Thesis for the Degree of Master of Science

Home Security System

Jean-Paul Kouma

Supervisors
Hung-Son Le, PhD student, Umeå university
Johannes Karlsson, PhD student, Umeå university

Applied Physics and Electronics
SE-901 87 Umeå, Sweden

Tillämpad Fysik och Elektronik
901 87 Umeå

Umeå, Sweden 2006

Abstract

This thesis report presents an investigation and implementation of a real-time security system based on facial feature recognition.

The system has a (certain number of) camera(s) mounted at some strategic point(s) in the house toward strategic point(s) (e.g. door, windows, etc.) in a house. If the system detects an intruder, his/her image is then sent to the user by MMS. The system recognizes the people living in the house and automatically get deactivated if one of these people enters the house and activated when all people leave the house.

As background knowledge, an intense literature review has been done, mostly in the area of facial feature processing, in the aim to select the best techniques.

For the implementation purpose, the programming languages MATLAB™ [1] and C were used. As complement to the standard AINSI C library, a computer vision C library called OpenCV[2] was also used. The hardware development was mostly microprocessor-based, using the 8-bit ATMEL™[3] microprocessor, ATMEGA 162 and the C programming tool WinAVR[4] and PonyProg[5].

Keyword: Alarm system, Face detection, Face recognition, Image processing.

Sammanfattning

Denna ex-jobbsrapport behandlar en undersökning och implementering av en realtidsbaserat säkerhetssystem som i stort sett bygger på ansiktigenkänning.

Systemet inehåller en videokamera (eller ett visst antal videokameror) som är monterad på en strategisk punkt i ett hus och som tittar mot en potentiell ingång (det kan vara dörr, fönster, etc.). Om systemet detekterar en okänd skall en bild på dess ansikte skickas till användaren(-rna) via MMS. Systemet skall dessutom vara kapabelt till känna igen alla som tillhör hushållet och automatiskt avaktivera sig om en av användarna kommer in i huset. Det skall också automatiskt kunna aktivera sig så fort alla lämnat huset.

MATLAB och C användes som programmeringsspråk. Som tillägg till AINSI C-biblioteket användes ”computer vision”- biblioteket OpenCV. Hårvaruimplementeringen var i stort sett mikroprocessorsbaserad med ATMEGA 162 som processor och WinAVR och PonyProg som programmeringsverktyg.

Resumé

Ce rapport de thèse est le résultat d' une étude et d' une implementation d' un système de sécurité en temps reel essentiellement basé sur la reconnaissance faciale.

Le système est equipée d' une (ou plusieurs) camera(s) de surveillance placée(s) a un (des) endroit(s) strategique(s) pointée(s) vers une entrée stratégique, dans une maison. Si le système détecte un inconnu, une image de sa face est automatiquement evoyée à l' utilisateur par MMS.

Par ailleurs le système est capable de reconnaître tous les membres de cette maison, et de ce fait, se désactive et respectivement s' active automatiquement dès la détection d' un d'eux et dès que le dernier membre est parti.

La partie logiciel du système fut implementée en utilisant les languages MATLAB et C. Ensuite, comme complément aux bibliotèques standards AINSI C, la bibliotèque de traitement d' images, OpenCV, fut utilisée. Le ”hardware”, essentiellement programmable, fut aussi implementée en C, avec le processeur ATMEGA 162 de ATMEL, en utilisant les outils de programmation WinAVR et PonyProg.

I would like to express my deepest gratitude to the following people

Johannes Karlsson and Hung-Son Le, Umeå univerity
for their precious guidance through this thesis work

Dr. Apostolos Georgakis and Ulrik Söderström, Umeå university
for their worthwhile lectures in the area of Computer Vision

Noor and Sani Anani
and everyone else in the Digital Media Lab
and all my frends
for enthusiasm, encouragements and more

Lena Walfridsson and her family
for their endless support and encouragement

My mother
for believing in me

I dedicate this work to Stina Walfridsson

Jean-Paul Kouma
Umeå university, June 2006

Table of Contents

1 Introduction

Biometrics have been one of the hottest areas of research during the past 15 years and a lot of works have been done and hundreds of techniques have been proposed. Fingerprint identification was the first major topic. With the advance of technology and science, one could shoe that other parts of the body, such as the eyes (iris), or the hand geometry, or even the face, also could be used as biometric means. This report describes therefore a complete Home Security System based on face recognition.

1.1 Literature review

For a face to be recognized, it has first to be located in a image. Thus this review covers both face detection and face recognition algorithms. Time was mostly spent on finding a good face detection technique, since the accuracy of a face recognition algorithm is strongly related to the reliability of the face detection algorithm.

There has been a good deal of research in the area of face recognition/detection and many algorithms have been proposed, especially in the last decade. Some of the major techniques are briefly presented here.

The earlier face detection algorithms were essentially based on skin color segmentation, using different color space.

Dr. Jianping Fan[6] based his face detection algorithm on skin color model followed by a binary skin region refinement technique, in aim to discard error introduced by the model. Next step is a region based adaptive threshold for facial feature segmentation. Then an orientation, scale and pose invariance strategy is used to extract the candidate face region.

Jay P. Kapur[7] also uses color information for face detection. The input image (which has to be an RGB image) is converted into log-opponent values (IRgBy), which has been proven to be more robust to illumination or camera resolution. Then face candidates are extracted using skin color information and threshold as well.

The disadvantages of both techniques described above are quite many. The major ones are the processing time and robustness. They are not fast enough for real-time application. They are not that robust to illumination and/or camera resolution. Due to those disadvantages several other techniques were developed, but instead based on feature information. Among them Support Vector Machine[8], (modified) Haussdorf Distance[9] and Adaptive Boosting[10].

The latter technique was the one used in our software implementation for face detection. It is can

process up to 15 frames per second on a conventional 700 MHz Pentium III computer and has a very high detection accuracy. The Adaptive boosting algorithm was developed by Paul Viola and Michael Jones. It is a machine learning approach for visual object detection. The detection process is based on the value of simple features. The simple features used are the Haar basis function. Three kinds of features are used as show in figure 1:

- Two-rectangle feature. It is the difference between the sum of the pixels within two rectangular regions
- Three-rectangle feature. It is the sum within two outside rectangles subtracted from the sum in the center rectangle.
- Four-rectangle feature. It is the difference between diagonal pair of rectangles.

