HUMAN EMOTION RECOGNITION SYSTEM

Project Report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology
In
Information Technology
Under the Supervision of

Dr.Pooja Jain

By

Vivek Nagrath(091415)
Akshay Raj(091422)

to

Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

Certificate

This is to certify that project report entitled "HUMAN EMOTION RECOGNITION SYSTEM", submitted by Vivek Nagrath(091415) & Akshay Raj(091422) in partial fulfillment for the award of degree of Bachelor of Technology in Information Technology to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

Date:13 May 2013

Dr. Pooja Jain

Assistant Professor

ACKNOWLEDGEMENT

We owe a great many thanks to a great many people who have been helping and supporting us during this project. Our deepest thanks to Dr.Pooja Jain, the Guide of the project for guiding and correcting us at every step of our work with attention and care. She has taken pain to go through the project and make necessary correction as and when needed. Thanks and appreciation to the helpful people at college, for their support. We would also thank our Institution and my faculty members without whom this project would have been a distant reality. We also extend our heartfelt thanks to our family and well wishers.

SUMMARY

Objective

Affect recognition from body Gestures

Description of the work

Detection and recognition of the Human Body Composition and extraction their features (width, length and all measures of human body) in the Images are important field in Image, Signal and Vision Computing in recent years. We have tried to recognize body gestures in real time using human motion capturing techniques.
We have implemented hand and face tracking algorithms which track the movements of face and hands in real time. We intend to perform affect recognition from frame human body gestures.

Why we chose this topic?

Rapid growth in computer vision and image processing applications has been evident in recent years. One area of interest in vision and image processing is automated identification of objects in real-time or recorded video streams and analysis of these identified objects. An important topic of research in this context is identification of humans and interpreting their actions. Emotion recognition via body movements and gestures has only recently started attracting the attention of computer science and HCI communities.

Milestones and expected Results

- These are the milestones to be achieved for
- Silhouette outlining- using background subtraction
- Skin Segmentation- using skin color information
- Retrieving the contours- using Connected Component Analysis
- Tracking of individual body parts- using CamShift Tracking
- Shoulder detection and Tracking
- Building a classifier for affect recognition
- Training the classifier
- Affect Recognition from body gestures

List of Figures

List of Tables

Table of Contents

Chapter 1: INTRODUCTION

Chapter 1: INTRODUCTION

1.1 General

Gesture Recognition

Gesture recognition is a topic in computer science and language technology with the goal of interpreting human gestures via mathematical algorithms. Gestures can originate from any bodily motion or state but commonly originate from the face or hand. Current focuses in the field include emotion recognition from the face and body gesture recognition.

Gesture recognition can be seen as a way for computers to begin to understand human body language, thus building a richer bridge between machines and humans than primitive text user interfaces or even GUIs (graphical user interfaces), which still limit the majority of input to keyboard and mouse.

Gesture recognition enables humans to interface with the machine (HMI) and interact naturally without any mechanical devices. Gesture recognition is conducted with techniques from computer vision and image processing.

Uses

Gesture recognition is useful for processing information from humans which is not conveyed through speech or type. As well, there are various types of gestures which can be identified by computers.

- **Sign language recognition :** Just as speech recognition can transcribe speech to text, certain types of gesture recognition software can transcribe the symbols represented throughsign language into text.
- **For socially assistive robotics :** By using proper sensors (accelerometers and gyros) worn on the body of a patient and by reading the values from those sensors, robots can assist in patient rehabilitation. The best example can be stroke rehabilitation.
- **Directional indication through pointing :** Pointing has a very specific purpose in our[clarification needed] society, to reference an object or location based on its position

relative to ourselves. The use of gesture recognition to determine where a person is pointing is useful for identifying the context of statements or instructions. This application is of particular interest in the field of robotics.

- **Control through facial gestures :** Controlling a computer through facial gestures is a useful application of gesture recognition for users who may not physically be able to use a mouse or keyboard. Eye tracking in particular may be of use for controlling cursor motion or focusing on elements of a display.
- **Alternative computer interfaces :** Foregoing the traditional keyboard and mouse setup to interact with a computer, strong gesture recognition could allow users to accomplish frequent or common tasks using hand or face gestures to a camera.
- **Immersive game technology :** Gestures can be used to control interactions within video games to try and make the game player's experience more interactive or immersive.
- **Virtual controllers :** For systems where the act of finding or acquiring a physical controller could require too much time, gestures can be used as an alternative control mechanism. Controlling secondary devices in a car, or controlling a television set are examples of such usage.
- **Affective computing :** In affective computing, gesture recognition is used in the process of identifying emotional expression through computer systems.
- **Remote control :** Through the use of gesture recognition, "remote control with the wave of a hand" of various devices is possible. The signal must not only indicate the desired response, but also which device to be controlled.

1.2 Problem Statement

What are we going to do?

We have used body gesture recognition for affect recognition in our project. AFFECTIVE computing aims to equip computing devices with the means to interpret, understand, and respond to human emotions, moods, and, possibly, intentions without the user’s conscious orintentional input of information–similar to the way humans rely on their senses to assess each other’s affective state.

Chapter 2: LITERATURE SURVEY

Chapter 2: LITERATURE SURVEY

Background

The existing approaches for hand or body gesture recognition and analysis of human motion, in general, can be classified into three major categories: (1) model-based (i.e.,modeling the body parts or recovering 3-D configuration of articulated body parts); (2)appearance-based(i.e.,based on 2-Dinformation such as color/gray scale images or body silhouettes and edges); and (3)motion-based (i.e.,directly using the motion in formation without any structural information about the physical body).

