Chapter 9

Face Recognition Systems

On successful completion of this course, students will be able to:

- Explain how to detect face in OpenCV.
- Develop face recognition systems using library in OpenCV.

Introduction

The face is our primary focus of attention in developing an iintelligent robot to serves peoples. Unfortunatelly, developing a computational model of face recognition is quite difficult, because faces are complex, meaningful visual stimuli and multidimensional. Modelling of face images can be based on statistical model such as Principal Component Analysis (PCA) and Linear Discriminant analysis (LDA)and physical modelling based on the assumption of certain surface reflectance properties, such as Lambertian surface. OpenCV provides functions for face detector using Paul Viola and Michael Jones method, and OpenCV facetracker using Camshift algorithm.

Face Recognition in OpenCV

Face recognition is an easy task for humans, but not for computer systems. All face recognition models in OpenCV 2.4 are derived from the abstract base class FaceRecognizer, which provides a unified access to all face recongition algorithms in OpenCV. The currently available algorithms are:

- Eigenfaces (see createEigenFaceRecognizer())
- Fisherfaces (see createFisherFaceRecognizer())
- Local Binary Patterns Histograms (see createLBPHFaceRecognizer())

Experiments in [16] have shown, that even one to three day old babies are able to distinguish between known faces. So how hard could it be for a computer? It turns out we know little about human recognition to date. Are inner features (eyes, nose, mouth) or outer features (head shape, hairline) used for a successful face recognition? How do we analyze an image and how does the brain encode it? It was shown by David Hubel and Torsten Wiesel, that our brain has specialized nerve cells responding to specific local features of a scene, such as lines, edges, angles or movement. Since we don't see the world as scattered pieces, our visual cortex must somehow combine the different sources

of information into useful patterns. Automatic face recognition is all about extracting those meaningful features from an image, putting them into a useful representation and performing some kind of classification on them.

Face recognition based on the geometric features of a face is probably the most intuitive approach to face recognition. One of the first automated face recognition systems was described by Kanade in 1973, marker points (position of eyes, ears and nose) were used to build a feature vector (distance between the points, angle between them). The recognition was performed by calculating the euclidean distance between feature vectors of a probe and reference image.

The Eigenfaces method described in [15] took a holistic approach to face recognition: A facial image is a point from a high-dimensional image space and a lower-dimensional representation is found, where classification becomes easy. The lower-dimensional subspace is found with Principal Component Analysis, which identifies the axes with maximum variance. While this kind of transformation is optimal from a reconstruction standpoint, it doesn't take any class labels into account. Imagine a situation where the variance is generated from external sources, let it be light. The axes with maximum variance do not necessarily contain any discriminative information at all, hence a classification becomes impossible. So a class-specific projection with a Linear Discriminant Analysis was applied to face recognition in [17]. The basic idea is to minimize the variance within a class, while maximizing the variance between the classes at the same time.

Recently various methods for a local feature extraction emerged. To avoid the high-dimensionality of the input data only local regions of an image are described, the extracted features are (hopefully) more robust against partial occlusion, illumation and small sample size. Algorithms used for a local feature extraction are Gabor Wavelets [18], Discrete Cosinus Transform [19] and Local Binary Patterns [20]. It's still an open research question what's the best way to preserve spatial information when applying a local feature extraction, because spatial information is potentially useful information.

The problem with the image representation we are given is its high dimensionality. The Principal Component Analysis (PCA), which is the core of the Eigenfaces method, finds a linear combination of features that maximizes the total variance in data. While this is clearly a powerful way to represent data, it doesn't consider any classes and so a lot of discriminative information may be lost when throwing components away. Imagine a situation where the variance in your data is generated by an external source, let it be the light. The components

identified by a PCA do not necessarily contain any discriminative information at all, so the projected samples are smeared together and a classification becomes impossible.

The Linear Discriminant Analysis performs a class-specific dimensionality reduction and was invented by the great statistician Sir R. A. Fisher. In order to find the combination of features that separates best between classes the Linear Discriminant Analysis maximizes the ratio of between-classes to within-classes scatter, instead of maximizing the overall scatter. The idea is simple: same classes should cluster tightly together, while different classes are as far away as possible from each other in the lower-dimensional representation. This was also recognized by Belhumeur, Hespanha and Kriegman and so they applied a Discriminant Analysis to face recognition [22].

Haar Cascade Classifier

Viola-Jones framework has been widely used by researchers in order to detect the location of faces and objects in a given image. Face detection classifiers are shared by public communities, such as OpenCV [1]. Haar Cascade Classifier use AdaBoost at every node in cascade to study high detection level with multi-tree classifier rejection level at every node in cascade. This algorithm combines some innovative features, such as:

1) Use haar-like input feature, threshold that is used to sum and differentiate square regions from image.
2) Integral image technique that enable fast computation for square regions or regions that is rotated 45 degree. This data structure is used to make computation from Haar-like input feature faster.
3) Statistical Boosting to make binary node classification (yes/no) that characterized with high detection level and weak rejection level.
4) Organizing weak classifier nodes from a rejection cascade. In other words, first group from the classifiers is selected so best detection in image region consist of an object although enabling many mistakes in detection; the next classifier groups are the second best detection with weak level rejection; and so on. In testing, an object can be known if that object makes it through all cascades [2]. Haar-like input feature that are used by classifier are:

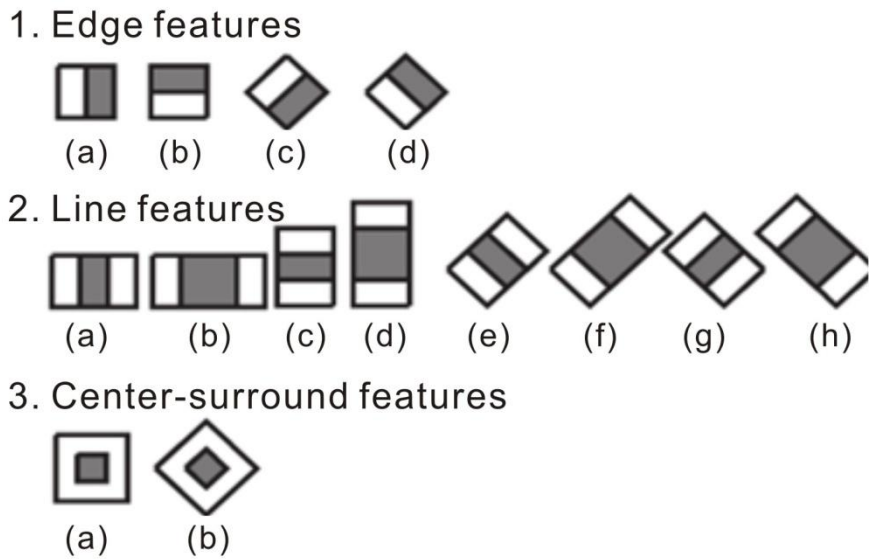

Figure 9.1 *Haar-like input feature that are used by classifiers [2].*

You have to inform to classifier, the directory to be used, such as haarcascade_frontalface_default.xml. At OpenCV, stored on:

```
Program_Files/OpenCV/data/haarcasades/haarcascade_frontalface_d
efault.xml.
```

To running the detector for face and eyes, you have to call detectMultiScale() that consist of 7 parameter:

```
//-- Detect faces
face_cascade.detectMultiScale (frame_gray, faces, 1.1, 2, 0,
Size(80, 80) );

for( int i = 0; i < faces.size(); i++ )
{
 Mat faceROI = frame_gray (faces[i]);
 std::vector<Rect> eyes;

 //-- In each face, detect eyes
 eyes_cascade.detectMultiScale (faceROI, eyes, 1.1, 2, 0
|CV_HAAR_SCALE_IMAGE, Size(30, 30) );
```

Program below show the demo to detect face and eyes using webcam:

HaarDetection.cpp

```
//Face and eyes detection using Haar Cascade Classifier
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>

using namespace std;
using namespace cv;

/** Function Headers */
void detectAndDisplay( Mat frame );

/** Global variables */
String face_cascade_name = "lbpcascade_frontalface.xml";
String                    eyes_cascade_name                    =
"haarcascade_eye_tree_eyeglasses.xml";
CascadeClassifier face_cascade;
CascadeClassifier eyes_cascade;
string window_name = "Face detection";

int main( int argc, const char** argv )
{
 CvCapture* capture;
 Mat frame;

 //-- 1. Load the cascade
 if( !face_cascade.load( face_cascade_name ) ){ printf("--
(!)Error loading face\n"); return -1; };
 if( !eyes_cascade.load( eyes_cascade_name ) ){ printf("--
(!)Error loading eye\n"); return -1; };

 //-- 2. Read the video stream
 capture = cvCaptureFromCAM(0);
 if( capture )
 {
  while( true )
  {
   frame = cvQueryFrame( capture );

   //-- 3. Apply the classifier to the frame
```

```
    if( !frame.empty() )
     { detectAndDisplay( frame ); }
    else
     { printf(" --(!) No captured frame -- Break!"); break; }

    int c = waitKey(10);
    if( (char)c == 'c' ) { break; }

   }
  }
  return 0;
 }

 /**
 * @function detectAndDisplay
 */
 void detectAndDisplay( Mat frame )
 {
  std::vector<Rect> faces;
  Mat frame_gray;

  cvtColor( frame, frame_gray, CV_BGR2GRAY );
  equalizeHist( frame_gray, frame_gray );

  //-- Detect faces
  face_cascade.detectMultiScale( frame_gray, faces, 1.1, 2, 0,
Size(80, 80) );

  for( int i = 0; i < faces.size(); i++ )
   {
    Mat faceROI = frame_gray( faces[i] );
    std::vector<Rect> eyes;

    //-- In each face, detect eyes
    eyes_cascade.detectMultiScale( faceROI, eyes, 1.1, 2, 0
|CV_HAAR_SCALE_IMAGE, Size(30, 30) );
    if( eyes.size() == 2)
    {
     //-- Draw the face
  Point center( faces[i].x + faces[i].width*0.5, faces[i].y +
faces[i].height*0.5 );
```

```
  ellipse( frame, center, Size( faces[i].width*0.5,
faces[i].height*0.5), 0, 0, 360, Scalar( 255, 0, 0 ), 2, 8, 0 );

   for( int j = 0; j < eyes.size(); j++ )
   { //-- Draw the eyes
    Point center( faces[i].x + eyes[j].x + eyes[j].width*0.5,
faces[i].y + eyes[j].y + eyes[j].height*0.5 );
     int radius = cvRound( (eyes[j].width +
eyes[j].height)*0.25 );
     circle( frame, center, radius, Scalar( 255, 0, 255 ), 3, 8,
0 );
     }
    }

   }
   //-- Show the result
   imshow( window_name, frame );
  }
```

