EEE 402 Project
Project Name: cursorControl

Group Members
Shafi Uddin Siddiqui
1206034
Md. Mesbahur Rahman
12060642
Syed Nakib Hossain
1206062
Prottay Debnath
1206063
Kazi Rezaul Karim Sagar
1206064

Submitted to
Sayeed Shafayet Chowdhury
Lecturer, EEE
Asir Intisar Khan
Lecturer, EEE

December 10, 2016

Contents

List of Figures

Abstract

The use of computers today has reached to a extent that we could not have imagined a ten to twelve years back. Its not only common to have a laptop or desktop computer in a household but it has become a must for a undergrad student to have one. On the other hand there are those unfortunates who can afford one but cannot use due to disability. This project "cameraCursor" is for those disabled person who cannot do any voluntary movement under the neck. We use face detection to detect a user and face tracking to continuously track user head movement to control the cursor. Also the user can simulate a click by keeping his face fixed in a position.

Chapter 1

Introduction

What could be the alternative to mouse or touch-pad, which are hand controlled, in using laptop? One thing that come to mind that is other moving parts of the body. Using face give us an advantage because face image can be captured using a webcam and done processing on it without making the user sit in an uncomfortable position. We can divide our project work to 3 steps -

1. **Face detection**. The most state of the art technique is "neural network". Companies like Google and Facebook have developed nearly accurate face detection system through neural network and deep learning. In our project we used **Viola Jones algorithm** - also a popular algorithm for recent years.

2. **Real-time face tracking**. Face tracking can be done by extracting the features on face and then tracking those features. KLT (KanadeLucasTomasi) tracker, PCA (Principal Component Analysis), AAM (Active Appearance Model) are some of the ways face tracking are done. We took a template image that contain the face of the user and matched it to the frame we captured in real time.

3. **Cursor position control**. For this a system function **SendInput** is used which sends a input command for mouse or keyboard using a structure called **Input**.

The **platform** we used is **Visual Studio 2012 Express Edition**. Also **Opencv 2.4.13** (Open Source Computer Vision) library is used for Viola Jones face detection and template matching.

Chapter 2

Previous Work

Our initial inspiration came from a program called **Camera Mouse**. It is a program that allows user to control the mouse pointer on a Windows computer by moving your head. It was invented by Prof. James Gips (Boston College) and Prof. Margrit Betke (then at Boston College, now at Boston University). Camera Mouse has proved very helpful to people who have no voluntary movement below the neck, people who can voluntarily control only their head. Since the first version of Camera Mouse was made available for free in June 2007, over 3,000,000 copies have been downloaded from **cameramouse.org**.

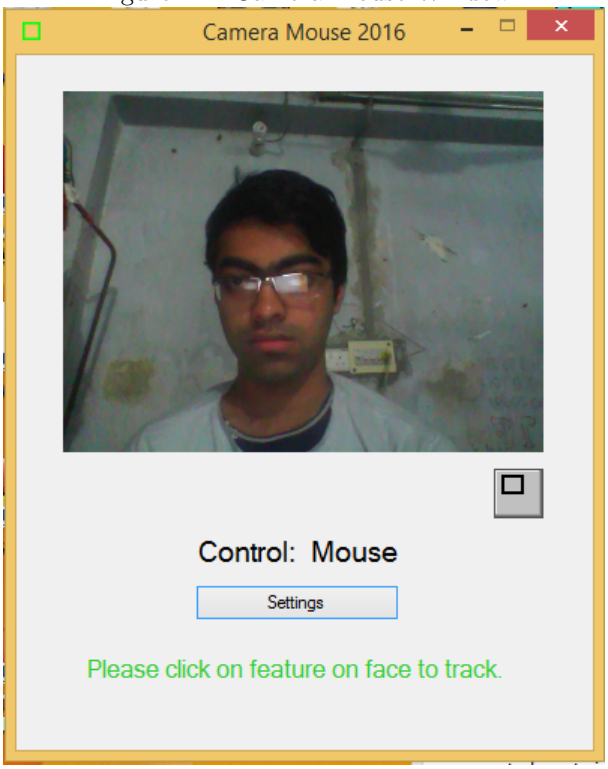
The program tells the user to select a feature on the head to track on a real time webcam preview window. It then track that feature as user move his
her head.