Basic Procedure

Our work focuses on communicative affective gestures generated with one hand or two hands, head, shoulders, or combinations of these. The feature extraction, analysis, and tracking procedures presented in this section are applied on the videos obtained from the body camera only. The main steps can be summarized as follows:

- The static background model of the observed space is created before detection can start, the head region is extracted from the images using cascaded classifiers, skin colored regions are extracted from the images using skin-region segmentation and connected component labeling, and tracking of each ROI is obtained with the CamShift technique. We chose to use the CamShift tracker, because it is one of the best single-cue trackers, it is quite efficient and robust when the object color remains the same, and there is no similar color in the background. Compared to typical particle filters, its computational requirements are less intense.
- The body model employed is a combination of silhouette-based and color-based body models to determine the image location of the main body parts while the person is in a sitting posture. The height of the bounding box of the silhouette is taken as the height of the body model. Then, fixed vertical scales are used to determine the initial approximate location (boundingbox) of individual body parts. The height of the initial bounding box for the head is set to be two fifth times the body silhouette height. The height of the initial bounding box for the torso is set to be three fifth times the body

silhouette height. The width of the bounding boxes of the head and torso are calculated by finding the median width (horizontal linewidths) inside their initial bounding boxes. In addition to finding sizes and locations, the principal axis of the foreground Pixels inside the initial bounding boxes is computed in order to estimate the pose of the body parts.The torso is located first, followed by head, shoulders, and hands. For each video frame, the raw image is converted to a color probability distribution via the color histogram model of the skin region being tracked. CamShift calculates the centroid of the color probability distribution, recenters the window, and then calculates the area for the next window size. If the region cannot be tracked, the algorithm is reinitialized.

- When two hands merge or when the hands touch the facial region, due to their skin colors being similar, they are segmented as a single foreground region by the CamShift algorithm. The merged region is tracked until it splits back into its constituent objects (face and hands, or hand and hand). When the merged region splits, the localization procedure is run again to obtain and reinitialize the current location of each region. An important point to note is that, when the hands move closer to the head region, the CamShift algorithm operates under a special condition called “tracking in the presence of distractions”.
- We extract the ROI rectangles for the extracted body features, which it later uses to calculate the following: (1) General change within the feature (e.g., how the centroid, rotation, length, width, and area of the feature increased or decreased); (2) texture/motion; and (3) optical flow in this region with respect to a neutral frame.

Body feature extraction

In each frame a segmentation process based on a background subtraction method is applied in order to obtain the silhouette of the upper body. We then apply thresholding, noise cleaning and morphological filtering. After thresholding, one iteration of 3*3 dilation is applied on the binary image. Then, a binary connected component operator is used to find the foreground regions, and small regions are eliminated. Since the remaining region is bigger than the original one it is restored to its original size by the erosion procedure. We then generate a set of features for the detected foreground object, including its centroid, area, bounding box and expansion/contraction ratio for comparison purpose.

Segmentation and tracking of the body parts: We first locate the face and the hands exploiting skin color information. Among the detected candidate regions, the largest connected component gives the face region; the second and third largest connected components give the hands, respectively. We then calculate the centroid of these regions in order to use them as reference points for the body movement. We employ color since we need to detect the hands even if they are located within the silhouette. Hand displacement is computed as the motion of the centroid coordinate. We employ Camshift technique for tracking the hands and comparison of bounding rectangles is used to predict their locations in subsequent frames.

Locating shoulders: We locate shoulders based on the model knowledge of where they usually occur with respect to the face, upper body and hands. According to our upper-body model, in the neutral frame, shoulders are the widest point of the upper half of the silhouette. First, we compute the 1D horizontal projections of the silhouette. We then assume that most people present a narrower row in skin blob at neck level and a much wider row at the shoulder level, compared to the neck level. Thus, starting from the face centroid, we search for the widest row in the upper body blob. We also compute the 1D vertical projection of the silhouette and locate the shoulders as two minimums on left and right hand side of the bounding rectangle for the head. For recognizing "shoulder shrug", we compare the horizontal position of the shoulders with respect to the neutral frame.

Region merging: When two hands merge or when the hand(s) cover the face region due to their skin color, they might be segmented as one foreground region by the Camshift algorithm. Camshift applies a simple analysis of the predicted bounding boxes of the tracked objects and the bounding box of the detected foreground. When the merged region splits, the localization procedure is run again to obtain and re-initialize the current location of each region.

Hand pose and orientation estimation: Orientation feature helps to discriminate between different poses of the hand. On convergence, the Camshift algorithm returns orientation, length and width of the bounding rectangle for the hand, hence, enabling the estimation of hand rotation. Using this information we decide if the hand is in vertical or horizontal position. After estimating the initial pose of the hand it is possible to find out the position of the fingers. Edges have proven useful features for discriminating between different poses of the hand. We define four categories for finger position estimation: up, down, right and left. We use this information when classifying the feature vectors into various BAUs (e.g. arms crossed, hands touching the head etc).

Object Tracking using Camshift Algorithm

The CamShift Algorithm

The CamShift algorithm[5] can be summarized in the following steps :

1. Set the region of interest (ROI) of the probability distribution image to the entire image.
2. Select an initial location of the Mean Shift search window. The selected location is the target distribution to be tracked.
3. Calculate a color probability distribution of the region centered at the Mean Shift search window.
4. Iterate Mean Shift algorithm to find the centroid of the probability image. Store the zeroth moment (distribution area) and centroid location.
5. For the following frame, center the search window at the mean location found in Step 4 and set the window size to a function of the zeroth moment.

Continuously Adaptive Distributions

In order to generate the PDF, an initial histogram is computed at Step 1 of the CamShift algorithm[5] from the initial ROI of the filtered image. The histogram consists of the hue channel in HSV color space; however multidimensional histograms from any color space may be used. The histogram is quantized into bins, which reduces the computational and space complexity and allows similar color values to be clustered together. The histogram bins are then scaled between the minimum and maximum probability image intensities using following equation:

$$\{\hat{p}_u = \min\left(\frac{255}{\max(\hat{q})}\hat{q}_u, 255\right)\}_{u=1..m}$$

Histogram Back-Projection

The back-projection[5] of the target histogram with any consecutive frame generates a probability image where the value of each pixel characterizes probability that the input pixel belongs to the histogram that was used.

Given that m-bin histograms are used, we define the n image pixel locations $\{x_i\}_{i=1..n}$ and the histogram $\{\hat{q}\}_{u=1..m.}$ We also define a function c: $R^2 \rightarrow \{1...m\}$ that associates to the pixel at location x_i^* the histogram bin index $c(x_i^*)$. The unweighted histogram is computed as

$$\hat{q}_u = \sum_{i=1}^{n} \delta[c(x_i^*) - u]$$

In all cases the histogram bin values are scaled to be within the discrete pixel range of the 2D probability distribution image using

$$\{\hat{p}_u = \min\left(\frac{255}{\max(\hat{q})}\hat{q}_u, 255\right)\}_{u=1..m}$$

That is, the histogram bin values are rescaled from [0,max(q)] to the new range [0, 55], where pixels with the highest probability of being in the sample histogram will map as visible intensities in the 2D histogram back-projection image.