The result of the program show in figure 9.2:

Figure 9.2 *Result of face detection using Haar classifier.*

Displaying face detected from webcam with ellipse and rectangle usually need by robotics engineer, because it can be used to measure distance between camera and the object, the rectangle codes:

```
cvCircle( img, center, radius, color, 3, 8, 0 );
cvRectangle( img,cvPoint( r->x, r->y ),cvPoint( r->x + r->width, r->y + r->height ),CV_RGB( 0, 255, 0 ), 1, 8, 0 );
```

Program below show an example for face detection with rectangle:

Figure 9.3 *Face detected using rectangle.*

FaceRectangle.cpp:

```
//Face Detection using Rectangle
#include "stdafx.h"
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
#include "cv.h"
```

```
#include "highgui.h"

#include <iostream>
#include <cstdio>

#ifdef _EiC
#define WIN32
#endif

using namespace std;
using namespace cv;

void detectAndDraw( Mat& img,
       CascadeClassifier& cascade, CascadeClassifier&
nestedCascade,
       double scale);

String cascadeName ="haarcascade_frontalface_alt.xml";

int main( int argc, const char** argv )
{
 CvCapture* capture = 0;
 Mat frame, frameCopy, image;
 const String scaleOpt = "--scale=";
 size_t scaleOptLen = scaleOpt.length();
 const String cascadeOpt = "--cascade=";
 size_t cascadeOptLen = cascadeOpt.length();
 String inputName;

 CascadeClassifier cascade, nestedCascade;
 double scale = 1;

 if( !cascade.load( cascadeName ) )
 {
  cerr << "ERROR: Could not load classifier cascade" << endl;
  cerr << "Usage: facedetect [--cascade=\"<cascade_path>\"]\n"
   "   [--nested-cascade[=\"nested_cascade_path\"]]\n"
   "   [--scale[=<image scale>\n"
   "   [filename|camera_index]\n" ;
  return -1;
 }
```

```
  capture = cvCaptureFromCAM(0);

   cvNamedWindow( "Face Detection with Rectangle", 1 );

   if( capture )
   {
    for(;;)
    {
     IplImage* iplImg = cvQueryFrame( capture );
     frame = iplImg;
     if( frame.empty() )
      break;
     if( iplImg->origin == IPL_ORIGIN_TL )
      frame.copyTo( frameCopy );
     else
      flip( frame, frameCopy, 0 );

     detectAndDraw( frameCopy, cascade, nestedCascade, scale );

     if( waitKey( 10 ) >= 0 )
     goto _cleanup_;
    }

    waitKey(0);
  _cleanup_:

    cvReleaseCapture( &capture );
   }

   cvDestroyWindow("result");

   return 0;
  }

  void detectAndDraw( Mat& img,
         CascadeClassifier& cascade, CascadeClassifier&
nestedCascade,
         double scale)
  {
   int i = 0;
   double t = 0;
   vector<Rect> faces;
```

```
   const static Scalar colors[] =  { CV_RGB(100,0,255),
     CV_RGB(0,100,255),
     CV_RGB(0,255,255),
     CV_RGB(0,255,0),
     CV_RGB(255,128,0),
     CV_RGB(255,255,0),
     CV_RGB(255,0,0),
     CV_RGB(255,0,255)} ;
   Mat gray, smallImg( cvRound (img.rows/scale),
cvRound(img.cols/scale), CV_8UC1 );

   cvtColor( img, gray, CV_BGR2GRAY );
   resize( gray, smallImg, smallImg.size(), 0, 0, INTER_LINEAR );
   equalizeHist( smallImg, smallImg );

   t = (double)cvGetTickCount();
   cascade.detectMultiScale( smallImg, faces,
     1.1, 2, 0
     //|CV_HAAR_FIND_BIGGEST_OBJECT
     //|CV_HAAR_DO_ROUGH_SEARCH
     |CV_HAAR_SCALE_IMAGE
     ,
     Size(30, 30) );
   t = (double)cvGetTickCount() - t;
   printf( "detection time = %g ms\n",
t/((double)cvGetTickFrequency()*1000.) );
   for( vector<Rect>::const_iterator r = faces.begin(); r !=
faces.end(); r++, i++ )
   {
     Mat smallImgROI;
     vector<Rect> nestedObjects;
     Point center;
     Scalar color = colors[i%8];
     int radius;
     center.x = cvRound((r->x + r->width*0.5)*scale);
     center.y = cvRound((r->y + r->height*0.5)*scale);
     radius = cvRound((r->width + r->height)*0.25*scale);

     circle( img, center, radius, color, 3, 8, 0 );
```

```
        cv::rectangle( img,cvPoint( r->x, r->y ),cvPoint( r->x +
r->width, r->y + r->height ),CV_RGB( 255, 0, 0 ), 1, 8, 0 );
   }
   cv::imshow( "Face Detection with Rectangle", img );
  }
```

