Vision based smart camera for visual surveillance

Submitted in the partial fulfilment of MPMC lab

By

Shubham Rath (B115053)
Aisha Aishwarya Singh (B115005)

Under Department of CSE

Under the guidance of
Venkata Sridhar T
(Assistant Professor)

Table of contents

Abstract

Visual surveillance in dynamic scenes, especially for humans and vehicles, is currently one of the most active research topics in computer vision. It has a wide spectrum of promising applications, including access control in special areas, human identification at a distance, crowd flux statistics and congestion analysis, detection of anomalous behaviors, and interactive surveillance using multiple cameras, etc. In general, the processing framework of visual surveillance in dynamic scenes includes the following stages: modeling of environments, detection of motion, classification of moving objects, tracking, understanding and description of behaviors, human identification, and fusion of data from multiple cameras.

In this project, we are using Haarcascade classifier to detect and track human face and eyes. First we are detecting the face using the algorithm. Once the face has been detected, we are trying to detect the eye using the classifier twice in successive frames. We are then comparing the results of the classifier to try and find if there has occurred any eye blink. Once an eye blink is detected, tracking of the person starts and the camera follows the face of the person being tracked.

We are doing the vision processing in a computer and sending the coordinates of the face to the arduino using serial communication. In the arduino, we are calculating the relative distance of the face with respect to the current camera position and moving the camera so as the face remains in the center of the frame.

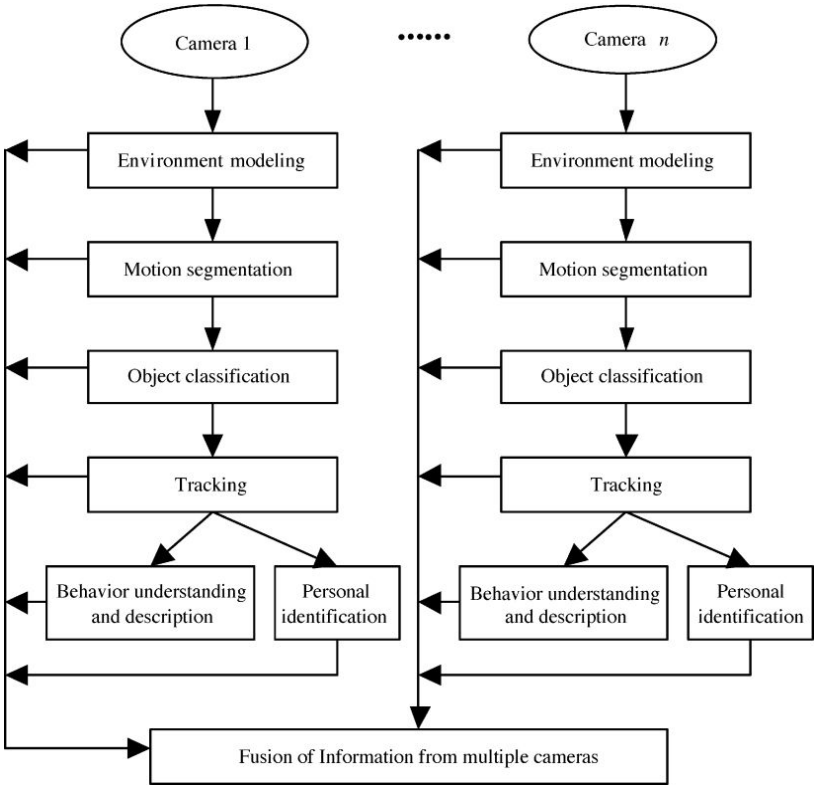
Introduction

In computer vision, visual surveillance is one of the active research topics. In dynamic scenes this area attempts to detect, recognize and track certain objects from image sequences, and more generally to understand and describe object behaviors. The aim is to develop intelligent visual surveillance to replace the traditional passive video surveillance that is proving ineffective as the number of cameras exceeds the capability of human operators to monitor them. In short, the goal of visual surveillance is not only to put cameras in the place of human eyes, but also to accomplish the entire surveillance task as automatically as possible.

Visual surveillance in dynamic scenes has a wide range of potential applications, such as a security guard for communities and important buildings, traffic surveillance in cities and expressways, detection of military targets, etc. We focus in this paper on applications involving the surveillance of people or vehicles, as they are typical of surveillance applications in general, and include the full range of surveillance methods. Surveillance applications involving people or vehicles include the following.

1. **Access control in special areas**. In some security-sensitive locations such as military bases and important governmental units, only people with a special identity are allowed to enter. A biometric feature database including legal visitors is built beforehand using biometric techniques. When somebody is about to enter, the system could automatically obtain the visitor's features, such as height, facial appearance and walking gait from images taken in real time, and then decide whether the visitor can be cleared for entry.
2. **Person-specific identification in certain scenes**. Personal identification at a distance by a smart surveillance system can help the police to catch suspects. The police may build a biometric feature database of suspects, and place visual surveillance systems at locations where the suspects usually appear, e.g., subway stations, casinos, etc. The systems automatically recognize and judge whether or not the people in view are suspects. If yes, alarms are given immediately. Such systems with face recognition have already been used at public sites, but the reliability is too low for police requirements.
3. **Crowd flux statistics and congestion analysis**. Using techniques for human detection, visual surveillance systems can automatically compute the flux of people at important public areas such as stores and travel sites, and then provide congestion analysis to assist in the management of the people. In the same way, visual surveillance systems can monitor expressways and junctions of the road network, and further analyze the traffic flow and the status of road congestion, which are of great importance for traffic management.