Mass Centre Calculation[5]

The mean location (centroid) within the search window of the discrete probability image computed in Step 3 is found using moments. Given that is the intensity of the discrete probability image at within the search window.

a) Compute the zeroth moment

   $M_{00} = \sum_x \sum_y I\ (x, y)$

b) Find the first moment for x and y

   $M_{10} = \sum_x \sum_y I\ (x, y)$

$M_{01} = \sum_x \sum_y I\ (x, y)$

c) Compute the mean search window location

$x_c = \frac{M_{10}}{M_{00}}$; $y_c = \frac{M_{01}}{M_{00}}$

Mean Shift Convergence Criteria

The Mean Shift component of the algorithm is implemented by continually recomputing new values of (x_c, y_c) for the window position computed in the previous frame until there is no significant shift in position. The maximum number of Mean Shift iterations is usually taken to be 10-20 iterations. Since sub-pixel accuracy cannot be visually observed, a minimum shift of one pixel in either of the x and y directions is selected as the convergence criteria. Furthermore, the algorithm must terminate in the case where M_{00} is zero, which corresponds to a window consisting entirely of zero intensity.

Shoulder detection and Tracking

After the face is located by the face detector, the range of possible shoulder position is set as a bounding box of 3 × 1.5 relative to the face size. The shoulder position is then detected by fitting a parabola to the nearby approximately horizontal edge using weighted Hough Transform.

The procedure is as follows:

1) Determine the searching region of the shoulders. After face is detected, the shoulders are assumed to be confined to the rectangle under the face center with 3 × 1.5 times of the face size.

2) Select shoulder candidate points. The candidate points is the top n maximum y-gradients from each column of the searching region, in total 3×n×face width points. The y-gradients are calculated by convolving the searching region with y-direction Sobel operator. We do not use x-gradients for two reasons: one is to avoid the disturbance of the y-direction edges which are not candidates of the shoulders; the other is to save computational cost which is critical for a real time system.

3) Vote by weighted Hough transform[2]. We use a weighted Hough Transform with the parameters a, b, c as its dimensions. The accumulator is updated for each candidate point. A small difference from the standard Hough transform is that the y-gradient associated with each candidate point is used as weight (instead of 1 for all candidates) to update the accumulator, so that the real edge points on the shoulder are emphasized.

4) Find the parabola parameters a, b, c. The value a, b, c associated with the highest peak in the accumulator is the estimated parabola parameters. Other local maxima are not considered in this paper because the parameterizing scheme (SP and TP) itself is able to smooth out most of the outliers.

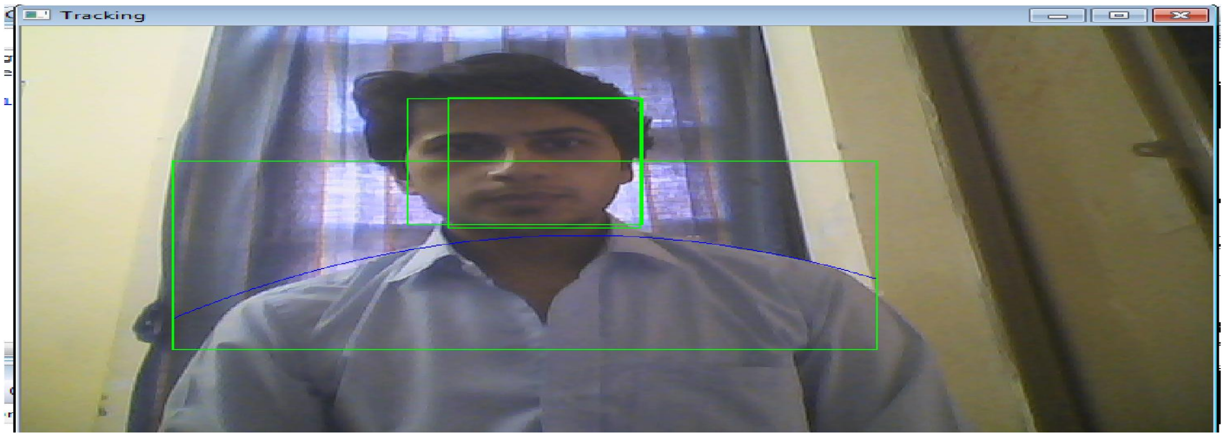


Fig 1: Weighted Hough Transform

(a) Searching region of the shoulder and candidate points

(b) Parabola fitted to the shoulders by weighted Hough Transform

Sobel Operator

The Sobel operator is used in image processing, particularly within edge detection algorithms. Technically, it is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel operator is either the corresponding gradient vector or the norm of this vector. The Sobel operator is based on convolving the image with a small, separable, and integer valued filter in horizontal and vertical direction and is therefore relatively inexpensive in terms of computations. On the other hand, the gradient approximation which it produces is relatively crude, in particular for high frequency variations in the image.

In simple terms, the operator calculates the gradient of the image intensity at each point, giving the direction of the largest possible increase from light to dark and the rate of change in that direction. The result therefore shows how "abruptly" or "smoothly" the image changes at that point, and therefore how likely it is that part of the image represents an edge, as well as how that edge is likely to be oriented. In practice, the magnitude (likelihood of an edge) calculation is more reliable and easier to interpret than the direction calculation.

Mathematically, the gradient of a two-variable function (here the image intensity function) is at each image point a 2D vector with the components given by the derivatives in the horizontal and vertical directions. At each image point, the gradient vector points in the direction of largest possible intensity increase, and the length of the gradient vector corresponds to the rate of change in that direction. This implies that the result of the Sobel operator at an image point which is in a region of constant image intensity is a zero vector and at a point on an edge is a vector which points across the edge, from darker to brighter values.