Face Features Detector

Face features detector such as eye, nose and mouth very important for intelligent robotics. Robot should be able to recognize the expression (angry, sad, happy etc) obtained from a face in front of robot. An example below show face detected with eye, nose and mouth using libraries:

- haarcascade_frontalface_alt2.xml
- haarcascade_mcs_eyepair_big.xml
- haarcascad_mcs_nose.xml
- haarcascade_mcs_mouth.xml
- haarcascade_smile.xml

FacialFeatures.cpp:

```
#include <stdio.h>
#include<conio.h>
#include "cv.h"
#include "highgui.h"
#include "cvaux.h"

CvHaarClassifierCascade
*cascade,*cascade_e,*cascade_nose,*cascade_mouth;
CvMemStorage           *storage;
char *face_cascade="haarcascade_frontalface_alt2.xml";
char *eye_cascade="haarcascade_mcs_eyepair_big.xml";
char *nose_cascade="haarcascade_mcs_nose.xml";
char *mouth_cascade="haarcascade_mcs_mouth.xml";

/*Deteksi mulut*/
void detectMouth( IplImage *img,CvRect *r){
 CvSeq *mouth;
 cvSetImageROI(img,/* the source image */
```

```
   cvRect(r->x,           /* x = start from leftmost */
   r->y+(r->height *2/3), /* y = a few pixels from the top */
   r->width,    /* width = same width with the face */
  r->height/3    /* height = 1/3 of face height */
         )
       );
   mouth = cvHaarDetectObjects(img,/* the source image, with the
estimated
  location defined */
   cascade_mouth,     /* the eye classifier */
   storage,           /* memory buffer */
   1.15, 4, 0,        /* tune for your app */
    cvSize(25, 15)  /* minimum detection scale */
            );
     for( int i = 0; i < (mouth ? mouth->total : 0); i++ )
     {
          CvRect *mouth_cord = (CvRect*)cvGetSeqElem(mouth, i);
  /* draw a red rectangle */
  cvRectangle(img,
  cvPoint(mouth_cord->x, mouth_cord->y),
  cvPoint(mouth_cord->x + mouth_cord->width, mouth_cord->y +
mouth_cord->height),
  CV_RGB(255,255, 255),
  1, 8, 0
  );
     }
  }
  /*Deteksi hidung*/
  void detectNose( IplImage *img,CvRect *r){
   CvSeq *nose;
   //nose detection- set ROI
   cvSetImageROI(img,       /* the source image */
      cvRect(r->x,     /* x = start from leftmost */
      r->y, /* y = a few pixels from the top */
      r->width,  /* width = same width with the face */
```

```
          r->height  /* height = 1/3 of face height */
         )
       );

   nose = cvHaarDetectObjects(img, /* the source image, with the
estimated location defined */
          cascade_nose,   /* the eye classifier */
          storage,   /* memory buffer */
          1.15, 3, 0,   /* tune for your app */
          cvSize(25, 15)   /* minimum detection scale */
          );

   for( int i = 0; i < (nose ? nose->total : 0); i++ )
    {
     CvRect *nose_cord = (CvRect*)cvGetSeqElem(nose, i);

  /* gambar kotak merah */
  cvRectangle(img,
  cvPoint(nose_cord->x, nose_cord->y),
  cvPoint(nose_cord->x + nose_cord->width, nose_cord->y +
nose_cord->height),
  CV_RGB(0,255, 0),
  1, 8, 0
        );

    }
  }

  /*eye detection*/
  void detectEyes( IplImage *img,CvRect *r){
   char *eyecascade;
   CvSeq *eyes;
   int eye_detect=0;
  /* Set the Region of Interest: estimate the eyes' position */

   cvSetImageROI(img,    /* the source image */
     cvRect
     (
      r->x,       /* x = start from leftmost */
      r->y + (r->height/5.5), /* y = a few pixels from the top */
```

```
       r->width,    /* width = same width with the face */
       r->height/3.0  /* height = 1/3 of face height */
     )
    );

    /* deteksi mata */
    eyes = cvHaarDetectObjects( img, /* the source image, with
the
  estimated location defined */
     cascade_e,      /* the eye classifier */
     storage,        /* memory buffer */
     1.15, 3, 0,     /* tune for your app */
     cvSize(25, 15)  /* minimum detection scale */
            );
    printf("\n eye detected  %d",eyes->total);

     /* draw rectangle */
     for( int i = 0; i < (eyes ? eyes->total : 0); i++ )
      {
       eye_detect++;
       /* get one eye */
       CvRect *eye = (CvRect*)cvGetSeqElem(eyes, i);
       /* draw a red rectangle */
          cvRectangle(img,
     cvPoint(eye->x, eye->y),
     cvPoint(eye->x + eye->width, eye->y + eye->height),
     CV_RGB(0, 0, 255),
     1, 8, 0
     );
      }

  }
  void detectFacialFeatures( IplImage *img,IplImage *temp_img,int
img_no){

    char image[100],msg[100],temp_image[100];
    float m[6];
    double factor = 1;
    CvMat M = cvMat( 2, 3, CV_32F, m );
```

```
   int w = (img)->width;
   int h = (img)->height;
   CvSeq* faces;
   CvRect *r;

   m[0] = (float)(factor*cos(0.0));
   m[1] = (float)(factor*sin(0.0));
   m[2] = w*0.5f;
   m[3] = -m[1];
   m[4] = m[0];
   m[5] = h*0.5f;

   cvGetQuadrangleSubPix(img, temp_img, &M);
   CvMemStorage* storage=cvCreateMemStorage(0);
   cvClearMemStorage( storage );

   if( cascade )
    faces = cvHaarDetectObjects(img,cascade, storage, 1.2, 2,
CV_HAAR_DO_CANNY_PRUNING, cvSize(20, 20));
   else
    printf("\nFrontal face cascade not loaded\n");

  printf("\n Jumlah wajah yang dideteksi %d",faces->total);

   /* for each face found, draw a red box */
   for(int i = 0 ; i < ( faces ? faces->total : 0 ) ; i++ )
   {
    r = ( CvRect* )cvGetSeqElem( faces, i );
    cvRectangle( img,cvPoint( r->x, r->y ),cvPoint( r->x + r-
>width, r->y + r->height ),
         CV_RGB( 255, 0, 0 ), 1, 8, 0 );

  printf("\n face_x=%d face_y=%d wd=%d ht=%d",r->x,r->y,r-
>width,r->height);

    detectEyes(img,r);
    /* reset region of interest */
    cvResetImageROI(img);
    detectNose(img,r);
    cvResetImageROI(img);
    detectMouth(img,r);
```

```
    cvResetImageROI(img);
   }
   /* reset region of interest */
    cvResetImageROI(img);

    if(faces->total>0)
     {
  sprintf(image,"D:\\face_output\\%d.jpg",img_no);
      cvSaveImage( image, img );
     }
  }

  int main( int argc, char** argv )
  {
   CvCapture *capture;
   IplImage *img,*temp_img;
   Int        key;

   char image[100],temp_image[100];

  storage = cvCreateMemStorage( 0 );
  cascade = ( CvHaarClassifierCascade* )cvLoad( face_cascade, 0,
0, 0 );
  cascade_e = ( CvHaarClassifierCascade* )cvLoad( eye_cascade, 0,
0, 0 );
  cascade_nose = ( CvHaarClassifierCascade* )cvLoad( nose_cascade,
0, 0, 0 );
  cascade_mouth =
( CvHaarClassifierCascade* )cvLoad( mouth_cascade, 0, 0, 0 );

   if( !(cascade || cascade_e ||cascade_nose||cascade_mouth) )
    {
    fprintf( stderr, "ERROR: Could not load classifier
cascade\n" );
    return -1;
    }

   for(int j=20;j<27;j++)
   {

    sprintf(image,"D:\\image\\%d.jpg",j);
```

```
    img=cvLoadImage(image);
    temp_img=cvLoadImage(image);

  if(!img)
    {
    printf("Could not load image file and trying once
again: %s\n",image);
   }
    printf("\n curr_image = %s",image);
    detectFacialFeatures(img,temp_img,j);
   }

   cvReleaseHaarClassifierCascade( &cascade );
   cvReleaseHaarClassifierCascade( &cascade_e );

   cvReleaseHaarClassifierCascade( &cascade_nose );
   cvReleaseHaarClassifierCascade( &cascade_mouth );
   cvReleaseMemStorage( &storage );

   cvReleaseImage(&img);
   cvReleaseImage(&temp_img);
   return 0;
 }
```