4. **Anomaly detection and alarming**. In some circumstances, it is necessary to analyze the behaviors of people and vehicles and determine whether these behaviors are normal or abnormal. For example, visual surveillance systems set in parking lots and supermarkets could analyze abnormal behaviors indicative of theft. Normally, there are two ways of giving an alarm. One way is to automatically make a recorded public announcement whenever any abnormal behavior is detected. The other is to contact the police automatically.
5. **Interactive surveillance using multiple cameras**. For social security, cooperative surveillance using multiple cameras could be used to ensure the security of an entire community, for example by tracking suspects over a wide area by using the cooperation of multiple cameras. For traffic management, interactive surveillance using multiple cameras can help the traffic police discover, track, and catch vehicles involved in traffic offences.



In this project we are trying to detect and track a human face realtime and move the camera according to the movement of the human. This has a number of applications in robotics and surveillance.

Literature Survey

It is the broad range of applications that motivates the interests of researchers worldwide. For example, the IEEE has sponsored the IEEE International Workshop on Visual Surveillance on three occasions, in India (1998), the U.S. (1999), and Ireland (2000). In [1] and [2], a special section on visual surveillance was published in June and August of 2000, respectively. In [3], a special issue on visual analysis of human motion was published in March 2001. In [4], a special issue on third-generation surveillance systems was published in October 2001. In [130], a special issue on understanding visual behavior was published in October 2002. Recent developments in human motion analysis are briefly introduced in our previous paper [5]. It is noticeable that, after the 9/11 event, visual surveillance has received more attention not only from the academic community, but also from industry and governments.

Visual surveillance has been investigated worldwide under several large research projects. For example, the Defense Advanced Research Projection Agency (DARPA) supported the Visual Surveillance and Monitoring (VSAM) project [6] in 1997, whose purpose was to develop automatic video understanding technologies that enable a single human operator to monitor behaviors over complex areas such as battlefields and civilian scenes. Furthermore, to enhance protection from terrorist attacks, the Human Identification at a Distance (HID) program sponsored by DARPA in 2000 aims to develop a full range of multimodal surveillance technologies for successfully detecting, classifying, and identifying humans at great distances. The European Union's Framework V Programme sponsored Advisor, a core project on visual surveillance in metrostations.

There have been a number of famous visual surveillance systems. The real-time visual surveillance system W4 [7] employs a combination of shape analysis and tracking, and constructs models of people's appearances in order to detect and track groups of people as well as monitor their behaviors even in the presence of occlusion and in outdoor environments. This system uses the single camera and grayscale sensor. The VIEWS system [87] at the University of Reading is a three-dimensional (3-D) model based vehicle tracking system. The Pfinder system developed by Wren et al. [8] is used to recover a 3-D description of a person in a large room. It tracks a single nonoccluded person in complex scenes, and has been used in many applications. As a single-person tracking system, TI, developed by Olsen et al. [9], detects moving objects in indoor scenes using motion detection, tracks them using first-order prediction, and recognizes behaviors by applying predicates to a graph formed by linking corresponding objects in successive frames. This system cannot handle small motions of background objects. The system at CMU [10] can monitor activities over a large area using multiple cameras that are connected into a network. It can detect and track multiple persons and vehicles within cluttered scenes and

monitor their activities over long periods of time. The above comments on [8]–[9][10] are derived from [7]. Please see [7] for more details.

As far as hardware is concerned, companies like Sony and Intel have designed equipment suitable for visual surveillance, e.g., active cameras, smart cameras [11], omni-directional cameras [12], [13], etc.

Visual tracking of objects of interest, such as faces, has received significant attention in the vision community. Accurate tracking is made difficult by the changing appearance of targets due to their nonrigid structure, 3D motion, interaction with other objects (*e.g.*, occlusions) and changes in the environment, such as illumination. Recent tracking methods, such as the Incremental Visual Tracker (IVT) [14], attempt to solve these problems using adaptive target appearance models. They represent the target in a low-dimensional subspace which is updated adaptively using the images tracked in the previous frames. Compared to the approaches equipped with a fixed target model such as eigentracking of [15], IVT is more robust to changes in appearance (*e.g.*, pose, illumination). However, the main drawback of the adaptive approaches is their susceptibility to drift: they can gradually adapt to non-targets as the target model is built solely from the previous tracked images accepted by the tracker. Methods such as IVT typically lack mechanisms for detecting or correcting drift as they have no global constraints on the overall appearance of the target object. For faces, such constraints could be learned from a set of generic (non-person specific) well-cropped and well-aligned face images that span possible variations in pose, illumination, and expressions. These can be seen as *visual constraints* that the target appearance should meet.