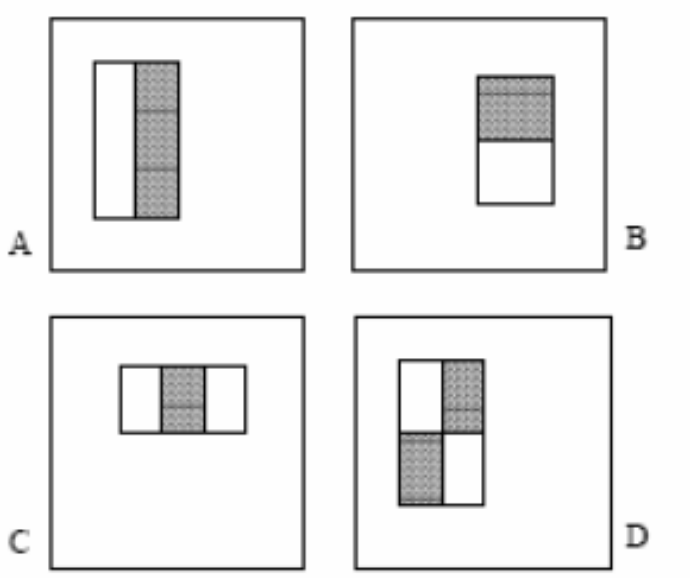


Figure 1: Rectangle features. A and B two-rectangle feature. C three-rectangle feature. D four rectangle-feature

The rectangle features can be computed very rapidly an intermediate representation of the image, a so-called **integral image**. It is given by:

$ii = \Sigma i(x', y')$, where i is the original image and *ii* the integral image. The integral image can be computed in one scan over the original image using the following recurrences:

$s(x, y) = s(x, y-1) + i(x, y)$

$ii(x, y) = ii(x-1, y) + s(x, y)$

where $s(x, y)$ is the cumulative row and $s(x, -1) = 0$ et $ii(-1, y) = 0$. See

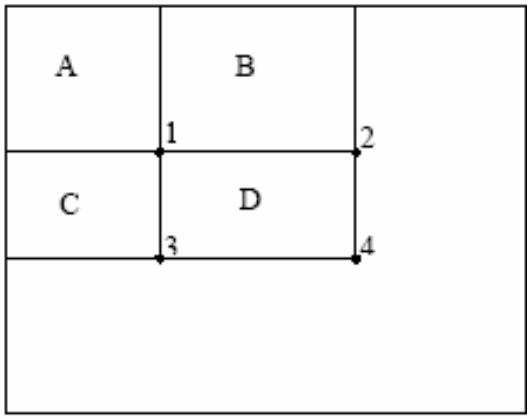


Figure 2:

In the figure 2 above, the value of the integral image at:

- Location 1 is the sum of the pixels in rectangle
- Location 2 is A+B
- Location 3 is A+C
- Location 4 is A+B+C+D

Thus the sum within D can be computed as 4+1-(2+3).
Given a feature set and a training set of positive and negative images, the Adaptive Boosting (AdaBoost) learning algorithm (adaptive boosting), sometime called the weak learning algorithm, is used to train the image set. This leaning algorithm select the rectangle feature which best separate the positive and the negative examples. Figure 3 shown below summarizes the boosting process

- Given example images $(x_1, y_1), \ldots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, \ldots, T$:
  1. Normalize the weights,
     $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$
     so that w_t is a probability distribution.
  2. For each feature, j, train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t, $\epsilon_j = \sum_i w_i \left| h_j(x_i) - y_i \right|$.
  3. Choose the classifier, h_t, with the lowest error ϵ_t.
  4. Update the weights:
     $$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$
     where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
- The final strong classifier is:
  $$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$
  where $\alpha_t = \log \frac{1}{\beta_t}$

Figure 3: Summarization of the boosting procees

Selecting a face recognition algorithm was not as difficult as the selection of the best face detection algorithm. The choice went almost without hesitation to Principal Component Analysis[11] or PCA. This technique is well known by the author, because previously used in some school projects in the area of computer vision. Beside it, PCA is a very common statistical method for pattern identification and data modeling/compression. To build eigenfaces following steps are done:

- Given a set S of M images of size $N \times N$, transform each image to a column vector Γ_i of size N^2 and put into the set. S = $\{\Gamma_1 \Gamma_2 \ldots \Gamma_M\}$

- Compute the mean image $\Psi = \frac{1}{M} \Sigma \Gamma_i$
- Subtract the mean from each image $\Phi_i = \Gamma_i - \Psi$
- Compute the covariance matrix $C = \frac{1}{M} \Sigma \Phi_i \Phi_i^{'}$
- Compute the eigenvectors of C. But C is very large, hence no practical. M. Turk and A. Pentland, developed a better way to compute the eigenvectors from the covariance matrix[12].
- Keep the L largest eigenvalued egeinvectors, in the decreasing order. Let call that set of vectors *u*. *u* is often called Eigen space or face space. L is chosen experimentally. One method is to choose L to be number of Eigen value that represents about 75% of the sum of all Eigen values.
- Project the normalized images to the face space to get the feature vectors $\Omega = \Psi^T u$

To recognize a new face image Ψ_{new}

- Convert it in to column vector and subtract the mean image Ψ from it. $\Phi_{new} = \Psi_{new} - \Psi$
- Project the new normalized column vector Φ_{new} on the Eigen space $\omega_{new} = \Phi_{new}^T u$
- Then to find the face that matches better compute distance between ω_{new} and each vector in the feature space Ω . The distance can be Euclidian, Mahalanobis or some other distance. We used Euclidian distance in our work. The new input face is recognized as face number *n* if the distance between ω_{new} and *n*:th vector in the feature space is the smallest and is under the predefined threshold. Otherwise it is not.

1.2 Outline of the report

The rest of this report is organized as follow:

- Proposed approach
- Interaction between hardware and software
- Experimental result
- Conclusion and discussion
- References
- Source codes (the most important)

2 Proposed approach

The system is composed of two main parts: software and hardware part. The rough idea behind is summarized as follow:

- Two sensors placed at the door, for sensing entrances/exits. Thus the system

is automatically activated or deactivated when it has to.