The program works well but has some flaws -

1. The user, who has no hand, has to select a feature using mouse or touch-pad which need hand movement.
2. Sometimes the tracker loses the tracking feature and once it does it cannot relocate the feature.

We tried to solve these problems in our project.

Figure 2.1: Camera Mouse Window



Chapter 3

Project

Here we discuss the project work step by step. The complete code is provided in appendix A.

3.1 Capturing a Frame

Frame is captured using Opencv **VideoCapture** class. We used the integrated webcam provided with the laptop. The resolution of the images was 640×480.

```
VideoCapture capture;
// check the video stream
capture.open (0);
if (!capture.isOpened ())
{
printf ("--(!)Error opening video capture\n");
if (waitKey (0) >= 0) return -1;
}
capture.read (frame);
```

The frame captured is then re-sized to 320×240 resolution. The Viola Jones algorithm convolves a quite number of kernels with the image. So, if the image is not re-sized the program becomes far too slow for practical use. The image webcam gives is mirrored one time so when the user moves right, in the image we see him to move left. Therefore the image also needs to be mirrored.

```
resize (frame,frame,Size(320,240),0,0,INTER_CUBIC);
flip (frame, frame, 1);
```

3.2 Initialization

When the program starts we run the initialization once. In this function we first detect faces in the given image. There might be multiple number of detected faces in the image. We consider the nearest person (and thus the bigger face) to be the user. Then a template image containing the face is returned. A region of interest of interest of a rectangle form is calculated.

The protoype for initialization function

```
int initialize (Mat &iniFrame, Mat &theTemplate, Rect &region_of_interest)
```

Where iniFrame is the frame given input to the function, theTemplate is returned template and, `region_of_interest` contains the region of interest information. All these are explained below -

3.2.1 Face Detection

Face is detected using the Viola Jones algorithm. The main concept of these algorithms are-

- **Haar feature** - Rectangular box with black and white square in it and consist of pixels as any image would. They are placed upon a region of an image and sum of product of pixels are calculated. There is a threshold to decide whether that place of image possesses that feature.

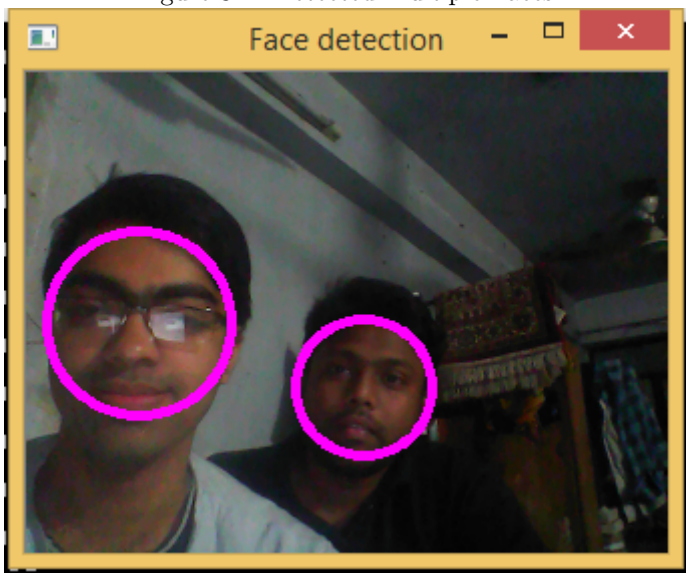
- **Image training** - Generally images of size 24×24 (in adaboost) are taken which contain only faces. 160000+ haar feature can be generated in this size of image. There are some positive images which contain faces and some negative images which do not. The haar features will have different values for positive and negative image (creating threshold).
- **Integral image** - All these feature calculations are very time-consuming. Integral image is a simple way to sum the pixel values in a haar feature.
- **Adaboost** - All the 160000+ feature are not useful. In adaboost technique, first all features are given the same weight. Now based on to what degree the features classifies the positive and negative images the weights are changed. There are some errors which are adjusted and new weights are calculated. This process goes on for several time. The whole process is very time-consuming - takes days to finish training. Now we have some 6000 feature with weights that best classifies the image. Other features are discarded. The sum of the weighted feature (weak classifier) that are accepted is the final classifier (strong classifier). This strong classifier is used to detect face in a image.
- **Cascade classifier** - The 6000 is also a very large number. So we made a cascade (or stage) of classifier. In first stage there are small number of weighted features. If first stage is passed there comes the second stage. there are more number of weak classifiers here. The difficulty of stages increases like this. Typically there are 20-30 stages. If all the stages are passed a face is detected.
- **Minimum neighbor** On a given image for face detection we will do the above tests taking 24×24 rectangles from the image. These rectangles are taken from throughout the image serially. If we set minimum neighbor to, for example, 3 then if there are 3 rectangles that passed the test overlaps one another then that region probably have a face. Increasing minimum neighbor we can decrease the chance of false positive - detecting faces in part of image where actually no face exists.