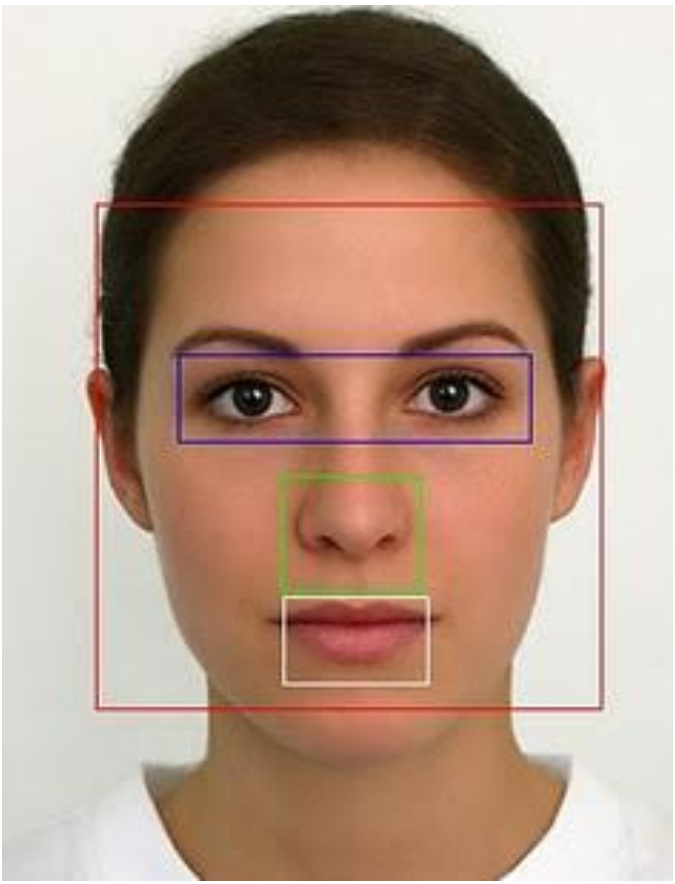

Figure 9.4 *Face features detected.*

Face Recognition Systems

We have developed a framework for face recognition system and faces database called ITS face database and will be compared with ATT and Indian face database. The advantages of our framework is able to store ordered item from customer in.xml file and displayed on the screen. In this research, we construct images under different illumination conditions by generate a random value for brightness level for ITS face database. Each of face database consists of 10 sets of people's face. Each set of ITS face database consists of 3 poses (front, left, right) and varied with illumination [13].