- A microcontroller digitalizes/encodes the signals sent by the sensors, since their outputs are analogue.
- A computer where the digitalized signals are sent, to get decoded as either "entrance" or "exit". Then the number of people in the house is updated. The computer also takes care of face recognition process.
- A surveillance camera that is switched on/off automatically by the computer, depending on the number of people left in the house.
- A GSM module for sending images of intruders to user's mobile phone

Figure 1. below shows the system flowchart

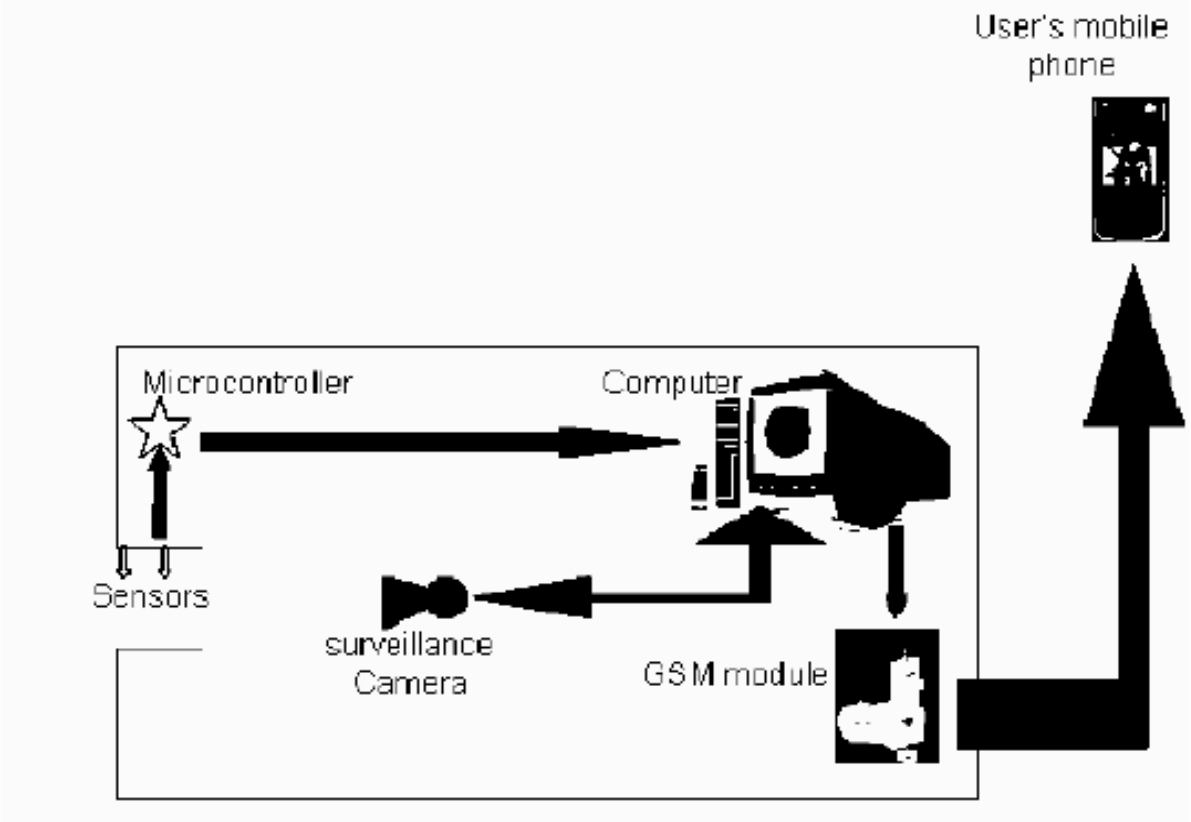


Figure 4: Flowchart of the system

2.1 Hardware implementation

2.1.1 The infrared sensors

Those sensors are "high" by default. It means until someone is sensed, the output voltage of the

sensors is unstable 9 volts. Thus when someone get sensed the output voltage goes “low” down to unstable 1 to 2 volts and back to “high” again, waiting for next sensing. The reaction time is a few micro second, therefore able to sense events “happening at the same time”.

2.1.2 The microcontroller

The microcontroller is composed of a microprocessor (ATMEL 162) and two “home-made regulators” composed of resistors and Zener-diodes. The regulators are connected between the output of the regulators and two pins of the microprocessors, which are configured as inputs as shown in figure 2.

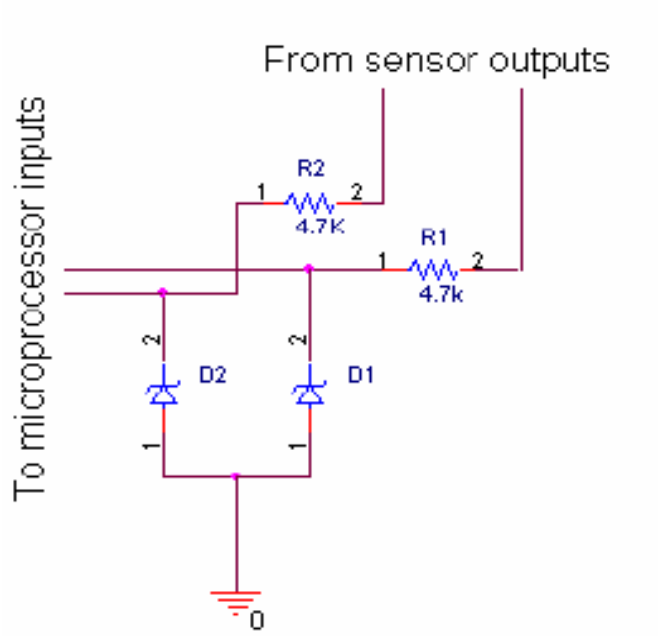


Figure 5: Home-made regulators

The aim of the “home-made” regulators – There are often called home-made because there do exist commercial regulators. The commercial ones are of type IC. The principle is although the same. – is to cut down, unstable 9V or 2V, to stable 5V and 0V, respectively. Recall that the analogue signals from the sensors have to be digitalized/encoded. Thus the output has to be either 5V or 0V, for “high” or “low” respectively, to be processed properly by the microprocessor. The microprocessor is programmed so that it waits for one of its connected pins to go “low”. Depending on which pin went low first, yields to an entrance or exit detection. Then if an entrance is detected the string “IN” is sent to the computer through it serial port (the microcontroller is serially connected to the computer), else if an exit is detected, the string “UT” is sent. Thus the number of people in the house is updated. See figure 3

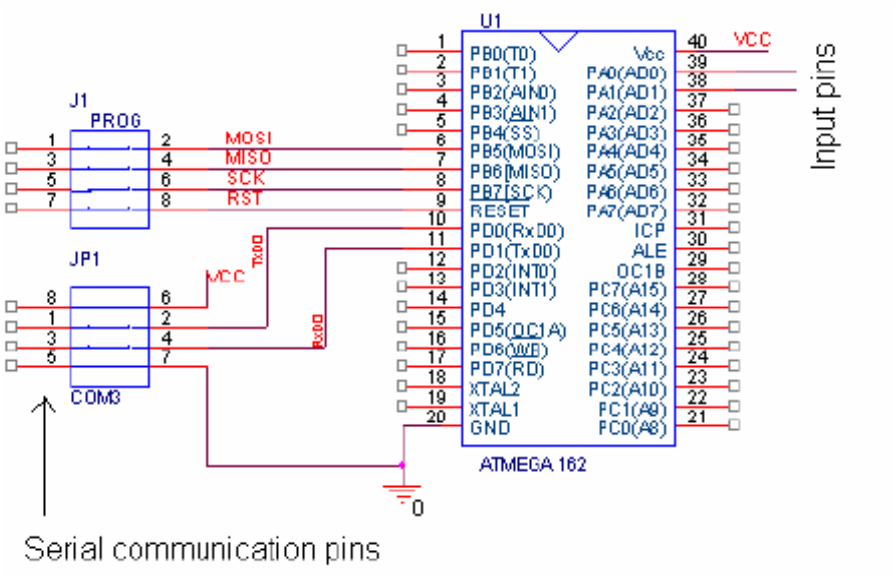


Figure 6: Microprocessor configuration

Figure 4 and 5 shows the whole hardware schematic and picture of the microcontroller, respectively.