The function detectFaceAndDisplay detects faces

```
face detectFaceAndDisplay (Mat &frame)
{
        face faces(1);
    Mat frame_gray, faceROI;

        // some preprocessing
    cvtColor (frame, frame_gray, COLOR_BGR2GRAY);

        // Detect faces
        face_cascade.detectMultiScale (frame_gray, faces.faceRects,
            1.1, 5, 0 | CASCADE_SCALE_IMAGE, Size(30, 30));

        // if faces are found show them and set the empty pointer to 0
        if (!faces.faceRects.empty())
    {
                // set the pointer
                faces.empty = 0;
                faces.length = faces.faceRects.size() - 1;

                // draw ellipse on faces
                for (size_t i = 0; i <= faces.length; i++)
                {
                        Point centers;
                        centers.x = faces.faceRects[i].x + faces.
                            faceRects[i].width / 2;
                        centers.y = faces.faceRects[i].y + faces.
                            faceRects[i].height / 2;
                        ellipse (frame, centers, Size(faces.faceRects[i
                            ].width / 2, faces.faceRects[i].height / 2)
                            , 0, 0, 360,
                                Scalar(255, 0, 255), 4, 8, 0);
```

```
                }

                // Show what you got
                imshow ("Face detection", frame);
            }

            return faces;
}
```

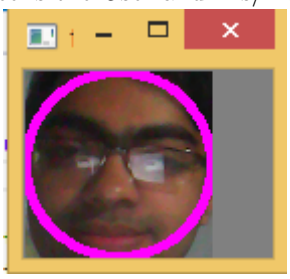
Figure 3.1: Detected Multiple Faces



3.2.2 Template

After locating the user in the given image we create a template image that contains the face of the user. Later this face is used to track image.

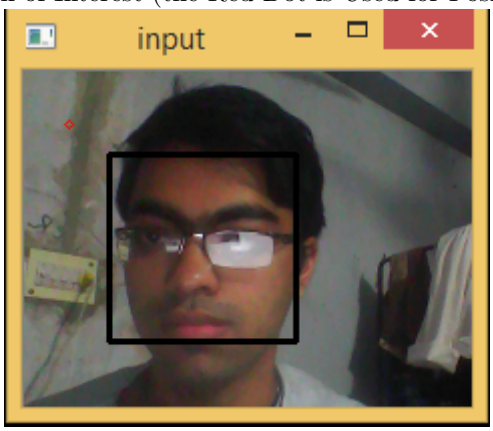
Figure 3.2: The Nearest Face is the User and His/Her Face is Used as Template



3.2.3 Region of Interest

The real time tracking is not done in 320×240 frame as the image are taken. Because then the user has to move his/her head much to reach the corners of monitor. Also other faces in full resolution image cause problem in template matching. So we take a suitable region of interest from the images taken in real time capturing where other face occurring is less probable and in which the user can move his head comfortably. We took a frame that has width 2.4 times and height 1.8 times the template.