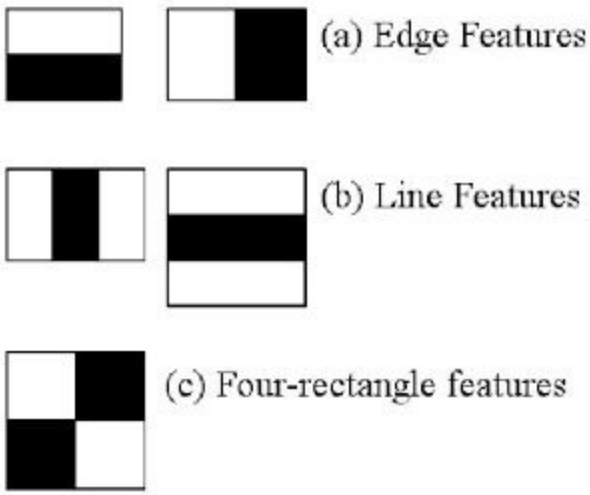
We are implementing the surveillance system on a normal computer with 4 CPUs and no GPU support. The use of only CPUs slows the processing by a bit but is not much recognizable for a small scale tracking of a single person.

Project Work

We are trying to implement a simple visual surveillance system using Computer Vision algorithms. Our project is able to detect and track a single person's face and move the camera according to his/her movements in 3D plane. We have used the **Haarcascade classifier** algorithm for the detection of face and eye.

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "*Rapid Object Detection using a Boosted Cascade of Simple Features*" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

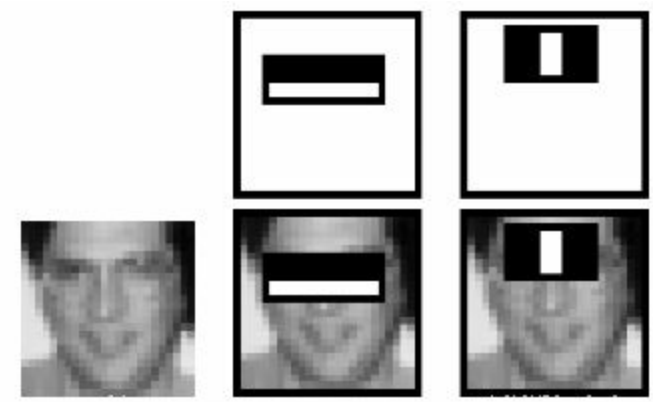
Here we work with face detection. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, Haar features shown in the below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle.



Now, all possible sizes and locations of each kernel are used to calculate lots of features. (Just imagine how much computation it needs? Even a 24x24 window results over 160000 features). For each feature calculation, we need to find the sum of the pixels under white and black rectangles. To solve this, they introduced the integral image. However large your image, it

reduces the calculations for a given pixel to an operation involving just four pixels. Nice, isn't it? It makes things super-fast.

But among all these features we calculated, most of them are irrelevant. For example, consider the image below. The top row shows two good features. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. But the same windows applied to cheeks or any other place is irrelevant. So how do we select the best features out of 160000+ features? It is achieved by **Adaboost**.



We are using a pre-trained open-source dataset for the classification of faces and detection. To discard photos and other patterns, we have implemented an eye blink detection mechanism. The tracking of the face starts only after an eye blink is detected by the camera.

To detect an eye blink, we have used haarcascade's eye detection algorithm. We are detecting the eye twice at a steady framerate with successive frames and comparing for any change. If there is a change in the result, the eye has opened and closed and a blink is detected. The functions used to detect the eye:

```
leftEyeDetector.detectMultiScale(faceROI, eyes, 1.1, 2,
CASCADE_FIND_BIGGEST_OBJECT, Size(0, 0));
rightEyeDetector.detectMultiScale(faceROI, eyes, 1.1, 2,
CASCADE_FIND_BIGGEST_OBJECT, Size(0, 0));
```

Hardware Requirement:

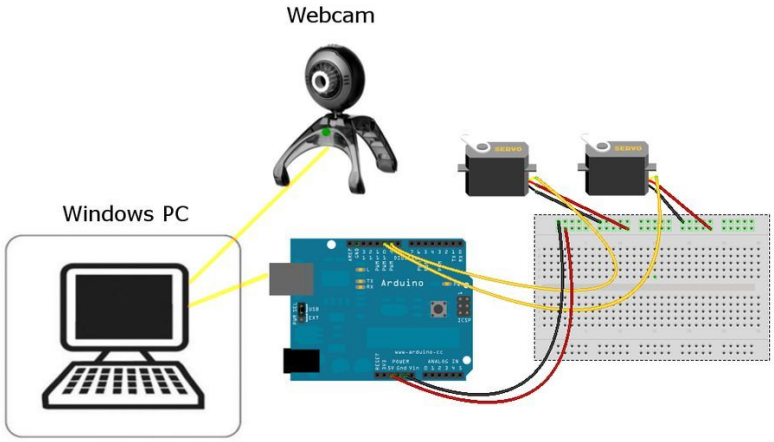
- PC preferably running Windows/Linux. The faster CPU the better.
- Arduino Uno or compatible + power source.
- Standard servos X 2 (SG90).
- Webcam w/UBS interface.
- Breadboard.
- Jumper wires.
- Glue and stand.