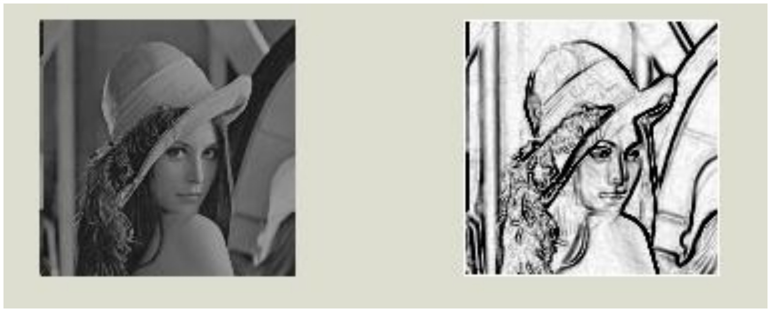


Fig 2: Detect edges using Sobel method

Hough Transform

The Hough Transform uses (xi, yi) points in the original 2D image space to generate

- (*rho, theta*) points in the Hough transform space for lines,
- (*x0, y0, radius*) points in the Hough transform space for circles,
- (*p, q, r1, r2, theta*) points in the Hough transform space for ellipses,
- and (*xv, yv, phi, p*) points in the Hough transform space for parabolas.

The Hough transform is a technique which is used to determine and isolate features of a particular shape within an image. The classical Hough transform requires that the desired shapes be specified in some parametric form. It is most commonly used for the detection of simple curves such as lines, circles, and ellipses within a given image.

The algorithm is as follows:

1. Define the range of angles [θ_b, θ_e] and the number M of sections in the range.
2. Split [θ_b, θ_e] into M sections. Let the split sections be θ_i , I ε{1,…,M}. Define the center of section θ_i as the representative value of the section.
3. Define the range of distances between the text line to be detected and the origin as [0,δ], where δ is the length of the diagonal line of the input image, and define the number of sections in the range as N.
4. Split [0, δ] by N. Let the split sections be $\rho_{i\ ,}$ I ε{1,…,M}, j ε{1,…,N} to zero.
5. Initialize the counters c (i, j), I ε {1,…,M}, j ε{1,…,N} to zero.
6. Choose a point (x, y) among the unprocessed sample points. If all sample points have been processed, jump to step 10.
7. Repeat the following process for all sections of angles θ in the detection range.
8. Increment the counter c(i,j) by 1 if the value of ρ given b

   $\rho = x \cos \theta_i + y \sin \theta_i$

   is included in section ρ_j .
9. Return to step 6.
10. Sort the counters c(i,j), i ε { 1,…,M}, j ε {1,…,N} in descending order.
11. The detection result is the most frequent angle θ_i in the highest Q counters.

The numbers of sections of angles, M, and of distance, N, are determined on the basis of the required range and resolution. The number of counters to be examined, 6, is set at close to the expected number of text lines.

Building a Classifier for Affect recognition

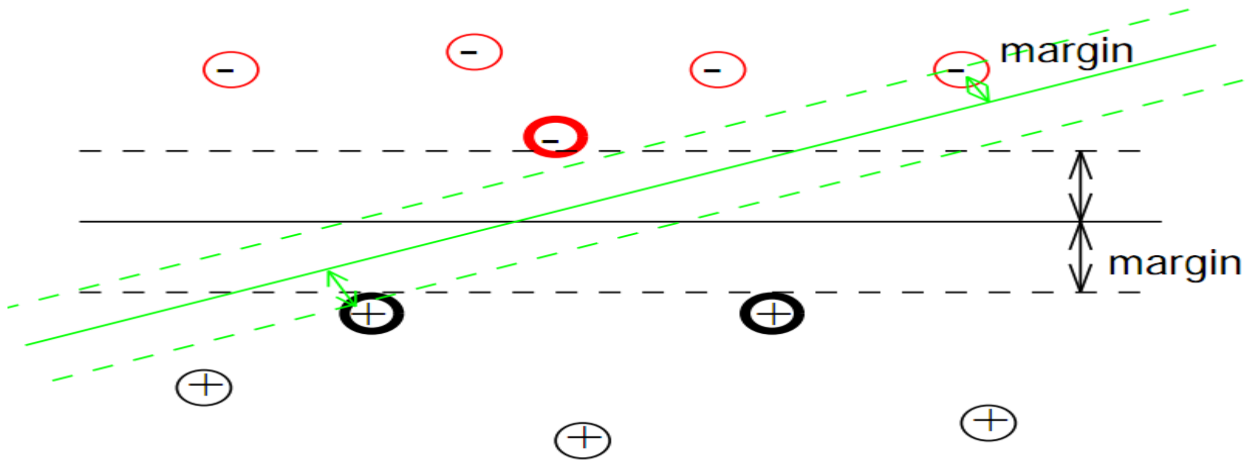
Classifiers which can be used for training and classification :

- SVM (Support Vector Machine)
- C4.5

SVM

Introduction

A Support Vector Machine (SVM) performs classification by constructing an N-dimensional hyperplane that optimally separates the data into two categories. The standard SVM takes a set of input data and predicts, for each given input, which of two possible classes the input is a member of, which makes the SVM a non-probabilistic binary linear classifier. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.



Implementation of Linear SVM

We are given some training data D, a set of n points of the form

D = $\{(x_i, y_i) \mid x_i \in R^n, y_i \in \{-1, 1\}\}_{i=1}^{n}$

where the y_i is either 1 or -1, indicating the class to which the point x_i belongs. Each x_i is a p-dimensional real vector. We want to find the maximum-margin hyperplane that divides the points having y_i=1 from those having y_i=-1. Any hyperplane can be written as the set of points x satisfying

$w.x - b = 0,$

Where '**.**' denotes the dot product. The vector w is a normal vector: it is perpendicular to the hyperplane. The parameter $\frac{b}{||w||}$ determines the offset of the hyperplane from the origin along the normal vector w.

We want to choose the w and b to maximize the margin, or distance between the parallel hyperplanes that are as far apart as possible while still separating the data. These hyperplanes can be described by the equations:

$w.x - b = 1,$

and

$w.x - b = -1.$

If the training data are linearly separable, we can select the two hyperplanes of the margin in a way that there are no points between them and then try to maximize their distance. By using geometry, we find the distance between these two hyperplanes is $\frac{2}{||w||}$, so we want to minimize ||w||. As we also have to prevent data points from falling into the margin, we add the following constraint: for each i either

$w.x - b \geq -1.$ For x_i of the first class

or

$w.x - b \leq -1.$ For x_i of the second.

This can be rewritten as:

$y_i(w.x - b) \geq -1,$ for all $1 \leq i \leq n$

We can put this together to get the optimization problem:

Minimize (in w, b)

||w||

subject to (for any i = 1,…,n)

y_i (w. x_i – b) ≥ 1.

Example:

Vs.

Suppose we have 50 photographs of elephants and 50 photos of tigers.