Figure 3.3: Region of Interest (the Red Dot is Used for Position Calculation)



3.3 Template Matching

There are two primary components-

1. **Source image (I)**- the image in which we expect to find a match to the template image. we get this image from real time image capturing.
2. **Template image (T)**- the image which will be compared to the source image

The goal is to detect the highest matching area. The template is compared against the source image by sliding it one pixel at a time (left to right, up to down). At each location, a metric is calculated so it represents how good or bad the match at that location is. Each location (x,y) in result matrix R contains the match metric. The brightest or the darkest (depend upon which method is used) part of the normalized result matrix is calculated. The rectangle formed by that point as a corner and width and height equal to the patch image is considered the match.

We used the method `CV_TM_CCOEFF_NORMED` which has the equation

$$R(x,y) = \frac{\Sigma_{x',y'}(T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\Sigma_{x',y'}(T'(x',y')^2 \cdot \Sigma_{x',y'} I'(x+x',y+y')^2)}}$$

The function we used -

```
matchTemplate (img, templ, result, match_method);
normalize (result, result, 0, 1, NORM_MINMAX, -1, Mat());
```

And the best match location is found using -

```
minMaxLoc (result, &minVal, &maxVal, &minLoc, &maxLoc, Mat());
```

In our method best match is at the brightest location -

```
matchLoc = maxLoc
```

3.4 Changing the Co-ordinate

The position is calculated using the co-ordinate of the region of interest frame we got after initialization. But when user face nears the corner of this region template matching often fails. So we changed the co-ordinates for position calculation retaining a margin in the region of interest frame.

```
// changing the co ordinates
if (matchLoc.x > 20)
matchLoc.x -= 20;
else
matchLoc.x = 0;

if (matchLoc.y > 15)
matchLoc.y -= 15;
else
matchLoc.y = 0;
```

3.5 SendInput Function

The SendInput function inserts the events in the INPUT structures serially into the keyboard or mouse input stream. A simplified syntax is -

```
UINT WINAPI SendInput(UINT nInputs, LPINPUT pInputs, int cbSize);
```

Where, **nInputs** the number of structures in the pInputs array, **pInputs** an array of INPUT structures where, each structure represents an event to be inserted into the keyboard or mouse input stream and, **cbSize** the size, in bytes, of an INPUT structure.

The syntax for Input structure is -

```
typedef struct tagINPUT {
  DWORD type;
  union {
    MOUSEINPUT    mi;
    KEYBDINPUT    ki;
    HARDWAREINPUT hi;
  };
} INPUT, *PINPUT;
```

where, **type** is the type of the input event (0 for mouse which uses the mi structure) and **mi** is a MOUSEINPUT structure that contains the information about a simulated mouse event.

3.6 Clicking

We simulated a clicking (left-click) event if the cursor is in a position for a fixed amount of time. For this we check a 100 (that means a little more than 3 seconds) consecutive frames to see whether the cursor moved less than 4 pixels in either x or y direction combined.

```
if (check < clickThresh)
{
accuxPos += matchLoc.x - matchLocOld.x;
accuyPos += matchLoc.y - matchLocOld.y;

check++;
}
if (check == clickThresh)
{
accuxPos += matchLoc.x - matchLocOld.x;
accuyPos += matchLoc.y - matchLocOld.y;

if ( accuxPos < 4 && accuyPos < 4)
mouseLeftClick();
```

```
check = 0;
}
```

For the next clicking event we see the next 100 consecutive frames.