Software Requirement:

- OpenCV (3.0 or greater)
- Any code editor
- Arduino IDE
- Serial C++ library

Procedure:

- Installing OpenCV: https://sr6033.github.io/computer-vision/2017/11/08/OpenCV-part-1.html
- Once OpenCV is installed, connection of the parts are done as per the circuit diagram below:



Wiring of the various parts:

- **SERVOS:**
  - The orange/signal wire for the pan (X axis) servo goes to digital pin 9.
  - The orange/signal wire for the tilt (y axis) servo goes to digital pin 10.
  - The red/Vcc wires of both servos go to the Arduino's 5V pin.
  - The brown/GND wires of both servos go to Arduino's GND pin.
- **WEBCAM:**
  - The webcam's USB goes to the PC. The C++ code will identify it via a number representing the USB port it is connected to.
- **ARDUINO:**
  - The Arduino Uno is connected to the PC via USB.
- After the wiring has been done, the c++ code is compiled using the following command:
  - *g++ `pkg-config --cflags opencv` track.cpp `pkg-config --libs opencv`*
- The COM port is given access for serial communication by the following command:
  - *sudo chown sr6033 /dev/ttyUSB0*
- The arduino code is dumped in the board and then the execution of the program is done using the following command:
  - *./a.out /dev/ttyUSB0 3*

C++ code for vision and tracking:

File: track.cpp

```
001: #include "opencv2/objdetect/objdetect.hpp"
002: #include "opencv2/highgui/highgui.hpp"
003: #include "opencv2/imgproc/imgproc.hpp"
004:
005: #include <iostream>
006: #include <cstdio>
007: #include "Eserial.h"
008:
009: using namespace std;
010: using namespace cv;
011:
012: /** Function Headers */
013: void detectAndDisplay (Mat frame, int threshold);
014: vector<Rect> storeLeftEyePos(Mat rightFaceImage);
015: vector<Rect> storeLeftEyePos_open(Mat rightFaceImage);
016:
```

```
017: /** Global variables */
018: String face_cascade_name = "haarcascade_frontalface_alt.xml";
019: String eyes_cascade_name = "haarcascade_eye_tree_eyeglasses.xml";
020: String leftEyeCascade = "haarcascade_lefteye_2splits.xml";
021: String rightEyeCascade = "haarcascade_righteye_2splits.xml";
022: CascadeClassifier face_cascade;
023: CascadeClassifier eyes_cascade;
024: CascadeClassifier leftEyeDetector;
025: CascadeClassifier rightEyeDetector;
026: string window_name = "Captcha";
027:
028: // Serial to Arduino global declarations
029: int arduino_command;
030: Eserial * arduino_com;
031: short MSBLSB = 0;
032: unsigned char MSB = 0;
033: unsigned char LSB = 0;
034:
035: int open_eye = 0, close_eye = 0;
036:
037: // Serial to Arduino global declarations
038: int main (int argc, const char **argv)
039: {
040:     VideoCapture capture(1);
041:     Mat frame;
042:     char c = argv[2][0];
043:     cout << c << endl;
044:
045:     // serial to Arduino setup
046:     arduino_com = new Eserial ();
047:     if (arduino_com != 0)
048:     {
049:             //Standard Arduino uses /dev/ttyACM0 typically, Shrimping.it
050:             //seems to use /dev/ttyUSB0 or /dev/ttyUSB1.
051:             arduino_com->connect(argv[1], 57600, spNONE);
052:     }
053:
054:     // serial to Arduino setup
055:
056:     //-- 1. Load the cascades
057:     if (!face_cascade.load (face_cascade_name))
058:     {
059:             printf("--(!)Error loading\n");
060:             return -1;
```

```
061:    };
062:    if (!eyes_cascade.load (eyes_cascade_name))
063:    {
064:            printf("--(!)Error loading\n");
065:            return -1;
066:    };
067:
068:    if (!leftEyeDetector.load (leftEyeCascade))
069:    {
070:            printf("--(!)Error loading\n");
071:            return -1;
072:    };
073:    if (!rightEyeDetector.load (rightEyeCascade))
074:    {
075:            printf("--(!)Error loading\n");
076:            return -1;
077:    };
078:    //-- 2. Read the video stream
079:    //capture = cvCaptureFromCAM (-1);
080:
081:    if (capture.isOpened())
082:    {
083:            while (true)
084:            {
085:                    capture.read(frame);
086:                    //-- 3. Apply the classifier to the frame
087:                    if(!frame.empty ())
088:                    {
089:                            detectAndDisplay (frame, c-'0');
090:                    }
091:                    else
092:                    {
093:                            printf (" --(!) No captured frame -- Break!");
094:                            break;
095:                    }
096:                    int c = waitKey (10);
097:                    if((char) c == 'c')
098:                    {