We digitize them into 100 x 100 pixel images, so we have $x \in R^n$ where n = 10,000

Now, given a new (different) photograph we want to answer the question:

Is it an elephant or a tiger? [We assume it is one or the other.]

Our aim is to learn the mapping: $X \rightarrow Y$, where $x \in X$ is some object and $y \in Y$ is a class label. Let's take the simplest case: 2-class classification. So: $x \in R^n$, $y \in \{ \pm 1 \}$.

Training Sets and prediction Models

- We provide training set $(x_1,y_1),\ldots,(x_m,y_m)$ for training.
- When training is performed, we give $x \in X$, and get $y \in Y$ as output where x is image vector and y is its class ($y \in$ { elephant, tiger }).
- We want to learn a classifier $y = f(x,\alpha)$, where α are the parameters of the function.
- If we are choosing our model from the set of hyperplanes in R^n, then we have:
  $f(x, \{w,b\}) = \text{sign}(w.x + b)$

C4.5

Introduction

C4.5 is an algorithm used to generate a decision tree developed by Ross Quinlan[1]. C4.5 is an extension of Quinlan's earlier ID3 algorithm. The decision trees generated by C4.5 can be used for classification, and for this reason, C4.5 is often referred to as a statistical classifier.

Algorithm

C4.5 adopts a greedy (i.e., non backtracking) approach in which decision trees are constructed in a top-down recursive divide-and-conquer manner.

C4.5 builds decision trees from a set of training data in the same way as ID3, using the concept of information entropy. The training data is a set S = s1,s2,... of already classified samples. Each sample si = x1,x2,... is a vector where x1,x2,... represent attributes or features of the sample. The training data is augmented with a vector C = c1,c2,... where c1,c2,...represent the class to which each sample belongs.

At each node of the tree, C4.5 chooses one attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. Its criterion is the normalized information gain (difference in entropy) that results from choosing an attribute for splitting the data. The attribute with the highest normalized information gain is chosen to make the decision. The C4.5 algorithm then recurs on the smaller sublists.

Algorithm: Generate_decision_tree. Generate a decision tree from the training tuples of data partition D.

Input:

- Data partition, D, which is a set of training tuples and their associated class labels;
- attribute list, the set of candidate attributes;
- Attribute_selection_method, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a splitting attribute and, possibly, either a split point or splitting subset.

Output: A decision tree.

```
Method:Create a node N;

If tuples in D are all of the same class, C then

        Return N as a leaf node labeled with the class C;

If attribute list is empty then

        Return N as a leaf node labeled with the majority class in D; // majority voting

Apply Attribute selection method (D, attribute list) to find the "best" splitting criterion;

Label node N with splitting criterion;

If splitting attribute is discrete-valued and
```

```
        multiway splits allowed then
        attribute_list ← attribute_list - splitting attribute; // remove splitting attribute
For each outcome j of splitting criterion
//partition the tuples and grow subtrees for each partition
        Let $D_j$ be the set of data tuples in D satisfying outcome j; // a partition
        If $D_j$ is empty then
                Attach a leaf labeled with the majority class in D to node N;
        Else attach the node returned by Generate_decision_tree ($D_j$, attribute_list) to node N;
Endfor
return N;
```

Chapter 3: Analysis, Design, and Modeling

3.1 Overall Description of the project

a) Product Perspective

The project is an attempt to use techniques available for body gesture recognition for real time and classification of emotions with high accuracy rate .For this purpose we have studied various algorithms (discussed above) and experimented with various libraries available.

b) Product Functions

- Taking video of face and upper body with the help of camera.
- Tracking the face and upper body movements and outline detection of face, hands, shoulders arms and torso.
- Using the 2D frames from the video for feature extraction, temporal segment detection and finally emotion classification and recognition.

c) User Characteristics

The project does not have any constraints on who can use it; it can be used by anyone and everyone, from researcher, teacher, and student to developer any one can use the product in accordance to his needs or build application on it. The user can be any human body facing the camera with its upper part in camera view and face and hands uncovered. The background should be of any color other than skin color.

d) Design and Implementation Constraints

- All the programming code is written in Open CV 2.1.0.
- Visual Studio is required for the program to run.
- A decent webcam is required and should not be used in darkness.
- The project is supposed to run in both Windows and Linux environment.

e) Assumptions and dependencies

- The background should be of any color other than skin color.
- The face and hands should be uncovered for detection.

3.2 Specific Requirements

3.2.1 External Interfaces

User Interface: User Interface will have one start button to start emotional classification displaying on frame basis emotions displayed by user. There will be a stop button to stop the above process.

Hardware Interface: There is a webcam + computer interaction the only hardware used is webcam.

Software Interface: We have used OpenCv library and the programming language is “C”. Visual Studio ir required to run the application.

Communication Interface: WebCam is the only communication interface.

3.2.2 Functions

- Trainer: take images from database and use them to find feature points and store emotion to train the classifier for emotion detection .
- The main program: takes in live feed from webcam and detects upper body (face, hands, shoulders and torso) and classifies emotions. The next frame shot and constrained windows are used for searching based on trained values

3.2.3 Performance Requirements

The basic requirement for running of the application is

- Webcam connected
- OpenCV installed

Dynamic requirements

- Memory Available

3.2.4 Design Constraints

- The results are better on more efficient computers as the search process gets fast.

- Constantly (the faster the better) moving objects are better segmented from the background.

3.2.5 Software System Attributes

a) Reliability: Result should be reliable as the application further working depends on it. Feature vector points have to be accurate for best working of the application.

b) Robust: As it takes lots of computation and memory resources it's a necessary point that software must be a robust one and free memory resources used and gradually degrade there by not annoying the user .

d) Maintenance: Training is an important phase for Classification for better and efficient classification one needs to have a very rich database so that it covers nearly all the gestures that can be generated by all varied body postures.