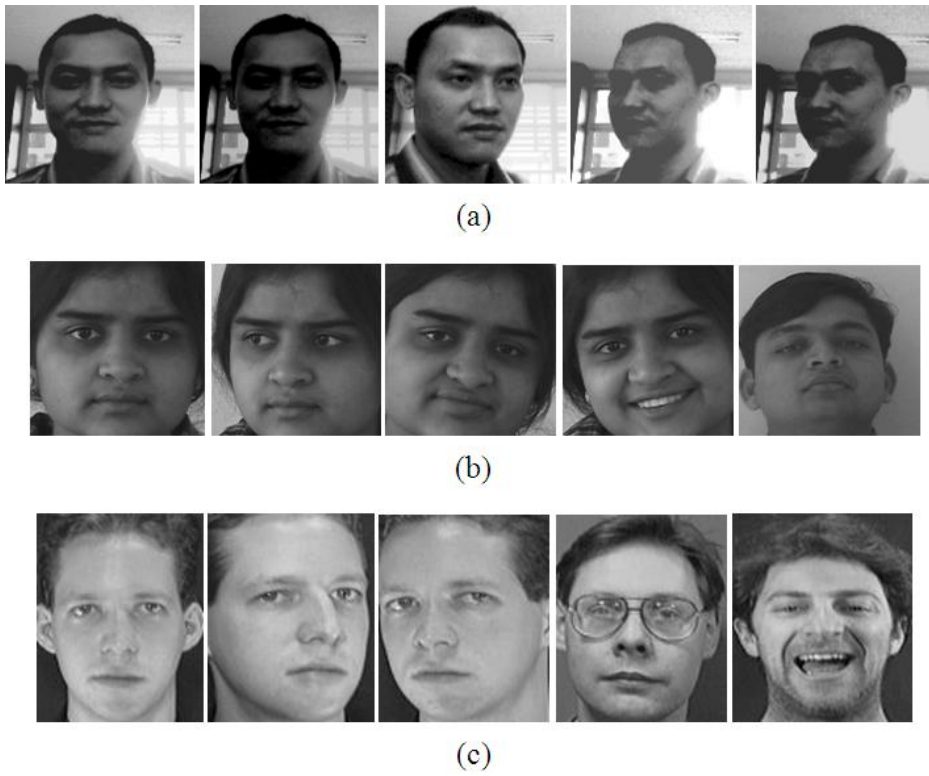

Figure 9.5 *ITS, Indian and ATT face database used as comparison to see the effect of illumination at face recognition [13].*

Rapid Object Detection with a Cascade of Boosted Classifiers Based on Haar-like Features

To train and use a cascade of boosted classifiers for rapid object detection. A large set of over-complete haar-like features provide the basis for the simple individual classifiers. Examples of object detection tasks are face, eye and nose detection, as well as logo detection. The sample detection task is logo detection, since logo detection does not require the collection of large set of registered and carefully marked object samples. For training a training samples must be collected. There are two sample types: negative samples and positive samples. Negative samples correspond to non-object images. Positive samples correspond to object images.

Negative Samples

Negative samples are taken from arbitrary images. These images must not contain object representations. Negative samples are passed through background description file. It is a text file in which each text line contains the filename (relative to the directory of the description file) of negative sample image. This file must be created manually. Note that the negative samples and sample images are also called background samples or background samples images, and are used interchangeably in this document. Example of negative description file:

```
/img
 img1.jpg
 img2.jpg

bg.txt
File bg.txt:
img/img1.jpg
img/img2.jpg
```

Positive Samples

Positive samples are created by createsamples utility. They may be created from single object image or from collection of previously marked up images. The single object image may for instance contain a company logo. Then are large set of positive samples are created from the given object image by randomly rotating, changing the logo color as well as placing the logo on arbitrary background.

The amount and range of randomness can be controlled by command line arguments.

Command line arguments:

- vec <vec_file_name>

name of the output file containing the positive samples for training

- img <image_file_name>

source object image (e.g., a company logo)

- bg <background_file_name>

background description file; contains a list of images into which randomly distorted versions of the object are pasted for positive sample generation

- num <number_of_samples>

number of positive samples to generate

- bgcolor <background_color>

background color (currently grayscale images are assumed); the background color denotes the transparent color. Since there might be compression artifacts, the amount of color tolerance can be specified by –bgthresh. All pixels between bgcolor-bgthresh and bgcolor+bgthresh are regarded as transparent.