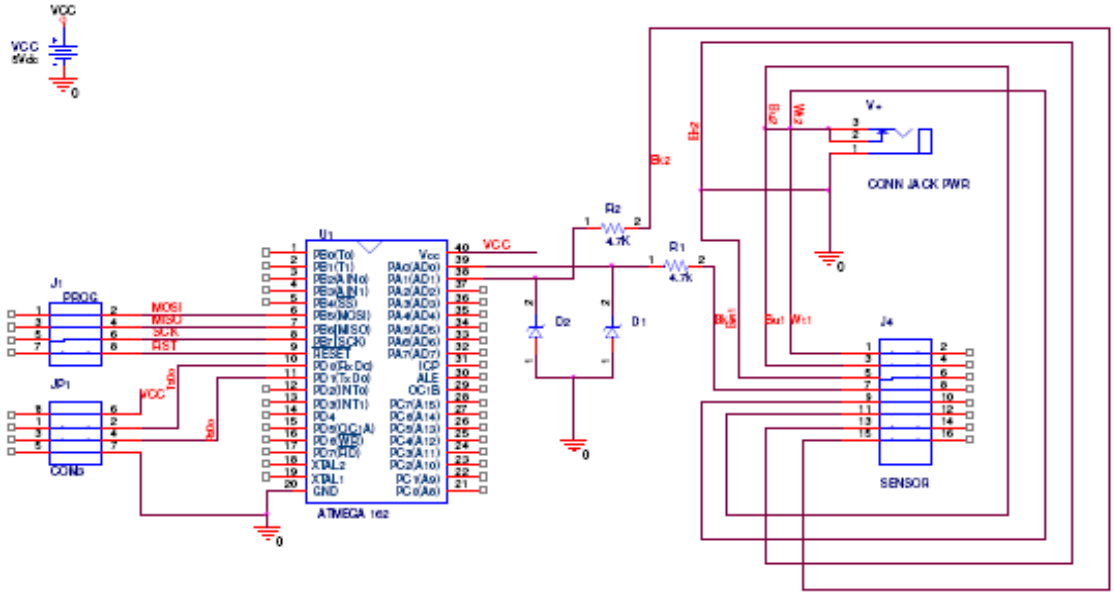


Figure 7: Hardware schematics

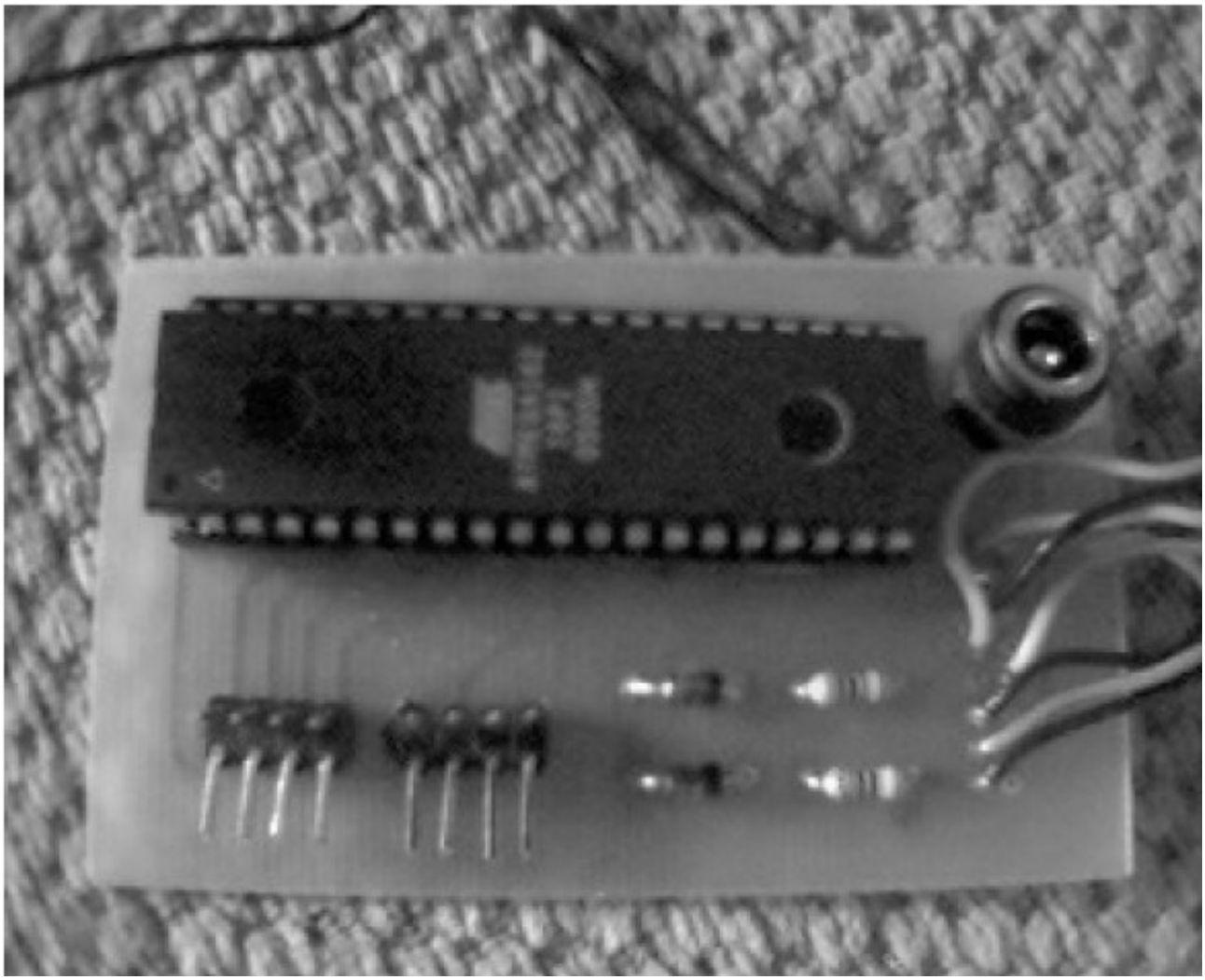


Figure 8: Microcontroller

2.2 Software implementation

The software is of type easy-to-use GUI and divided in two processes: an enrollment part and a recognition part.