099:                            break;
100:                    }
101:            }
102:    }
103:
104:    // Serial to Arduino - shutdown
```

```
105:    arduino_com->disconnect ();
106:    delete arduino_com;
107:    arduino_com = 0;
108:
109:    // Serial to Arduino - shutdown
110:    return 0;
111: }
112:
113:
114: /**
115:  * @function detectAndDisplay
116:  */
117: void detectAndDisplay (Mat frame, int threshold)
118: {
119:    std::vector <Rect> faces;
120:    Mat frame_gray;
121:    cvtColor (frame, frame_gray, CV_BGR2GRAY);
122:    equalizeHist (frame_gray, frame_gray);
123:
124:    //-- Detect faces
125:    face_cascade.detectMultiScale (frame_gray, faces, 1.1, 2, 0 |
CV_HAAR_SCALE_IMAGE, Size (30, 30));
126:    for (int i = 0; i < faces.size (); i++)
127:    {
128:            Point center (faces[i].x + faces[i].width * 0.5, faces[i].y + faces[i].height * 0.5);
129:            //ellipse (frame, center, Size (faces[i].width * 0.5, faces[i].height * 0.5), 0, 0,
360, Scalar (255, 0, 255), 2, 8, 0);
130:
131:            Mat faceROI = frame_gray (faces[i]);
132:            std::vector <Rect> eyes;
133:            std::vector <Rect> eyes1;
134:
135:            leftEyeDetector.detectMultiScale(faceROI, eyes, 1.1, 2,
CASCADE_FIND_BIGGEST_OBJECT, Size(0, 0));
136:            rightEyeDetector.detectMultiScale(faceROI, eyes1, 1.1, 2,
CASCADE_FIND_BIGGEST_OBJECT, Size(0, 0));
137:
138:            for ( size_t j = 0; j < eyes.size(); j++ )
139:            {
140:                    Point eye_center( faces[i].x + eyes[j].x + eyes[j].width/2, faces[i].y +
eyes[j].y + eyes[j].height/2 );
141:                    int radius = cvRound( (eyes[j].width + eyes[j].height)*0.25 );
142:                    circle( frame, eye_center, radius, Scalar( 255, 0, 0 ), 4, 8, 0 );
143:            }
```

```
144:            for ( size_t j = 0; j < eyes1.size(); j++ )
145:            {
146:                    Point eye_center( faces[i].x + eyes1[j].x + eyes1[j].width/2, faces[i].y +
eyes1[j].y + eyes1[j].height/2 );
147:                    int radius = cvRound( (eyes1[j].width + eyes1[j].height)*0.25 );
148:                    circle( frame, eye_center, radius, Scalar( 255, 0, 0 ), 4, 8, 0 );
149:            }
150:
151:            if (eyes.size() > 0)
152:            {
153:               // Now look for open eyes only
154:               vector<Rect> eyesRightNew = storeLeftEyePos_open(faceROI);
155:
156:               if(open_eye > threshold && close_eye > threshold)
157:                    {
158:                            //  cout << "X:" << faces[i].x  <<  "  y:" << faces[i].y  << endl;
159:
160:                            // send X,Y of face center to serial com port
161:                            // send X axis
162:                            // read least significant byte
163:
164:                            LSB = faces[i].x & 0xff;
165:
166:                            // read next significant byte
167:                            MSB = (faces[i].x >> 8) & 0xff;
168:                            arduino_com->sendChar (MSB);
169:                            arduino_com->sendChar (LSB);
170:                            //cout << "X MSB: " << hex << (int) MSB << ", X LSB: " <<
(int) LSB << endl;
171:
172:                            // Send Y axis
173:                            LSB = faces[i].y & 0xff;
174:                            MSB = (faces[i].y >> 8) & 0xff;
175:                            arduino_com->sendChar (MSB);
176:                            arduino_com->sendChar (LSB);
177:                            //cout << "Y MSB: " << hex << (int) MSB << ", Y LSB: " <<
(int) LSB << endl;
178:
179:                            // serial com port send
180:                    }
181:                    else
182:                    if (eyesRightNew.size() > 0) //Eye is open
183:               {
184:                            cout << "OPEN" << endl;
```

```
185:                                    open_eye++;
186:                    }
187:                    else //Eye is closed
188:                    {
189:                            cout << "close" << endl;
190:                            close_eye++;
191:                    }
192:                }
193:    }
194:    //-- Show what you got
195:    imshow (window_name, frame);
196: }
197:
198: // Method for detecting open eyes in face
199: vector<Rect> storeLeftEyePos_open(Mat rightFaceImage)
200: {
201:    std::vector<Rect> eyes;
202:    eyes_cascade.detectMultiScale(rightFaceImage, eyes, 1.1, 2,
CASCADE_FIND_BIGGEST_OBJECT, Size(0, 0));
203:    return eyes;
204: }
```