- bgthresh <background_color_threshold>

- inv

if specified, the colors will be inverted

- randinv

if specified, the colors will be inverted randomly

- maxidev <max_intensity_deviation>

maximal intensity deviation of foreground samples pixels

- maxxangle <max_x_rotation_angle>,

- maxyangle <max_y_rotation_angle>,

- maxzangle <max_z_rotation_angle>

maximum rotation angles in radians

-show

if specified, each sample will be shown. Pressing 'Esc' will continue creation process without samples showing. Useful debugging option.

- w <sample_width>

width (in pixels) of the output samples

- h <sample_height>

height (in pixels) of the output samples

White noise is added to the intensities of the foreground. If –inv key is specified then foreground pixel intensities are inverted. If –randinv key is specified then it is randomly selected whether for this sample inversion will be applied. Finally, the obtained image is placed onto arbitrary background from the background description file, resized to the pixel size specified by –w and –h and stored into the file specified by the –vec command line parameter. Positive samples also may be obtained from a collection of previously marked up images. This collection is described by text file similar to background description file. Each line of this file corresponds to collection image. The first element of the line is image file name. It is followed by number of object instances. The following numbers are the coordinates of bounding rectangles (x, y, width, height).

Example of description file:

Directory structure:

```
/img
 img1.jpg
 img2.jpg
info.dat
```

```
File info.dat:
img/img1.jpg 1 140 100 45 45
img/img2.jpg 2 100 200 50 50   50 30 25 25
```

Image img1.jpg contains single object instance with bounding rectangle (140, 100, 45, 45). Image img2.jpg contains two object instances.

In order to create positive samples from such collection –info argument should be specified instead of –img:

```
- info <collection_file_name>
```

description file of marked up images collection

The scheme of sample creation in this case is as follows. The object instances are taken from images. Then they are resized to samples size and stored in output file. No distortion is applied, so the only affecting arguments are –w, -h, -show and –num.

Create samples utility may be used for examining samples stored in positive samples file. In order to do this only –vec, –w and –h parameters should be specified.

Note that for training, it does not matter how positive samples files are generated. So the createsamples utility is only one way to collect/create a vector file of positive samples.

Training

The next step after samples creation is training of classifier. It is performed by the haartraining utility.

Command line arguments:

- data <dir_name>

directory name in which the trained classifier is stored

- vec <vec_file_name>

file name of positive sample file (created by trainingsamples utility or by any other means)

- bg <background_file_name>

background description file

- npos <number_of_positive_samples>,

- nneg <number_of_negative_samples>

number of positive/negative samples used in training of each classifier stage. Reasonable values are npos = 7000 and nneg = 3000.

- nstages <number_of_stages>

number of stages to be trained

- nsplits <number_of_splits>

determines the weak classifier used in stage classifiers. If 1, then a simple stump classifier is used, if 2 and more, then CART classifier with number_of_splits internal (split) nodes is used

- mem <memory_in_MB>

available memory in MB for precalculation. The more memory you have the faster the training process

- sym (default),

- nonsym

specifies whether the object class under training has vertical symmetry or not. Vertical symmetry speeds up training process. For instance, frontal faces show off vertical symmetry

- minhitrate <min_hit_rate>

minimal desired hit rate for each stage classifier. Overall hit rate may be estimated as (min_hit_rate^number_of_stages)

- maxfalsealarm <max_false_alarm_rate>

maximal desired false alarm rate for each stage classifier. Overall false alarm rate may be estimated as (max_false_alarm_rate^number_of_stages)

- weighttrimming <weight_trimming>

Specifies wheter and how much weight trimming should be used. A decent choice is 0.90.

- eqw

- mode <BASIC (default) | CORE | ALL>

selects the type of haar features set used in training. BASIC use only upright features, while ALL uses the full set of upright and 45 degree rotated feature set. See [1] for more details.

- w <sample_width>,

- h <sample_height>

Size of training samples (in pixels). Must have exactly the same values as used during training samples creation (utility trainingsamples)

Note: in order to use multiprocessor advantage a compiler that supports OpenMP 1.0 standard should be used. OpenCV cvHaarDetectObjects() function (in particular haarFaceDetect demo) is used for detection.

Test Samples

In order to evaluate the performance of trained classifier a collection of marked up images is needed. When such collection is not available test samples may be created from single object image by createsamplesutility. The scheme of test samples creation in this case is similar to training samples creation since each test sample is a background image into which a randomly distorted and randomly scaled instance of the object picture is pasted at a random position. If both –img and –info arguments are specified then test samples will be created by createsamples utility. The sample image is arbitrary distorted as it was described below, then it is placed at random location to background image and stored. The corresponding description line is added to the file specified by –info argument. The –w and –h keys determine the minimal size of placed object picture.

The test image file name format is as follows:

```
imageOrderNumber_x_y_width_height.jpg,
```

where x, y, width and height are the coordinates of placed object bounding rectangle.

Note that you should use a background images set different from the background image set used during training. In order to evaluate the performance of the classifier performance utility may be used. It takes a collection of marked up images, applies the classifier and outputs the performance, i.e. number of found objects, number of missed objects, number of false alarms and other information.