2.2.1 The enrollment process

At each enrollment following steps are done:

- Sequences of images are grabbed from the camera toward the face of the candidate being enrolled.
- The candidate's faces are automatically detected and cropped out of the input images. Adaptive Boosting Algorithm is used for face detection.

• Six different relevant face poses are selected and stored
encoded/compressed using the PCA algorithm. The ensemble of the stored images will be the face database.

2.2.2 The recognition process

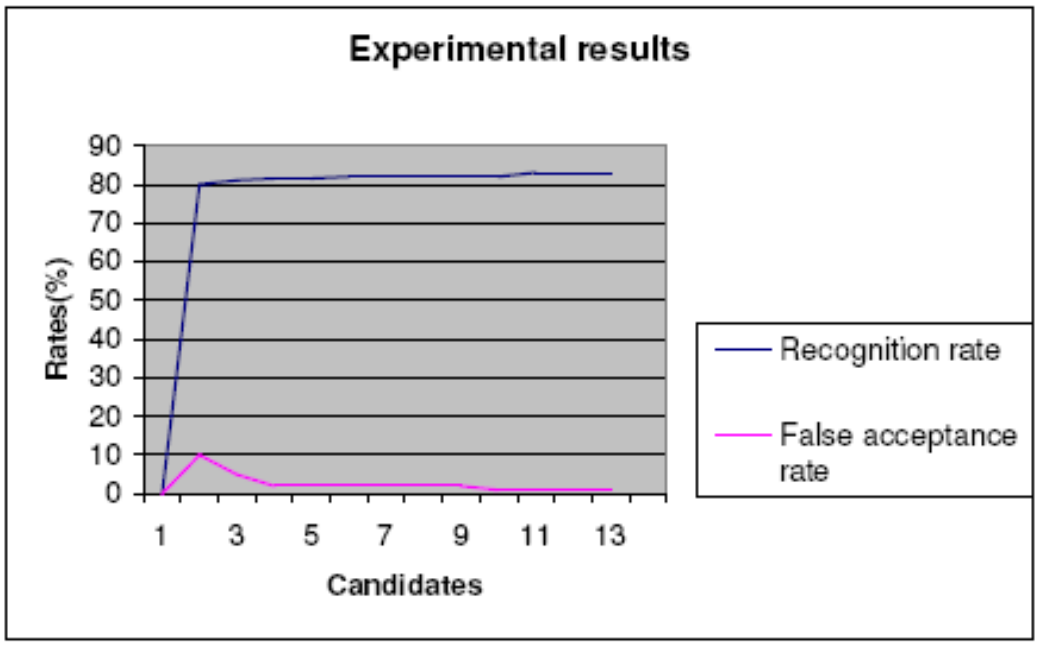
Recall that the recognition process takes place when the alarm is on and as soon as the sensors detect entrance. Then:

- The surveillance camera is switched on and images are captured
- Faces are automatically detected and cropped out the input images
- PCA is applied on each detected face.

If one of those faces are recognized, the candidate is a house member. Otherwise the candidate is an intruder. In that case his/her face is loaded on a web server and the link is sent to the user's mobile phone, by SMS. Due to the relatively bad resolution of the camera, the cropped images are enhanced – using histogram equalization, followed by a 3-by-3 average filter, followed by the Laplace sharpening filter – before they are projected in the face space.

3 Experimental results

The system was tested on 13 persons. Only accuracy of the face recognition was taken in account, since the reliability of the the whole system is strongly correlated to the performance of the face recognition algorithm. Six face images – two frontal, two left profile and two right profile – from each person were stored in the face database. The results of these experiments are shown in the graph below



Graph showing the experimental results

The experiments show that the more people in the database the more accurate the system got. The system works like worse when only one person is enrolled. The recognition rate is very low (under 10%), but the false acceptance remains also low, which is a good point, technically spoken. The performance of the system increases to over 80% when more people are stored in the data but has an asymptotic behavior in the neighborhood of 85%.

85% accuracy is not bad at all, when taking in account that the camera used here was a web camera with relatively bad resolution.

Despite the poor image resolution the device the detection algorithm remains highly accurate and surprisingly fast.

Finally it has been shown by experiment that the mean execution time of whole system, on a 2.4 GHz Pentium IV computer, was estimates to 3 second, not taken in account the time taken to send the MMS. The latter lies on other factors that are not discussed here.

Figures 6, 7, 8, below show some results of the system: successful recognition, intruder detection (hidden face) and false rejection respectively.

