture2 )
% makes the timederivative of two pictures

matrix1=double(picture1);
matrix2=double(picture2);

% double converts picture1 and picture2 into matrix1 and matrix2

matrix3=a-b;
matrix3=sqrt(matrix3.*matrix3);
picture3=uint8(matrix3);

% uint8 converts the matrix3, which is the derivative picture of the

% picture1 and picture2 into a real picture, picture3, again
```

5.8 Correlation1.m

*Correlation*1.*m* computes the correlation between the kernel and the frame of video signal by taking the contrast between the positive weighted and negative weighted areas defined in the kernel. The matrix of the actual frame image is multiplied component wise with the kernel. The sum of all components in the resulting matrix gives a value for the correlation. The kernel consists of the same number of 1 as -1 therefore the negative components annul the positive components for frames without a face and the average be zero. But in case that the frame has a bright shape similar to the shape of the mask than the value rises. The range of possible values depends on the resolution of the frame.

5.8.1 Matlab Code of Correlation1.m

```
 function [ correlation ] = Correlation1( kernal, picture )

a=double(picture);
correlation=sum(sum(kernal.*a));
```

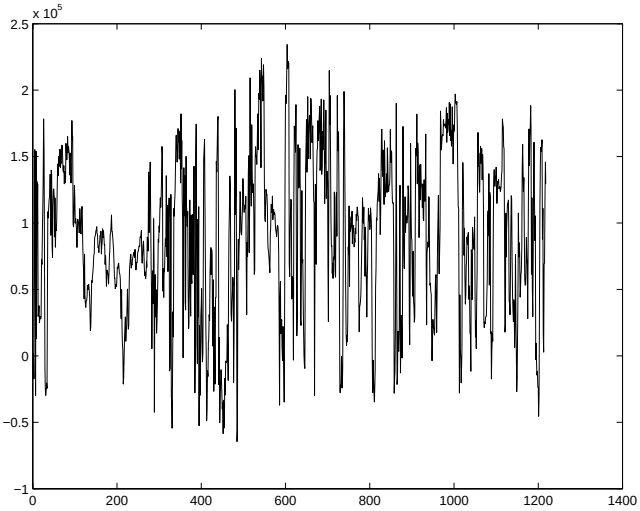

Figure 5.2: Correlation

5.9 Correlation2.m

Correlation2.m computes the correlation between the rows and columns. First the columns of the matrix of the actual frame image are treated as vectors and are summed up. The same happens with the transposed matrix to get the sum of all rows. To get only one matrix at end the vectors of the two sums are listed up in one matrix only by adding to the shorter one some zeros. A two column matrix is the result where each column can be plotted independently.

5.9.1 Matlab Code of Correlation2.m

```
function [m] = Correlation2(picture)

matrix=double(picture);

sumcolumns=sum(matrix);
sumrows=sum(matrix');

 [a,b]=size(sumcolumns);
 [c,d]=size(sumrows);

o=ones([1 d-b]);
s=[sumcolumns o];

m=[s ; sumrows];
```

5.10 Threshold.m

For comparing different algorithms of computing the correlation or testing which mask returns the best result an observable had to be found. The goal is to calculate an appropriate threshold which is adaptive to size and the brightness of the video signal and the mask. While playing the test video the vector *Truth_vector* records whether the frame is marked or not. A component of the vector is set to 1 if the corresponding frame is marked as a frame with a good visible face. Otherwise there is a 0 instead. Meanwhile the vector *Correlation_vector* records the correlation between the video signal and the mask. The program *Threshold.m* returns the positions of the frames with the highest correlation as the vector *Respond_vector*. Due to adaptation of the threshold the *Respond_vector* will have as many ones as the *Truth_vector* . Therefore the respond will contain as many wrong detected faces as wrong detected frames without a face. Further the percentages of the right detected

frame of faces and the right detected frames without a face are displayed. When running the program real time with the web cam from the bear a good threshold must be predefined in the *master.m*.

5.10.1 Matlab Code of Threshold.m

```
function [Respond_vector ] = Threshold( Truth_vector,Correlation_vector )

m=sum(Truth_vector);
SortedCorrelation=sort(Correlation_vector);
Schwellwert=SortedCorrelation(length(SortedCorrelation)-m);
Respond_vector=zeros(size(Truth_vector));
Respond_vector(find(Schwellwert<Correlation_vector))=1;

disp('The Threshold is set to')
disp(Correlation_vector(m))

disp(' Quality:  Right detected Faces in ')
disp((Respond_vector*Truth_vector')/sum(Truth_vector)*100)
disp(' Right detected Not-a-face in ')
Inverse_Respond_vector=(ones(size(Respond_vector))-Respond_vector);
Inverse_Truth_vector=(ones(size(Truth_vector))-Truth_vector);
disp((Inverse_Respond_vector*Inverse_Truth_vector')/sum(Inverse_Truth_vector)*100)
```

5.11 Illustrater.m

Illustrater.m plots the columns of the incoming matrix as a graph.

5.11.1 Matlab Code of Illustrater.m

```
function [] = Illustrator( curve_matrix , Bilder )
curve_matrix= curve _matrix';
s=size(curve_matrix);
if(size(s)>1)
     if(s(1)<s(2))
          curve_matrix=curve_matrix';
     end
end
clear s;

figure(1)
colormap(gray);
set(gcf,'doublebuffer','on');

s=size(curve_matrix);
for(i=1:s(2))
     subplot(s(2),1,i)
     plot(curve_matrix(:,i))
end

clear s;

figure(2)
colormap(gray);
set(gcf,'doublebuffer','on');
dimension=size(Bilder);
```

```
dimsize=size(dimension);
if(dimsize(2)>2)
     s=dimension(3);
else
     s=1;
end

for(i=1:s)
     subplot(1,s,i)
     imagesc(Bilder(:,:,i))
end
```

Chapter 6

Final Results

At the end of the semester we have found an algorithm which could detect faces. To make it nicer we let the bear make a noise each time he detected a face. Most of the time the face was detected properly even though red objects in the background could irritate it and led to wrong results. The time was to short to reach a better results. But we are happy with our work and hope someone else can continue it on the base we made.