Command line arguments:

```
- data <dir_name>
```

directory name in which the trained classifier is stored

```
- info <collection_file_name>
```

file with test samples description

```
- maxSizeDiff <max_size_difference>,
```

```
- maxPosDiff <max_position_difference>
```

determine the criterion of reference and detected rectangles coincidence. Default values are 1.5 and 0.3 respectively.

```
- sf <scale_factor>,
```

detection parameter. Default value is 1.2.

```
- w <sample_width>,
```

```
- h <sample_height>
```

Size of training samples (in pixels). Must have exactly the same values as used during training (utility haartraining).

Exercises

1) Create a program for smile detector using haarcascade_smile.xml.
2) Create a program for online face and Gender Recognition system using fischerfaces and OpenCV.

Figure 9.6 *Face and Gender Recognition Systems (improved from [21]).*

References

[1] Acosta, L., González, E.J., Rodríguez, J.N., Hamilton, A.F., Méndez J.A., Hernández S., Sigut S.M, and Marichal G.N. Design and Implementation of a Service Robot for A Restaurant. International Journal of robotics and automation. 2006; vol. 21(4): pp. 273-281.

[2] Qing-wiau Y., Can Y., Zhuang F. and Yan-Zheng Z. Research of the Localization of Restaurant Service Robot. International Journal of Advanced Robotic Systems. 2010; vol. 7(3): pp. 227-238.

[3] Chatib, O., Real-Time Obstacle Avoidance for Manipulators and Mobile Robots., The International Journal of Robotics Research, 1986; vol. 5(1), pp. 90-98.

[4] Borenstein, J., Koren, Y., The Vector Field Histogram- Fast Obstacle Avoidance for Mobile Robots, in proc. IEEE Trans. On Robotics and Automation. 1991; vol 7(3): pp.278-288.

[5] S. Nuryono, Penerapan Multi Mikrokontroler pada Model Robot Mobil Menggunakan Logika Fuzzy, Journal Telkomnika, 2009, vol. 7(3), pp, 213-218.

[6] Masehian E., Katebi Y. Robot Motion Planning in Dynamic Environments with Moving Obstacles and Target. International Journal of Mechanical Systems Science and Engineering. 2007; vol. 1(1), pp. 20-29.

[7] Budiharto, W., Purwanto, D. and Jazidie, A. A Robust Obstacle Avoidance for Service Robot using Bayesian Approach. International Journal of Advanced Robotic Systems. Intech Publisher – Austria. 2011; Vol. 8(1): pp. 52-60.

[8] Budiharto, W., Purwanto, D. & Jazidie, A. A Novel Method for Static and Moving Obstacle Avoidance for Service robot using Bayesian Filtering. Proceeding of IEEE 2nd International conf. on Advances in Computing, Control and Telecommunications Technology.2010; pp. 156-160. DOI: 10.1109/ACT.2010.51.

[9] Purwanto, D. Visual Feedback Control in Multi-Degrees-of-Freedom Motion System. PhD thesis at Graduate School of Science and Technology - Keio University, Japan. 2001.

[10] Turk, M. & Pentland A. Eigenfaces for recognition. International Journal of Cognitive Neuroscience. 1991; vol. 3(1): pp. 71-86.

[11] Belhumeur, P. & Kriegman, D. What is the set of images of an object under all possible illumination conditions. International Journal of Computer Vision. 1998; Vol. 28(3), pp. 245-260.

[12] Etemad, K. & Chellappa R. Discriminant analysis for recognition of human face images. Journal of the Optical Society of America A. 1997; vol. 14(8): pp. 1724-1733.

[13] Budiharto, W., Santoso A., Purwanto, D. and Jazidie, A. An Improved Face recognition System for Service Robot using Stereo Vision. In: Tudor Barbu Editor. Face Recognition / Book 3. Intech Publisher – Austria; 2011: pp. 1-12.

[14] Hu, H. & Brady, M. A Bayesian Approach to Real-Time Obstacle Avoidance for a Mobile Robot. Autonomous Robots. 1994; vol. 1: pp. 69-92.

[15] Turk, M., and Pentland, A. Eigenfaces for recognition. Journal of Cognitive Neuroscience 3 (1991), 71–86.

[16] Chiara Turati, Viola Macchi Cassia, F. S., and Leo, I. Newborns face recognition: Role of inner and outer facial features. Child Development 77, 2 (2006), 297–311.

[17] Belhumeur, P. N., Hespanha, J., and Kriegman, D. *Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection.* IEEE Transactions on Pattern Analysis and Machine Intelligence 19, 7 (1997), 711–720.

[18] Wiskott, L., Fellous, J., Krüger, N., Malsburg, C. *Face Recognition By Elastic Bunch Graph Matching.* IEEE Transactions on Pattern Analysis and Machine Intelligence 19 (1997), S. 775–779.

[19] Messer, K. et al. *Performance Characterisation of Face Recognition Algorithms and Their Sensitivity to Severe Illumination Changes.* In: In: ICB, 2006, S. 1–11.

[20] Ahonen, T., Hadid, A., and Pietikainen, M. *Face Recognition with Local Binary Patterns.* Computer Vision - ECCV 2004 (2004), 469–481.

[21] Daniel Bagio et al, Mastering OpenCV with Practical Computer Vision Project, Pact publisher, 2012.

[22] Opencv.org.