Figure 9: Successful recognition

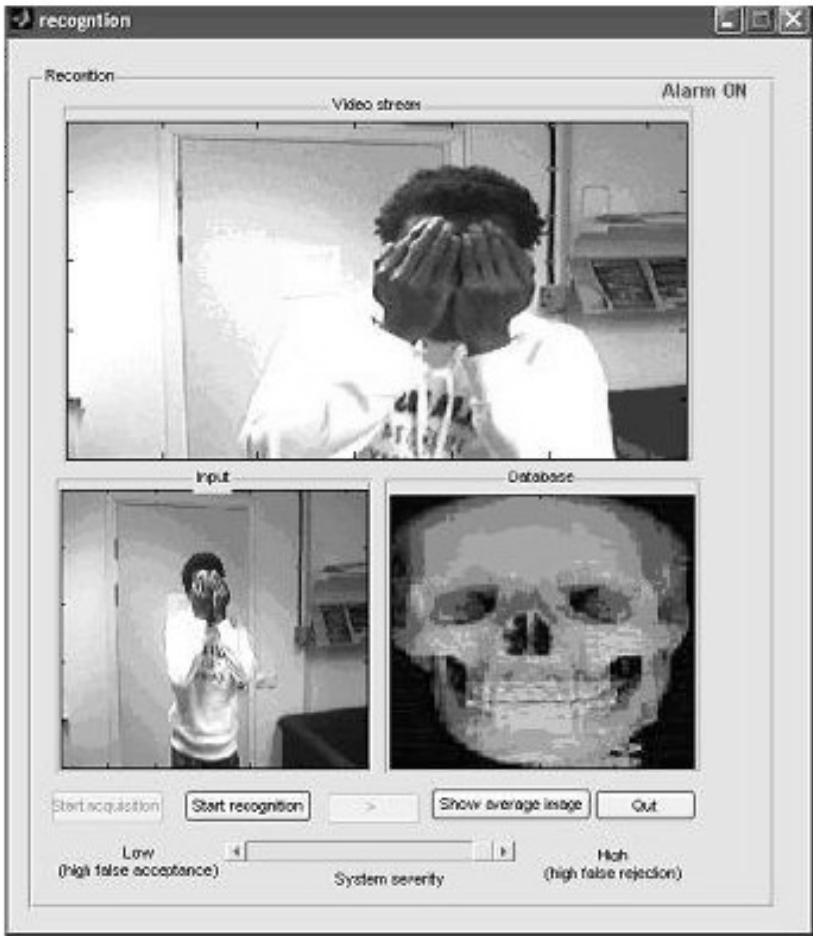


Figure 10: Intruder detection

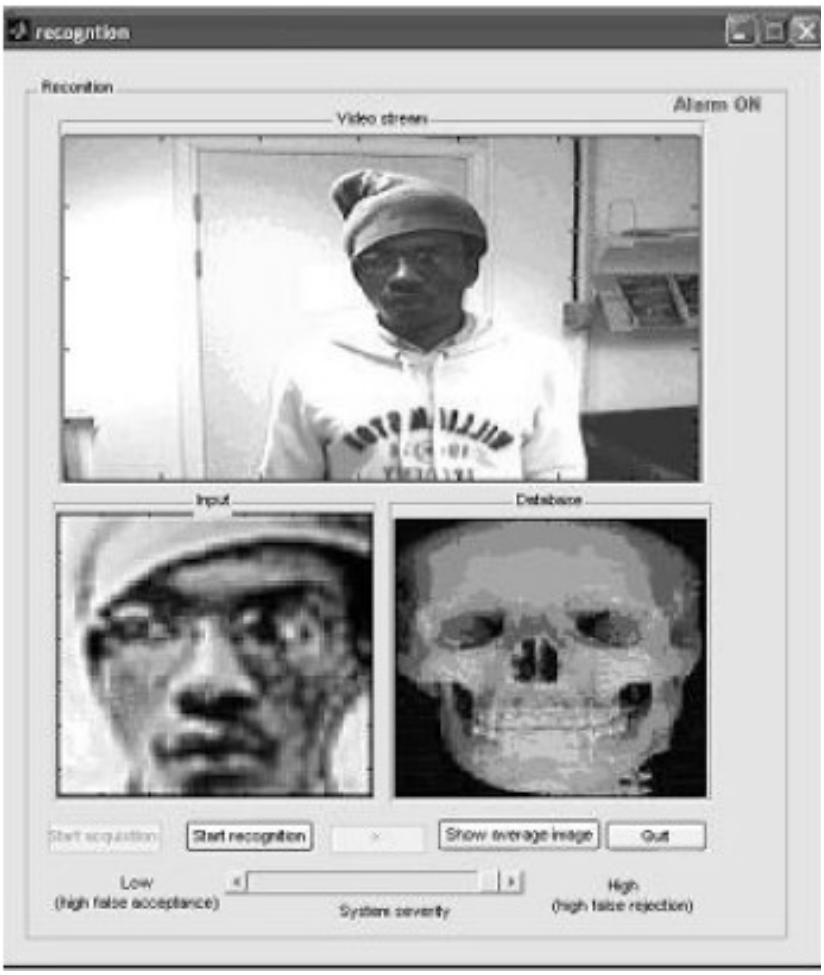


Figure 11: Fasle rejection

4 Conclusion and discussion

Despite the poor resolution of the camera (as you can see in the experiment part), the experiments have proven that the system can achieve high recognition rates. Its reliability and fastness make it suitable for real-time application, since the whole process – from sensing to recognition – takes up to a few seconds on a conventional Pentium IV, 2.4 GHz. The system can even run fine on a slower computer, i.e. Pentium III, 700 MHz. This alarm system can be seen as "the first on the market of this kind". There was a limitation in the project, when it comes to the GSM communication part. The GSM module should, as specified in the task, be able to send MMS to the user's mobile phone. That part got modified a little bit, since MMS programming is time demanding. Instead, the intruder's images are loaded on a server and the links are sent to the user as SMS.

The microprocessor used in the microcontroller was unnecessary powerful. A less powerful and cheaper microprocessors could has been used, i.e. an ATMEGA 8L, or an ATtiny 2313 and still give the same results. All those microprocessors are relatively easy to program, since they can be programmed in C (or C++) and are programmed using a low-cost In-System-Programmer, which makes software development very fast (I am not doing commercial, just telling the truth). Still there are some improvements to do (have you ever encounter bug-free system? If yes, let me know. Even the classical “hello world” is full of bugs. Check twice, you'll see some bug somewhere... Who am I talking with?). The most important could be making the microprocessor compute all entrances/exits, or make the computer process less images for recognition and still get high recognition rates. This will probably make the system run even faster, but on the other hand it will imply more programming, and you maybe know when it comes to programming and hardware...

5 References

[1] www.mathworks.com

[2] http://www.intel.com/technology/computing/opencv/

[3] http://www.atmel.com/

[4] http://winavr.sourceforge.net/

[5] http://www.lancos.com/prog.html

[6] Dr. Jianping Fan. Toward facial feature locating and verification for omni-face detection in video/images. http://www.cs.uncc.edu/~jfan/

[7] Jay P. Kapur. Face detection in color images. University of Washington Department of Electrical Engineering.1997

[8] C. Cortes and V. Vapnik. Support Vector network. Machine learning, vol 20, pp 1-25, 1995.

[9] Oliver Jesorsky, Klaus J. Kirchberg, and Robert W. Frischholz. Robust face detection using the Haussdorf distance. Third International Conference on Audio- and Video-based Biometric Person Authentication, Springer, Lecture Notes in Computer Science, LNCS-2091, pp. 90–95, Halmstad, Sweden, 2001.