3.7 Halting the program

The program needs to be halted if we want control the cursor by mouse or touch-pad. Thus we check for a external intervention after doing operation on each frame. The program is halted using **Sleep()** function. Which takes milliseconds as a parameter and does not execute any instruction from code in this time span.

```
int sleepFlag = 0, count = 0;
while (++count < 25)
{
getMousePosition ();
cout << xPos << " " << yPos << endl << matchLoc << endl;
if (abs (xPos - matchLoc.x) > 2 || abs (yPos - matchLoc.y) > 2)
{
sleepFlag = 1;
break;
}
}
while (sleepFlag)
{
Sleep (10);


if (waitKey(20) > 0)
{
sleepFlag = 0;
break;
}
}
```

Chapter 4

Results

We emphasized before in chapter 2 that we wanted to improve two aspects of the previously done project Camera Mouse. We can say we accomplished this -

1. The user do not need to give any input hand related.
2. We did not used feature but only a face template and it always gives a match so we always gets a output. Also its face tracking is pretty accurate.

Chapter 5

Further Work

Although we improved in two aspects we wanted to our work has some drawbacks of its own -

1. Viola Jones algorithm cannot work well in various illumination. face detection fails in those cases. PCA or other techniques are good in prospects but there accuracy overall is inferior to Viola Jones.
2. Viola Jones algorithm also can only detect frontal faces.
3. Also our program cannot give angle information (to which degree the face is rotated to). So our position calculation is based on only translational position of faces. The cursor control can be made smooth by extracting angle information along with translational information.
4. We cannot move our head up-down direction with ease as we can do for right-left direction. So the user faces difficulty reach upper or corner portion of the display with our program.

The future work for these project should be remove above draw-backs. We should add some features to make it more usable -

1. Simulating right click
2. Simulating double click
3. Simulating holding down left mouse button for scrolling
4. A virtual keyboard.

Also we do not made the work in a suitable executable software format which also a later work that should be done to made the work commercially available.

Chapter 6

Conclusion

Our cursor control program were able to perform well in some aspects. But a better face detection algorithm (which can detect every person in any condition in the image) and more accurate face tracking algorithm can make it better. Neural network and KLT algorithm would be good choice. Also a lot more thinking needed to be done how to extract more information from face features and head gestures so that more features can be added to this program.

Bibliography

[1] Paul Viola , Michael Jones, "Rapid object detection using a boosted cascade of simple features", *ACCEPTED CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION*, 2001.

[2] Jianbo Shi , Carlo Tomasi, "Good Features to Track", *IEEE Conference on Computer Vision and Pattern Recognition (CVPR94)*, Seattle, June 1994

[3] Haoxiang Li, Zhe Lin, Xiaohui Shen, Jonathan Brandt, Gang Hua "A Convolutional Neural Network Cascade for Face Detection", *IEEE Conference on Computer Vision and Pattern Recognition (CVPR2015)*, 2015

[4] Documentaion: Opencv 2.4.13,
:http://docs.opencv.org/2.4/index.html

[5] Tutorial: Cascade Classifier,
:http://docs.opencv.org/2.4/doc/tutorials/objdetect/cascade_classifier/cascade_classifier.html

[6] Tutorial: Template Matching,
http::/docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/template_matching/template_matching.html

[7] Documentaion: SendInput function,
https://msdn.microsoft.com/en-us/library/windows/desktop/ms646310(v=vs.85).aspx