e) Portability: The software is portable, i.e. can be installed anywhere

3.3 Design

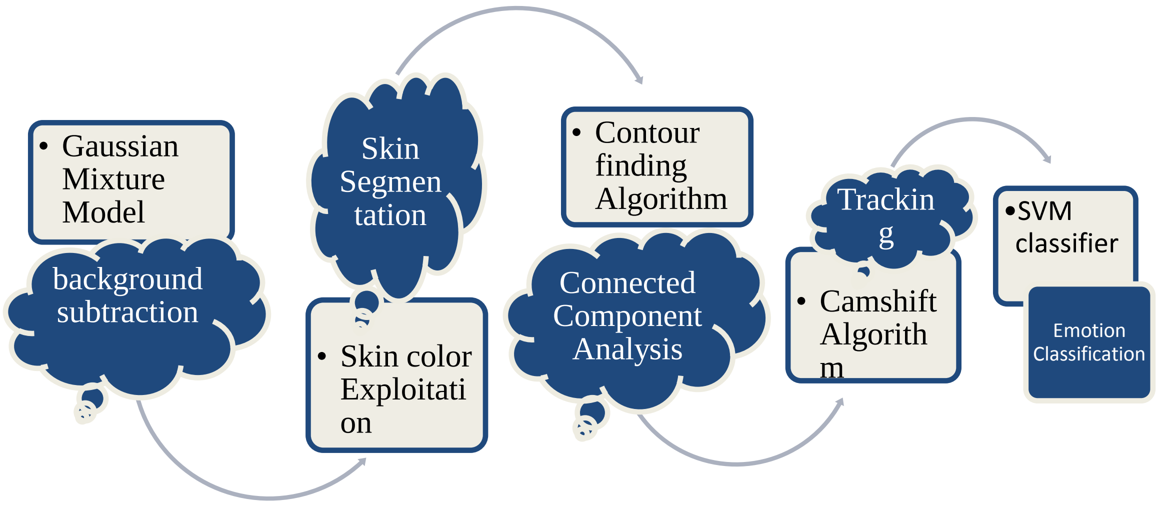


Fig 4. System Design

3.4 Modelling

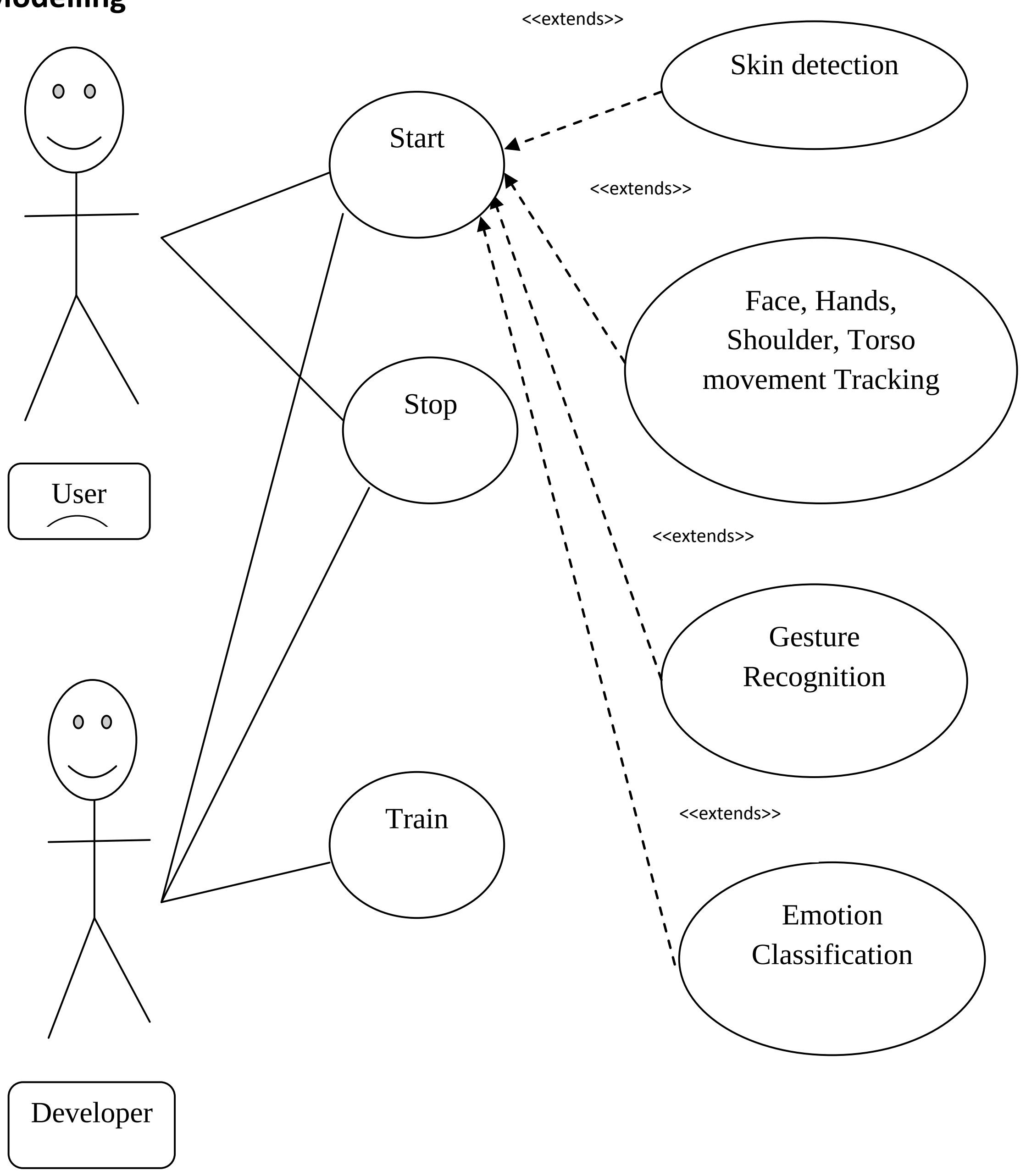


Fig5 Use Case Diagram

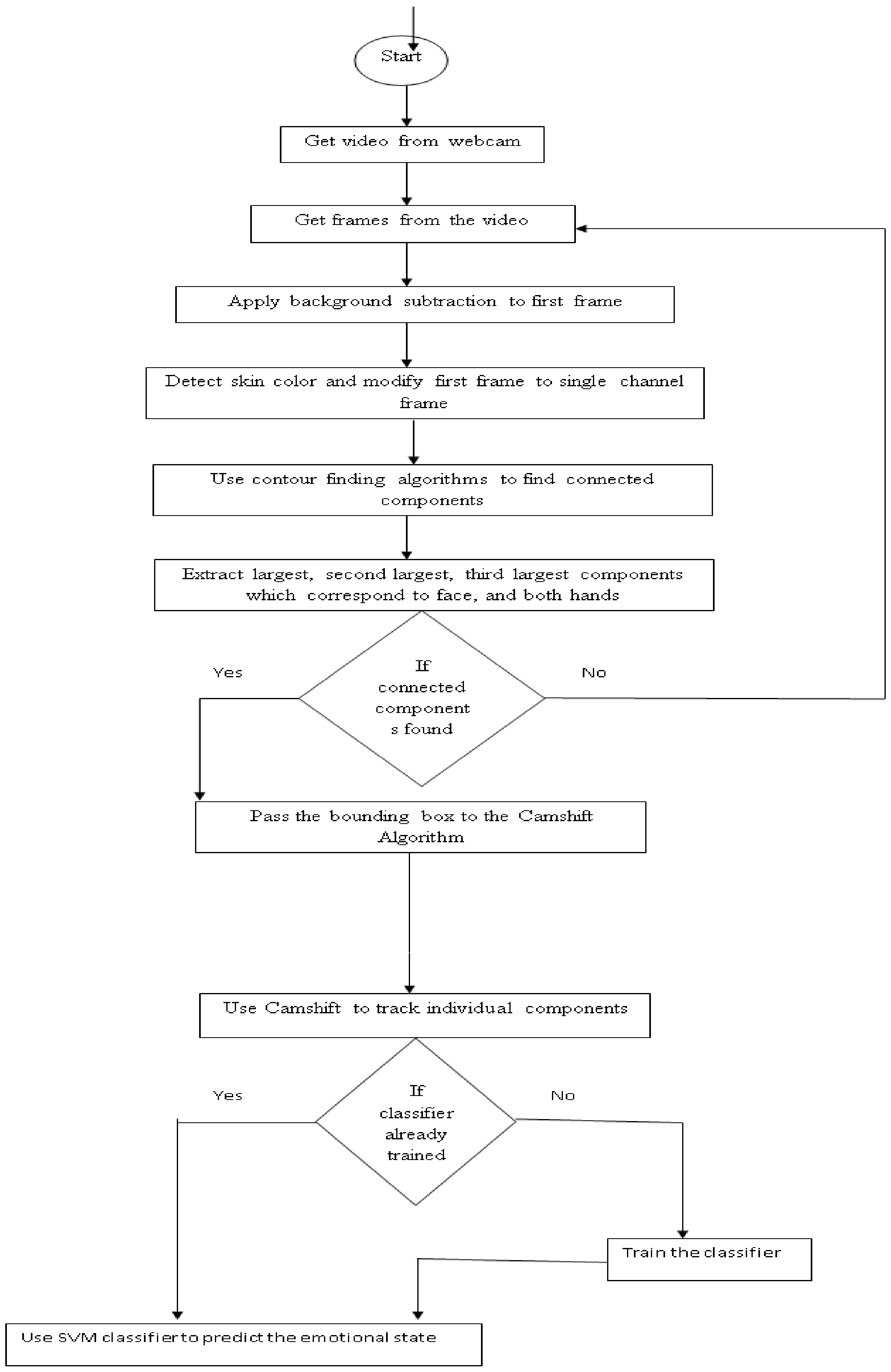

Start
Get video from webcam
Get frames from the video
Apply background subtraction to first frame
Detect skin color and modify first frame to single channel frame
Use contour finding algorithms to find connected components
Extract largest, second largest, third largest components which correspond to face, and both hands
If connected component s found
Yes
No
Pass the bounding box to the Camshift Algorithm
Use Camshift to track individual components
If classifier already trained
Yes
No
Train the classifier
Use SVM classifier to predict the emotional state

Fig6 Activity Diagram