[10] P. Viola and M. Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. Accepted conference on computer vision and pattern recognition, 2001

[11] Matthew A. Turk and Alex P. Pentland. Face recognition using Eigenfaces. Vision and Modeling Group , the Media Laboratory, Massachusetts Institute of Technology

[12] M. Turk and A. Pentland, "Eigenfaces for Recognition", Journal of Cognitive Neuroscience, vol.3, no. 1, pp. 71-86, 1991, hard copy

6 Appendix: Most relevant source codes

6.1 Face detection using the Adaptive Boosting algorithm (Using OpenCV library)

The following source code is a slightly modified version of the original sample code that came with the OpenCV library.

```
#include "cv.h"
#include "highgui.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <math.h>
#include <float.h>
#include <limits.h>
#include <time.h>
#include <ctype.h>

#ifdef _EiC
#define WIN32
#endif

static CvMemStorage* storage = 0;
static CvHaarClassifierCascade* cascade = 0;

void detect_and_draw( IplImage* image );

const char* cascade_name =
    "haarcascades/haarcascade_frontalface_alt.xml";
/*    "haarcascade_profileface.xml";*/

int main( int argc, char** argv )
{
    CvCapture* capture = 0;
    IplImage *frame, *frame_copy = 0;
    int optlen = strlen("--cascade=");
    const char* input_name;

    if( argc > 1 && strncmp( argv[1], "--cascade=", optlen ) == 0 )
    {
        cascade_name = argv[1] + optlen;
        input_name = argc > 2 ? argv[2] : 0;
    }
```

```
    else
    {
        cascade_name = "haarcascades/haarcascade_frontalface_alt2.xml";
        input_name = argc > 1 ? argv[1] : 0;
    }

    cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name, 0, 0, 0 );
      //printf("css = %s\n", cascade_name);

    if( !cascade )
    {
        fprintf( stderr, "ERROR: Could not load classifier cascade\n" );
        fprintf( stderr,
        "Usage: FaceDetect --cascade=\"<cascade_path>\" [filename|
camera_index]\n" );
        return -1;
    }
    storage = cvCreateMemStorage(0);

    if( !input_name || (isdigit(input_name[0]) && input_name[1] == '\0') )
        capture = cvCaptureFromCAM( !input_name ? 0 : input_name[0] - '0' );
    else
        capture = cvCaptureFromAVI( input_name );

    //cvNamedWindow( "result", 1 );

    if( capture )
    {
        for(;;)
        {
            if( !cvGrabFrame( capture ))
                break;
            frame = cvRetrieveFrame( capture );
            if( !frame )
                break;
            if( !frame_copy )
                frame_copy = cvCreateImage( cvSize(frame->width,frame->height),
                                            IPL_DEPTH_8U, frame->nChannels );
            if( frame->origin == IPL_ORIGIN_TL )
                cvCopy( frame, frame_copy, 0 );
            else
                cvFlip( frame, frame_copy, 0 );

            detect_and_draw( frame_copy );

            if( cvWaitKey( 10 ) >= 0 )
                break;
        }

        cvReleaseImage( &frame_copy );
        cvReleaseCapture( &capture );
    }
    else
    {
        const char* filename = input_name ? input_name : (char*)"lena.jpg";
        IplImage* image = cvLoadImage( filename, 1 );

        if( image )
```

```
        {
            detect_and_draw( image );
            cvWaitKey(0);
            cvReleaseImage( &image );
        }
        else
        {
            /* assume it is a text file containing the
               list of the image filenames to be processed - one per line */
            FILE* f = fopen( filename, "rt" );
            if( f )
            {
                char buf[1000+1];
                while( fgets( buf, 1000, f ) )
                {
                    int len = (int)strlen(buf);
                    while( len > 0 && isspace(buf[len-1]) )
                        len--;
                    buf[len] = '\0';
                    image = cvLoadImage( buf, 1 );
                    if( image )
                    {
                        detect_and_draw( image );
                        cvWaitKey(0);
                        cvReleaseImage( &image );
                    }
                }
                fclose(f);
            }
        }

    }

    return 0;
}


void detect_and_draw( IplImage* img )
{
      int scale = 1;
      IplImage* temp = cvCreateImage( cvSize(img->width/scale,img-
>height/scale), 8, 3 );
      CvPoint pt1, pt2;
      int i, cnt;
      char tmp_file[262];
      FILE *inp;

      //cvPyrDown( img, temp, CV_GAUSSIAN_5x5 );
      cnt = 1;
      cvClearMemStorage( storage );
      if( cascade )
      {
            CvSeq* faces = cvHaarDetectObjects( img, cascade, storage,
            1.1, 2,
            CV_HAAR_DO_CANNY_PRUNING,
            cvSize(40, 40) );
            for( i = 0; i < (faces ? faces->total : 0); i++ )
            {
```

```
                  CvRect* r = (CvRect*)cvGetSeqElem( faces, i );
                  pt1.x = r->x*scale;
                  pt2.x = (r->x+r->width)*scale;
                  pt1.y = r->y*scale;
                  pt2.y = (r->y+r->height)*scale;
                  cvGetSubRect(img, (CvMat *)temp, *r);
                  sprintf(tmp_file, "temp/%d.jpg", cnt);
                  inp = fopen(tmp_file, "r");
                  while(inp!=NULL){
                        fclose(inp);
                        cnt++;
                        sprintf(tmp_file, "temp/%d.jpg", cnt);
                        inp = fopen(tmp_file, "r");
                  }
                  //printf("Saved in %s\n", tmp_file);
                  cvSaveImage(tmp_file, temp );
                  cnt = 1;
            }
      }

      cvReleaseImage( &temp );
}
```

6.2 Image enhancement

```
function Ie = EnhanceImage(I);
I = histeq(I);
I = imfilter(I, fspecial('unsharp'));
Ie = imfilter(I, fspecial('average'));
```

6.3 Serial communication between computer and microcontroller

```
%PortInit
s = serial('COM3');
set(s, 'BaudRate', 1200, 'StopBits', 2);
% s.ReadAsyncMode = 'continuous';
% s.Timeout = Inf;
warning off
fopen(s);
while(kvar > 0)
      data = fscanf(s);
      if(length(data)>=2)
            if(data(1:2) == 'UT')
                  kvar = kvar - 1
            elseif(data(1:2) == 'IN')
                  kvar = kvar + 1
            end
      end
end
```