Appendix : The Full Code

```
// should try 0x501 below instead of 0x500 if we run into error
#define _WIN32_WINNT 0x500

// headers
#include "opencv2/objdetect.hpp"‘
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/video.hpp"
#include "opencv2/core/core.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/calib3d/calib3d.hpp"
#include "opencv2/nonfree/nonfree.hpp"
#include <windows.h>
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <vector>

// namespaces
using namespace std;
using namespace cv;

// objects
class face
{
public:
        vector<Rect> faceRects;
        int empty;
        size_t length;

        face (int a) {empty = a;}
};

// function prototypes
face detectFaceAndDisplay (Mat &frame);
int initialize (Mat &iniFrame, Mat &theTemplate, Rect &
   region_of_interest);
void getMousePosition ();
void mouseRightClick ();
void mouseLeftClick ();
void mouseMoveTo (int toX, int toY);

// global variables
String face_cascade_name = "haarcascade_frontalface_alt.xml";
String eyes_cascade_name = "haarcascade_eye_tree_eyeglasses.xml";
CascadeClassifier face_cascade;
CascadeClassifier eyes_cascade;
int xPos = 0, yPos = 0, accuxPos = 0, accuyPos = 0, check = 0;
const int clickThresh = 100;
int frameRes_width = 320, frameRes_height = 240;

// main function
int main()
{
        // variable declarations
```

```
    VideoCapture capture;
    Mat frame, templ, input;
    Rect ROI;
    int captured, initialized;

    // check the video stream
    capture.open (0);
    if (!capture.isOpened ())
    {
            printf ("--(!)Error opening video capture\n");
            if (waitKey (0) >= 0) return -1;
    }

    // Load the cascades
if (!face_cascade.load (face_cascade_name))
    {
            printf ("--(!)Error loading face cascade\n");
            if (waitKey (0) >= 0) return -1;
    }

    // initialization - 1. detecting a face and, 2. getting a ROI
    captured = 1, initialized = 1;
    while (1)
    {
            // Read a frame
            capture.read (frame);
            if (frame.empty ())
                    captured = 0;

            // resizing and mirroring the frame to reduce
               calculation and faster application
            if (captured)
            {
                    resize (frame,frame,Size(320,240),0,0,
                       INTER_CUBIC);
                    flip (frame, frame, 1);
                    initialized = initialize (frame, templ, ROI);
            }

            frameRes_width -= (templ.cols + 20);
            frameRes_height -= (templ.rows + 15);

            if (initialized)
                    break;
    }

    // live streaming and face tracking to give mouse pointer
       commands
    captured  = 1;
    while (1)
    {
            // Read a frame
            capture.read (frame);
            if (frame.empty ())
                    captured = 0;

            // resizing and mirroring
            if (captured)
```

```
{
        resize (frame,frame,Size(320,240),0,0,
            INTER_CUBIC);
        flip (frame, frame, 1);
}

// croping an image out of frame Mat with size of ROI
Mat ROIImage (frame, ROI);
input = ROIImage;
Mat img = input;

// create the result matrix
Mat result;
int result_cols =  img.cols - templ.cols + 1;
int result_rows = img.rows - templ.rows + 1;
result.create( result_cols, result_rows, CV_32FC1 );

// do the Matching and Normalize
int match_method = CV_TM_CCOEFF_NORMED;
matchTemplate (img, templ, result, match_method);
normalize (result, result, 0, 1, NORM_MINMAX, -1, Mat()
    );

// localizing the best match with minMaxLoc
double minVal, maxVal;
Point minLoc, maxLoc, matchLoc, matchLocOld;
matchLocOld.x = 0;
matchLocOld.y = 0;
minMaxLoc (result, &minVal, &maxVal, &minLoc, &maxLoc,
    Mat());

// for SQDIFF and SQDIFF_NORMED, the best matches are
    lower values. For all the other methods, the higher
     the better
if( match_method  == CV_TM_SQDIFF || match_method ==
    CV_TM_SQDIFF_NORMED )