Chapter 4 : Implementation

4.1 Implementation

4.1.1 Implementation

BACKGROUND SUBTRACTION (SILHOUETTE EXTRACTION)

This is the first step of our implementation.

Moving objects are classified as foreground objects and a gray scale image is produced which consists of foreground (white) and background (black) pixels. Assuming the user is in a sitting posture he/she is identified itself in the first step.

We have used adaptive background mixture model as the background subtraction algorithm.

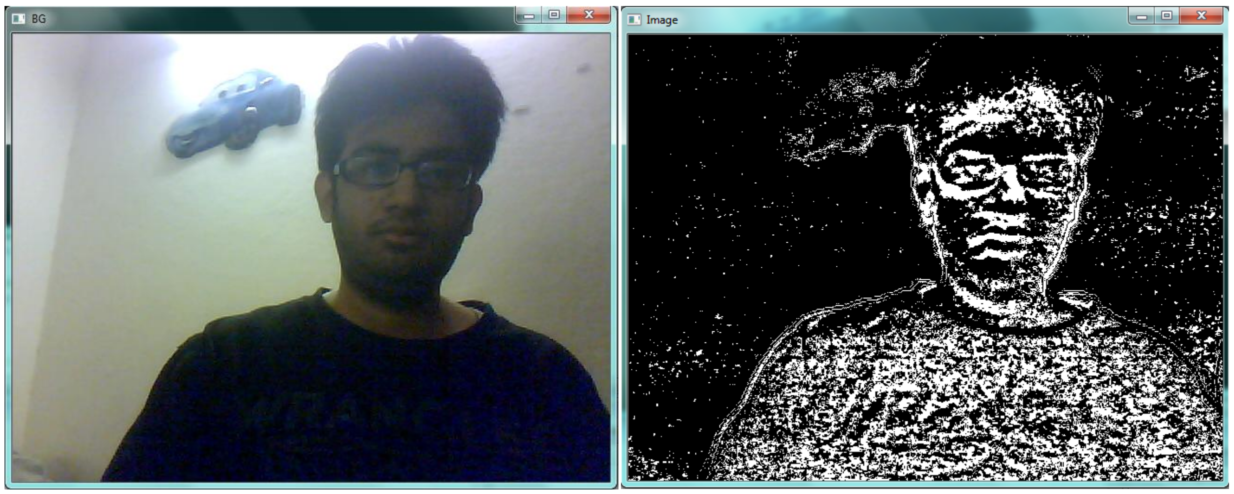


Fig 7. Background Segmentation

SKIN SEGMENTATION

Skin region was extracted from the silhouette exploiting color information. The extracted region included face and hands of the user.

Skin region is extracted by using the Y, Cr, Cb values of every pixel and checking if they lie within a standard region which can be expected to be a skin.