Arduino Code for camera control:

File: cam_servo.ino

```
 001: #include <Servo.h>
002:
003: #define  servomaxx   180   // max degree servo horizontal (x) can turn
004: #define  servomaxy   180   // max degree servo vertical (y) can turn
005: #define  screenmaxx   320   // max screen horizontal (x)resolution
006: #define  screenmaxy   240    // max screen vertical (y) resolution
007: #define  servocenterx   90  // center po#define  of x servo
008: #define  servocentery   90  // center po#define  of y servo
009: #define  servopinx   9   // digital pin for servo x
010: #define  servopiny   10  // digital servo for pin y
011: #define  baudrate 57600  // com port speed.
012: #define distancex 1  // x servo rotation steps
013: #define distancey 1  // y servo rotation steps
014:
015: int valx = 0;       // store x data from serial port
```

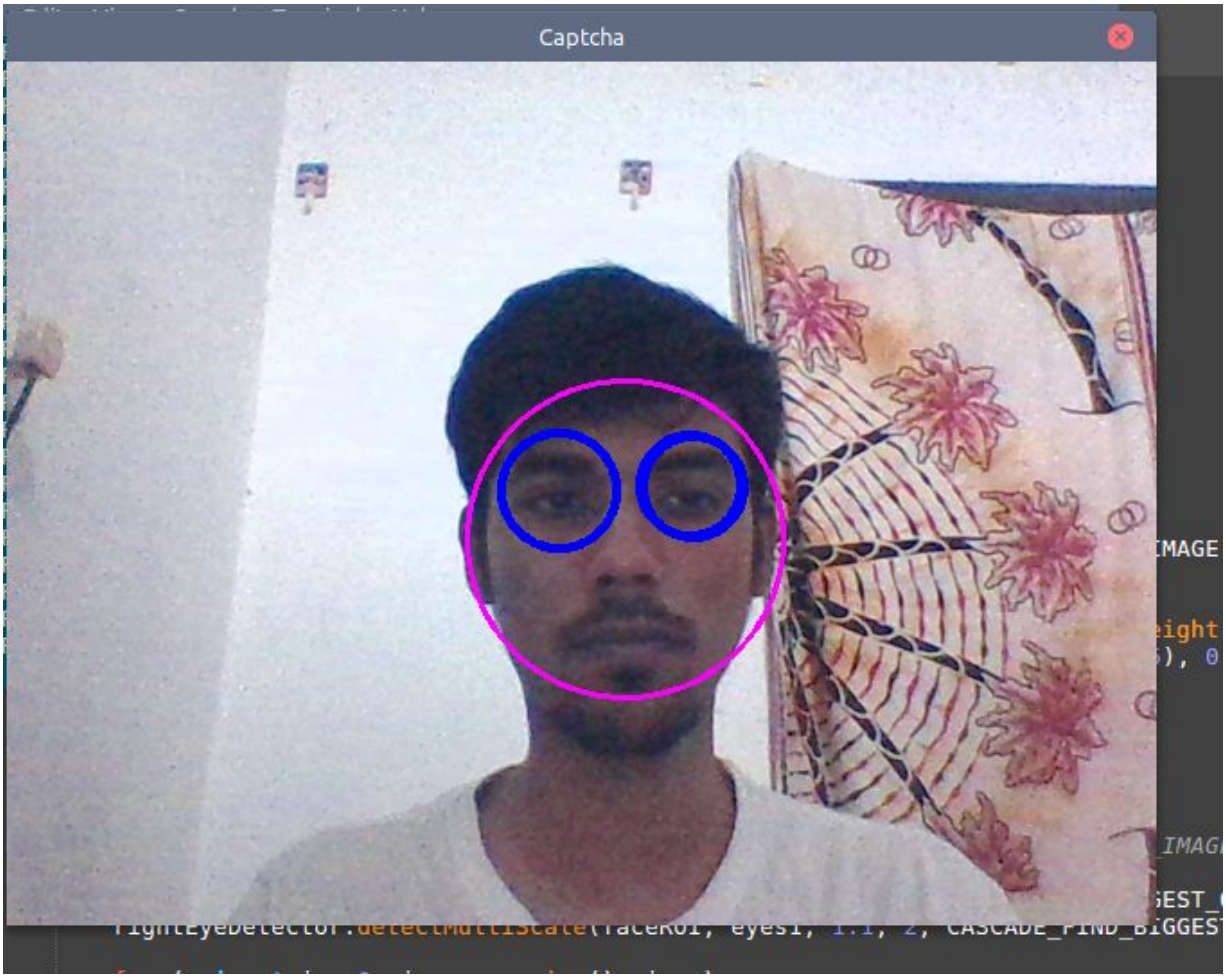
```
016: int valy = 0;       // store y data from serial port
017: int posx = 0;
018: int posy = 0;
019: int incx = 10;  // significant increments of horizontal (x) camera movement
020: int incy = 10;  // significant increments of vertical (y) camera movement
021:
022: Servo servox;
023: Servo servoy;
024:
025: short MSB = 0;  // to build  2 byte integer from serial in byte
026: short LSB = 0;  // to build  2 byte integer from serial in byte
027: int   MSBLSB = 0;  //to build  2 byte integer from serial in byte
028:
029: void setup() {
030:
031:   Serial.begin(baudrate);        // connect to the serial port
032:   Serial.println("Starting Cam-servo");
033:
034:   pinMode(servopinx,OUTPUT);    // declare the LED's pin as output
035:   pinMode(servopiny,OUTPUT);    // declare the LED's pin as output
036:
037:   servoy.attach(servopiny);
038:   servox.attach(servopinx);
039:
040:   // center servos
041:
042:   servox.write(servocenterx);
043:   delay(200);
044:   servoy.write(servocentery);
045:   delay(200);
046: }
047:
048: void loop () {
049:   while(Serial.available() <=0); // wait for incoming serial data
050:   if (Serial.available() >= 4)  // wait for 4 bytes.
051:   {
052:     // get X axis 2-byte integer from serial
053:     MSB = Serial.read();
054:     delay(5);
055:     LSB = Serial.read();
056:     MSBLSB=word(MSB, LSB);
057:     valx = MSBLSB;
058:     delay(5);
059:
```

```
060:    // get Y axis 2-byte integer from serial
061:    MSB = Serial.read();
062:    delay(5);
063:    LSB = Serial.read();
064:    MSBLSB=word(MSB, LSB);
065:    valy = MSBLSB;
066:    delay(5);
067:
068:    // read last servos positions
069:    posx = servox.read();
070:    posy = servoy.read();
071:
072:    //Find out if the X component of the face is to the left of the middle of the screen.
073:    if(valx < (screenmaxx/2 - incx)){
074:      if( posx >= incx ) posx += distancex; //Update the pan position variable to move the
servo to the left.
075:    }
076:    //Find out if the X component of the face is to the right of the middle of the screen.
077:    else if(valx > screenmaxx/2 + incx){
078:      if(posx <= servomaxx-incx) posx -=distancex; //Update the pan position variable to
move the servo to the right.
079:    }
080:
081:    //Find out if the Y component of the face is below the middle of the screen.
082:    if(valy < (screenmaxy/2 - incy)){
083:      if(posy >= 5)posy += distancey; //If it is below the middle of the screen, update the
tilt position variable to lower the tilt servo.
084:    }
085:    //Find out if the Y component of the face is above the middle of the screen.
086:    else if(valy > (screenmaxy/2 + incy)){
087:      if(posy <= 175)posy -= distancey; //Update the tilt position variable to raise the tilt
servo.
088:    }
089:
090:    // Servos will rotate accordingly
091:    servox.write(posx);
092:    servoy.write(posy);
093:
094:  }
095: }
```