        matchLoc = minLoc;
else
        matchLoc = maxLoc;

// draw the ractangles
rectangle (input, matchLoc, Point( matchLoc.x + templ.
    cols , matchLoc.y + templ.rows ), Scalar::all(0),
    2, 8, 0);
rectangle (result, matchLoc, Point( matchLoc.x + templ.
    cols , matchLoc.y + templ.rows ), Scalar::all(0),
    2, 8, 0);

// changing the co ordinates
if (matchLoc.x > 20)
        matchLoc.x -= 20;
else
        matchLoc.x = 0;

if (matchLoc.y > 15)
        matchLoc.y -= 15;
else
        matchLoc.y = 0;
```

```
// Show what we got
circle (input,matchLoc, 2, Scalar (0,0,255));
imshow ("input", input);
imshow ("template", templ);

// give mouse pointer a command to go to a position
if ( ( abs (matchLocOld.x - matchLoc.x) > 10 || abs (
    matchLocOld.y - matchLoc.y) > 10 )
        && (abs (matchLocOld.x - matchLoc.x) < 70 ||
            abs (matchLocOld.y - matchLoc.y) < 70) );
        mouseMoveTo (matchLoc.x, matchLoc.y);
matchLocOld = matchLoc;

// if the pointer is still at a position do a left
    click
if (check < clickThresh)
{
        accuxPos += matchLoc.x - matchLocOld.x;
        accuyPos += matchLoc.y - matchLocOld.y;

        check++;
}
if (check == clickThresh)
{
        accuxPos += matchLoc.x - matchLocOld.x;
        accuyPos += matchLoc.y - matchLocOld.y;

        if ( accuxPos < 4 && accuyPos < 4)
                mouseLeftClick();

        check = 0;
}

// halting the program
int sleepFlag = 0, count = 0;
while (++count < 25)
{
        getMousePosition ();
        cout << xPos << " " << yPos << endl << matchLoc
            << endl;
        if (abs (xPos - matchLoc.x) > 2 || abs (yPos -
            matchLoc.y) > 2)
        {
                sleepFlag = 1;
                break;
        }
}
while (sleepFlag)
{
        Sleep (10);


        if (waitKey(20) > 0)
        {
                sleepFlag = 0;
                break;
```

```
                        }
                }

                // if escaoe is pressed stop the program
                if (waitKey(20) > 0)
                        break;
        }

        getchar();
    return 0;
}

// function detectAndDisplay
face detectFaceAndDisplay (Mat &frame)
{
        face faces(1);
    Mat frame_gray, faceROI;

        // some preprocessing
    cvtColor (frame, frame_gray, COLOR_BGR2GRAY);

        // Detect faces
        face_cascade.detectMultiScale (frame_gray, faces.faceRects,
            1.1, 5, 0 | CASCADE_SCALE_IMAGE, Size(30, 30));

        // if faces are found show them and set the empty pointer to 0
        if (!faces.faceRects.empty())
    {
                // set the pointer
                faces.empty = 0;
                faces.length = faces.faceRects.size() - 1;

                // draw ellipse on faces
                for (size_t i = 0; i <= faces.length; i++)
                {
                        Point centers;
                        centers.x = faces.faceRects[i].x + faces.
                            faceRects[i].width / 2;
                        centers.y = faces.faceRects[i].y + faces.
                            faceRects[i].height / 2;
                        ellipse (frame, centers, Size(faces.faceRects[i
                            ].width / 2, faces.faceRects[i].height / 2)
                            , 0, 0, 360,
                                 Scalar(255, 0, 255), 4, 8, 0);
                }

                // Show what you got
                imshow ("Face detection", frame);
        }

        return faces;
}

// function initialize()
int initialize (Mat &iniFrame, Mat &theTemplate, Rect &
    region_of_interest)
{
        // detect face on the given image
```

```
face detectedFace(1);
detectedFace = detectFaceAndDisplay (iniFrame);
if (detectedFace.empty)
{
        cout << "detecting face....." << endl;
        return 0;
}

// determine the region of interest from the prime user
Point center;
size_t len = detectedFace.length;
center.x = detectedFace.faceRects[len].x + detectedFace.