6.4 Gui code for enrollment

```
function varargout = enrollment(varargin)
%ENROLLMENT M-file for enrollment.fig
% ENROLLMENT, by itself, creates a new ENROLLMENT or raises the
existing
% singleton*.
%
% H = ENROLLMENT returns the handle to a new ENROLLMENT or the
handle to
% the existing singleton*.
%
% ENROLLMENT('Property','Value',...) creates a new ENROLLMENT
using the
% given property value pairs. Unrecognized properties are passed
via
% varargin to enrollment_OpeningFcn. This calling syntax
produces a
% warning when there is an existing singleton*.
%
% ENROLLMENT('CALLBACK') and ENROLLMENT('CALLBACK',hObject,...)
call the
% local function named CALLBACK in ENROLLMENT.M with the given
input
% arguments.
%
% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
only one
% instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help enrollment
% Last Modified by GUIDE v2.5 21-Mar-2006 15:48:29
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name', mfilename, ...
'gui_Singleton', gui_Singleton, ...
'gui_OpeningFcn', @enrollment_OpeningFcn, ...
'gui_OutputFcn', @enrollment_OutputFcn, ...
'gui_LayoutFcn', [], ...
'gui_Callback', []);
if nargin && ischar(varargin{1})
gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
[varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
% --- Executes just before enrollment is made visible.
function enrollment_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% varargin unrecognized PropertyName/PropertyValue pairs from the
% command line (see VARARGIN)
```

```
% Choose default command line output for enrollment
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
% UIWAIT makes enrollment wait for user response (see UIRESUME)
% uiwait(handles.figure1);
global h1 ImageData W StopCam obj
StopCam = 1;
PutCamOn;
h1 = [];
h = findobj('Tag', 'frontal');
h1 = [h1 h];
h = findobj('Tag', 'frontal2');
h1 = [h1 h];
h = findobj('Tag', 'left_profile');
h1 = [h1 h];
h = findobj('Tag', 'left_profile2');
h1 = [h1 h];
h = findobj('Tag', 'right_profile');
h1 = [h1 h];
h = findobj('Tag', 'right_profile2');
h1 = [h1 h];
set(h1, 'Enable', 'off');
set(handles.enroll_face, 'Enable', 'off');
set(handles.previous_face_candidate, 'Enable', 'off');
set(handles.next_face_candidate, 'Enable', 'off');
% set(handles.frontal, 'Enable', 'off');
set(handles.new_enrollement, 'Enable', 'off');
set(handles.sart_enrollment, 'Enable', 'on');
%Loading/Creating Image data
if(exist('ImageData.mat')==0)
ImageData = [];
W = 100;
else
load('ImageData');
end
% --- Outputs from this function are returned to the command line.
function varargout = enrollment_OutputFcn(hObject, eventdata,
handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;
% GetFace %Run OpenCV for face detection
% --- Executes on button press in next_face_candidate.
function next_face_candidate_Callback(hObject, eventdata, handles)
% hObject handle to next_face_candidate (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
global n k fil Im filename
set(handles.enroll_face, 'Enable', 'on');
if n < 1
n = 1;
set(handles.previous_face_candidate, 'Enable', 'off');
fil = dir('./FaceDir/*.jpg');
k = length(fil);
```

```
end
n = n + 1;
filename = fil(n).name;
In = imread(sprintf('.//FaceDir/%s', filename));
Im = rgb2gray(In);
Im = EnhanceImage(Im);
FacesLeft;
imagesc(In);% colormap(gray);
set(handles.previous_face_candidate, 'Enable', 'on');
if n >= k
set(handles.next_face_candidate, 'Enable', 'off');
n = n-1;
end
% --- Executes on button press in enroll_face.
function enroll_face_Callback(hObject, eventdata, handles)
% hObject handle to enroll_face (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
global enr ImageData Im h1 W
temp = imresize(Im, [W W], 'bilinear');
temp = temp(:);
ImageData = [ImageData temp];
enr = enr + 1;
if enr > 6
set(h1, 'Enable', 'off');
save('ImageData', 'ImageData', 'W');
EigenFace(ImageData, W);
save('ImageData', 'ImageData', 'W');
% set(handles.right_profile2, 'Enable', 'off');
set(handles.enroll_face, 'Enable', 'off');
set(handles.previous_face_candidate, 'Enable', 'off');
set(handles.next_face_candidate, 'Enable', 'off');
% set(handles.frontal, 'Enable', 'off');
% set(handles.left_profile, 'Enable', 'off');
% set(handles.right_profile2, 'Enable', 'off');
set(handles.new_enrollement, 'Enable', 'on');
enr = 1;
msgbox('Enrollement Succesfull!');
else
set(h1(enr-1), 'Enable', 'off');
set(h1(enr), 'Enable', 'on');
end
% --- Executes on button press in previous_face_candidate.
function previous_face_candidate_Callback(hObject, eventdata,
handles)
% hObject handle to previous_face_candidate (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
global n k fil Im filename
n = n - 1;
set(handles.next_face_candidate, 'Enable', 'on');
if n <= 1
n = 1;
set(handles.previous_face_candidate, 'Enable', 'off');
end
filename = fil(n).name;
In = imread(sprintf('.//FaceDir/%s', filename));
Im = rgb2gray(In);
```

```
Im = EnhanceImage(Im);
FacesLeft;
imagesc(In); % colormap(gray);
% --- Executes on button press in exit_button.
function exit_button_Callback(hObject, eventdata, handles)
% hObject handle to exit_button (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
global obj
delete('.\FaceDir\*.jpg');
closepreview(obj);
delete(obj);
close all
% --- Executes on button press in sart_enrollment.
function sart_enrollment_Callback(hObject, eventdata, handles)
% hObject handle to sart_enrollment (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
set(handles.next_face_candidate, 'Enable', 'on');
set(handles.frontal, 'Enable', 'on');
global n enr nFrame StopCam obj purpose a
a = 0;
StopCam = mod((StopCam + 1),2);
nFrame = Inf;
purpose = 'E';
if(StopCam == 1)
set(handles.sart_enrollment, 'String', 'Start Recordning');
StopCamnSaveFiles; %Run OpenCV for face detection
else
set(handles.sart_enrollment, 'String', 'Stop Recordning');
GetFace;
end
enr = 1;
n = 0;
% --- Executes on button press in new_enrollement.
function new_enrollement_Callback(hObject, eventdata, handles)
% hObject handle to new_enrollement (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% enrollment_OpeningFcn(hObject, eventdata, handles);
% enrollment_OutputFcn(hObject, eventdata, handles);
set(handles.faces_left, 'String', '');
delete('.\FaceDir\*.jpg');
axes(handles.detected_face_window);
imagesc(zeros(100));
```

6.5 Eigenvector generation

```
function EigenFace(K, W)
h = waitbar(0,'Please wait...');
ImData = double(K)./255;
waitbar(0.05);
% Compute mean image
M = mean(ImData')';
waitbar(0.1);
% Compute zero-mean images
```

```
ImData = ImData-repmat(M, [1 size(K, 2)]);
waitbar(0.3);
% Compute Principal components of the covariance matrix
[EigVect,D] = eig(ImData'*ImData);
waitbar(0.7);
% Next line is after Alex Chirokov (Alex.Chirokov@gmail.com) code
EigVect = ImData*EigVect*(abs(D))^(-0.5);
ME = max(EigVect(:));
mE = min(EigVect(:));
% Compression
EigVect = uint8(255.*(EigVect-mE)./(ME-mE));
waitbar(0.8);
% % size(EigVect)
% E = reshape(EigVect(:,MaxData-3), [W W]);
% E = E./max(E(:));
% imshow(E)
save('EigFace', 'EigVect','M', 'W', 'mE', 'ME');
waitbar(1);
close(h);
```