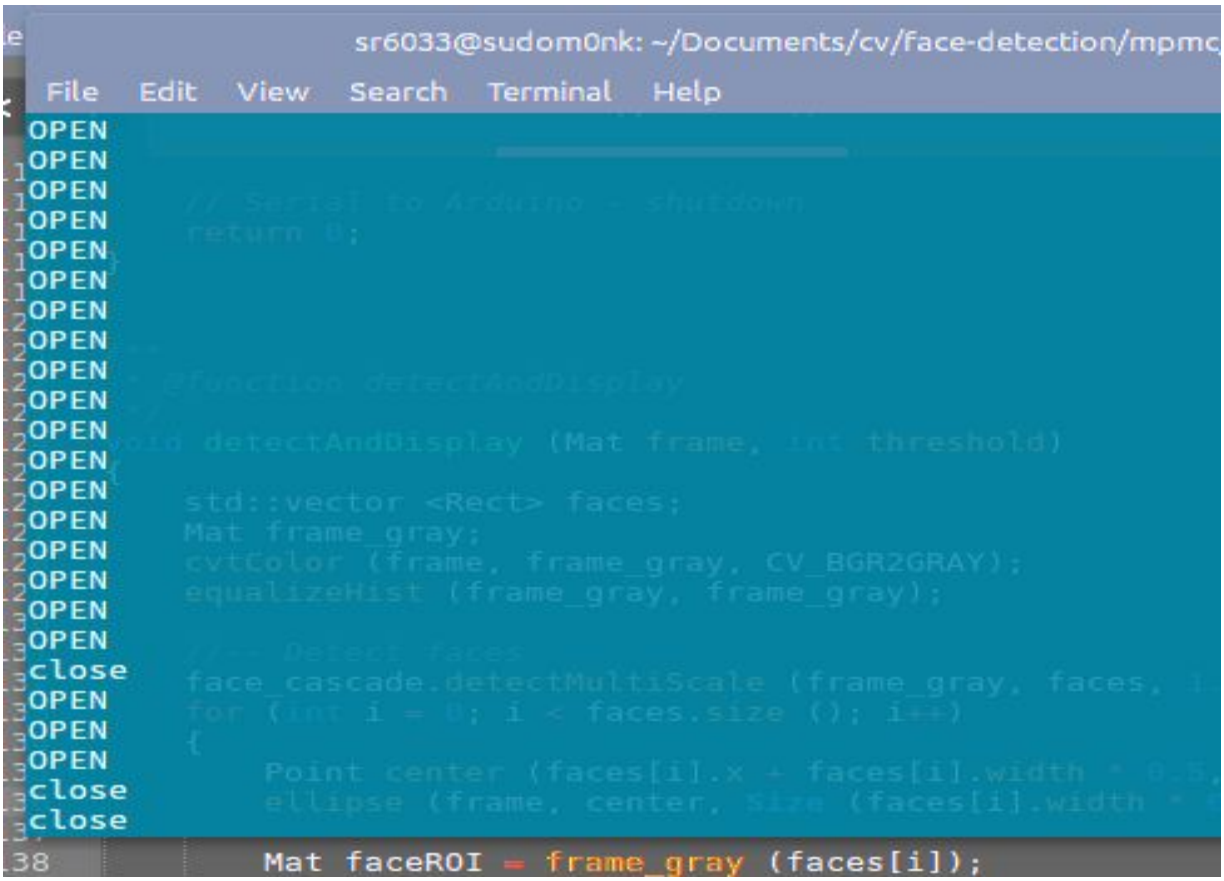
Implementation

- The c++ code is compiled using the following command:
  - *g++ `pkg-config --cflags opencv` track.cpp `pkg-config --libs opencv`*
- The COM port is given access for serial communication by the following command:
  - *sudo chown sr6033 /dev/ttyUSB0*
- The arduino code is dumped in the board and then the execution of the program is done using the following command:
  - *./a.out /dev/ttyUSB0 3*

The face and eye detection



The eye blink detection



Conclusion and Future Scope

Our project is successfully detecting and tracking the face of a single person with no time delay in a realtime environment. And the movement of the camera as per the face movement is smooth without any break or errors. Although a large amount of work has been done in visual surveillance for humans and vehicles, many issues are still open and deserve further research, especially in the following areas.

A. Occlusion Handling

Occlusion handling is a major problem in visual surveillance. Typically, during occlusion, only portions of each object are visible and often at very low resolution. This problem is generally intractable, and motion segmentation based on background subtraction may become unreliable. To reduce ambiguities due to occlusion, better models need be developed to cope with the correspondence between features and body parts, and thus eliminate correspondence errors that occur during tracking multiple objects. When objects are occluded by fixed objects such as buildings and street lamps, some resolution is possible through motion region analysis and partial matching. However, when multiple moving objects occlude each other, especially when their speeds, directions and shapes are very close, their motion regions coalesce, which makes the location and tracking of objects particularly difficult. The self-occlusion of a human body is also a significant and difficult problem. Interesting progress is being made using statistical methods to predict object pose, position, and so on, from available image information. Perhaps the most promising practical method for addressing occlusion is through the use of multiple cameras. Implementing of occlusion free detection and tracking to our project will enhance the surveillance in realtime.