   faceRects[len].width / 2;
center.y = detectedFace.faceRects[len].y + detectedFace.
   faceRects[len].height / 2;

if (!detectedFace.empty)
{
        Mat temp (iniFrame, detectedFace.faceRects[len]);
        theTemplate = temp;

        if (detectedFace.faceRects[len].x - detectedFace.
           faceRects[len].width * 0.7 > 0)
                region_of_interest.x = detectedFace.faceRects[
                   len].x - detectedFace.faceRects[len].width
                   /2;
        else
                region_of_interest.x = 0;

        if (detectedFace.faceRects[len].y - detectedFace.
           faceRects[len].height * 0.8 > 0)
                region_of_interest.y = detectedFace.faceRects[
                   len].y - detectedFace.faceRects[len].height
                   /2;
        else
                region_of_interest.y = 0;

        int flag_w = 1;
        for (float i = 2.4;; i = i - 0.2)
        {
                if ((region_of_interest.x + (int)(i *
                   detectedFace.faceRects[len].width) < 320)
                   && flag_w )
                {
                        region_of_interest.width = (int)(i *
                           detectedFace.faceRects[len].width);
                        flag_w = 0;
                }

                if (!flag_w)
                {
                        frameRes_width = region_of_interest.
                           width;
                        break;
                }
        }
        int flag_h = 1;
        for (float i = 1.8;; i = i - 0.2)
```

```
                {
                        if ((region_of_interest.y + (int)(i *
                           detectedFace.faceRects[len].height) < 240)
                           && flag_h)
                        {
                                region_of_interest.height = (int)(i *
                                   detectedFace.faceRects[len].height)
                                   ;
                                flag_h = 0;
                        }

                        if (!flag_h)
                        {
                                frameRes_height = region_of_interest.
                                   height;
                                break;
                        }
                }

                if (flag_w || flag_h)
                        cout << "cannot find suitable ROI" << endl;

        }

        return 1;
}

void getMousePosition ()
{
     // make a point and read into it, then update vars via pointers
     POINT cursorPos;
     GetCursorPos(&cursorPos);
     xPos = cursorPos.x;
     yPos = cursorPos.y;

         xPos = (xPos / 1365.0) * frameRes_width;
     yPos = (yPos / 767.0) * frameRes_height;
}

void mouseRightClick ()
{
     INPUT Input = {0};

     // right down
     Input.type = INPUT_MOUSE;
     Input.mi.dwFlags = MOUSEEVENTF_RIGHTDOWN;
     SendInput(1,&Input,sizeof(INPUT));

     // right up
     ZeroMemory(&Input,sizeof(INPUT));
     Input.type = INPUT_MOUSE;
     Input.mi.dwFlags = MOUSEEVENTF_RIGHTUP;
     SendInput(1,&Input,sizeof(INPUT));
}

void mouseLeftClick ()
{
     INPUT Input = {0};
```

```
    // left down
    Input.type = INPUT_MOUSE;
    Input.mi.dwFlags = MOUSEEVENTF_LEFTDOWN;
    SendInput(1,&Input,sizeof(INPUT));

    // left up
    ::ZeroMemory(&Input,sizeof(INPUT));
    Input.type = INPUT_MOUSE;
    Input.mi.dwFlags = MOUSEEVENTF_LEFTUP;
    SendInput(1,&Input,sizeof(INPUT));
}

void mouseMoveTo (int toX, int toY)
{
        // modify for the 65535 (as float) way we talk to the screen
            ...
    double dx = toX*(65535.0f / frameRes_width);
    double dy = toY*(65535.0f / frameRes_height);

    // movement stuff
    INPUT Input = {0};
    Input.type = INPUT_MOUSE;
    Input.mi.dwFlags = MOUSEEVENTF_MOVE|MOUSEEVENTF_ABSOLUTE;
    //Input.mi.dwFlags = MOUSEEVENTF_MOVE;
    Input.mi.dx = LONG(dx);
    Input.mi.dy = LONG(dy);
    SendInput(1,&Input,sizeof(INPUT));
}
```