B. Fusion of 2-D and 3-D Tracking

Two-dimensional tracking is simple and rapid, and it has shown some early successes in visual surveillance, especially for low-resolution application areas where the precise posture reconstruction is not needed, e.g., pedestrian and vehicle tracking in a traffic surveillance setting. However, the major drawback of the 2-D approach is its restriction of the camera angle.

Compared with 2-D approaches, 3-D approaches are more effective for accurate estimation of position in space, more effective handling of occlusion, and high-level judgments about complex object movements such as wandering around, shaking hands, dancing, and vehicle overtaking. However, applying 3-D tracking requires more parameters and more computation during the

matching process. Also, vision-based 3-D tracking brings a number of challenges such as the acquisition of object models, occlusion handling, parameterized object modeling, etc.

In fact, the combination of 2-D tracking and 3-D tracking is a significant research direction that few researches have attempted. This combination is expected to fuse the merits of the 2-D tracking algorithms and those of the 3-D tracking algorithms. The main difficulties of this combination are:

- deciding when 2-D tracking should be used and when 3-D tracking should be used;
- how to initialize pose parameters for 3-D tracking according to the results from 2-D tracking, when the tracking algorithm is switched from 2-D to 3-D.

D. Combination of Visual Surveillance and Personal Identification

E. Behavior Understanding

One of the objectives of visual surveillance is to analyze and interpret individual behaviors and interactions between objects to decide for example whether people are carrying, depositing or exchanging objects, whether people are getting on or getting off a vehicle, or whether a vehicle is overtaking another vehicle, etc. Recently, related research has still focused on some basic problems like recognition of standard gestures and simple behaviors. Some progress has been made in building the statistical models of human behaviors using machine learning. Behavior recognition is complex, as the same behavior may have several different meanings depending upon the scene and task context in which it is performed.

F. Anomaly Detection and Behavior Prediction

Anomaly detection and behavior prediction are significant in practice. In applications of visual surveillance, not only should visual surveillance systems detect anomalies such as traffic accidents and car theft etc, according to requirements of functions, but also predict what will happen according to the current situation and raise an alarm for a predicted abnormal behavior.

References

1. S. J. Maybank, T. N. Tan, "Special section on visual surveillanceintroduction", *Int. J. Comput. Vis.*, vol. 37, no. 2, pp. 173-174, 2000.
2. R. T. Collins, A. J. Lipton, T. Kanade, "Introduction to the special section on video surveillance", *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 22, pp. 745-746, Aug. 2000.
3. A. Hilton, P. Fua, "Foreword: modeling people toward vision-based understanding of a person's shape appearance and movement", *Comput. Vis. Image Understanding*, vol. 81, no. 3, pp. 227-230, 2001.
4. C. Regazzoni, V. Ramesh, "Special issue on video communications processing and understanding for third generation surveillance systems", *Proc. IEEE*, vol. 89, pp. 1355-1367, Oct. 2001.
5. L. Wang, W. Hu, T. Tan, "Recent developments in human motion analysis", *Pattern Recognit.*, vol. 36, no. 3, pp. 585-601, 2003.
6. R. T. Collins, A. J. Lipton, T. Kanade, H. Fujiyoshi, D. Duggins, Y. Tsin, D. Tolliver, N. Enomoto, O. Hasegawa, P. Burt, L. Wixson, *A system for video surveillance and monitoring*, Carnegie Mellon Univ., 2000.
7. I. Haritaoglu, D. Harwood, L. S. Davis, " \${rm W}^{4}\$: Real-time surveillance of people and their activities ", *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 22, pp. 809-830, Aug. 2000.
8. C. R. Wren, A. Azarbayejani, T. Darrell, A. P. Pentland, "Pfinder: real-time tracking of the human body", *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 19, pp. 780-785, July 1997.
9. T. Olson, F. Brill, "Moving object detection and event recognition algorithms for smart cameras", *Proc. DARPA Image Understanding Workshop*, pp. 159-175, 1997.
10. A. J. Lipton, H. Fujiyoshi, R. S. Patil, "Moving target classification and tracking from real-time video", *Proc. IEEE Workshop Applications of Computer Vision*, pp. 8-14, 1998.
11. S. E. Kemeny, R. Panicacci, B. Pain, L. Matthies, E. R. Fossum, "Multi-resolution image sensor", *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, no. Aug., pp. 575-583, 1997.
12. T. Boult, "Frame-rate multi-body tracking for surveillance", *Proc. DARPA Image Understanding Workshop*, pp. 305-308, 1998-Nov.
13. A. Basu, D. Southwell, "Omni-directional sensors for pipe inspection", *Proc. IEEE Int. Conf. Systems Man and Cybernetics*, vol. 4, pp. 3107-3112, 1995.
14. D. Ross, J. Lim, R.-S. Lin, and M.-H. Yang. Incremental learning for robust visual tracking. IJCV, 2007.

15. M. J. Black and A. D. Jepson. EigenTracking: Robust matching and tracking of articulated objects using a view-based representation, 1996. ECCV.