The image can be eroded and dilated as per the requirement.

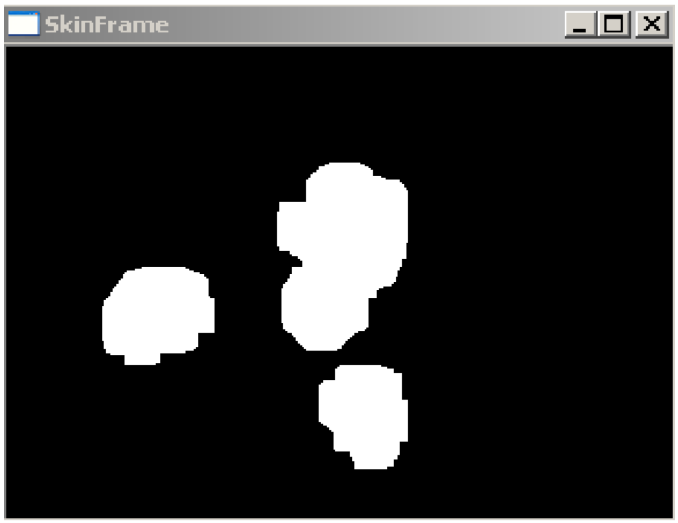
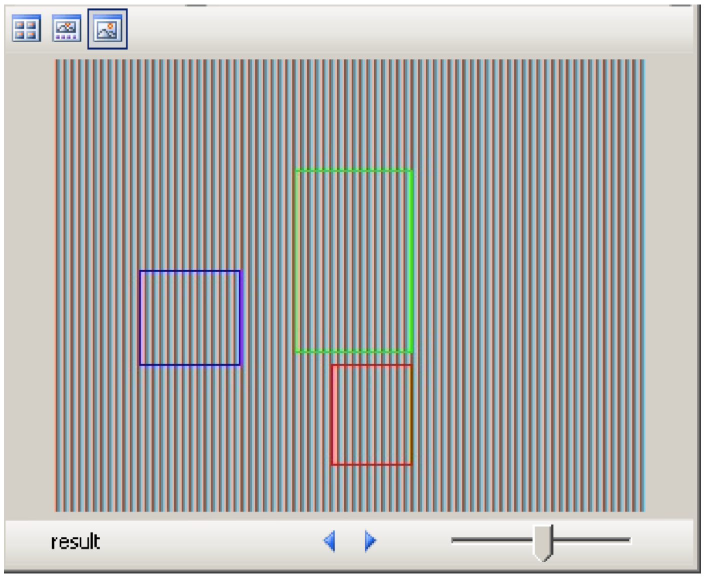
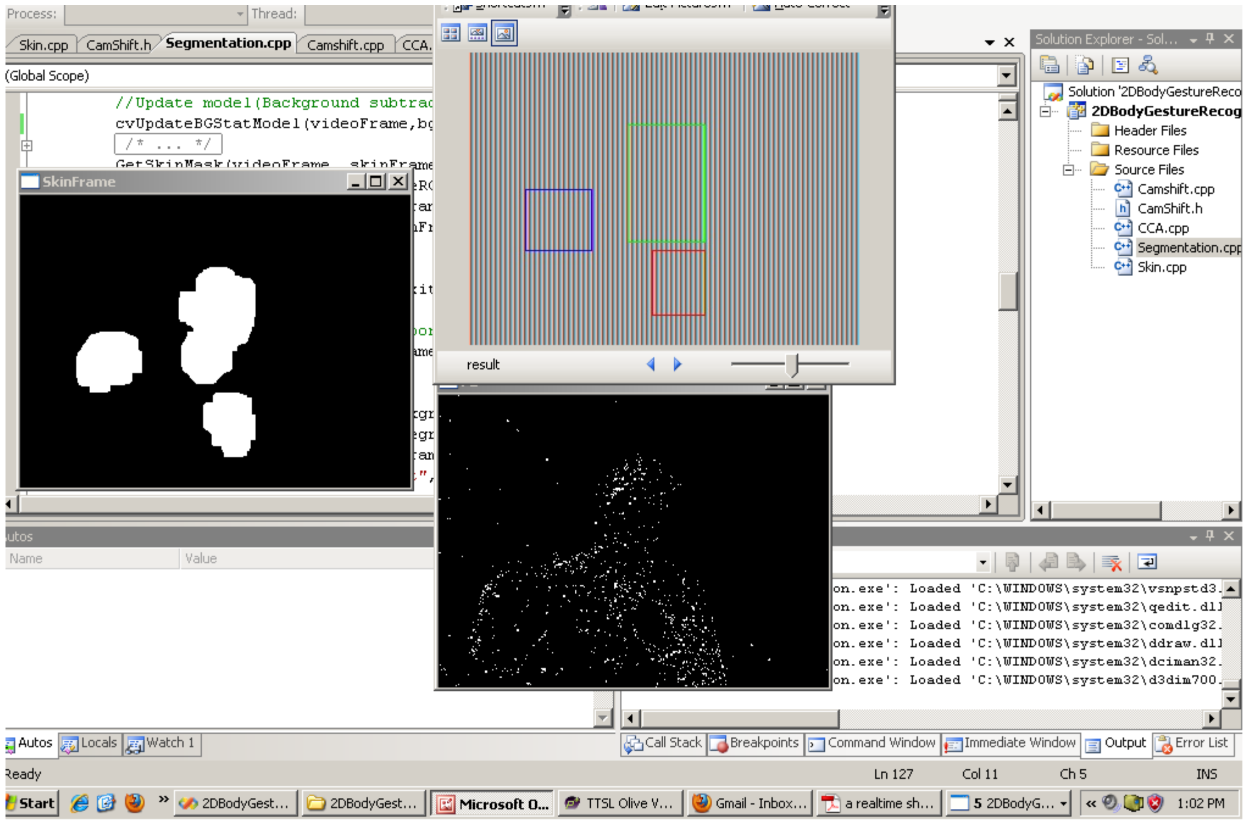


Fig 8. Skin Segmentation

CONNECTED COMPONENT ANALYSIS

- Connected component analysis is done on the extracted skin region. First the image is thresholded and then contours are extracted from the image.
- We assign a bounding box to each contour and then compare the size of the bounding box to calculate the three largest components.
- Among the detected candidate regions, the largest connected component gives the face region; the second and third largest connected components give the hands, respectively.





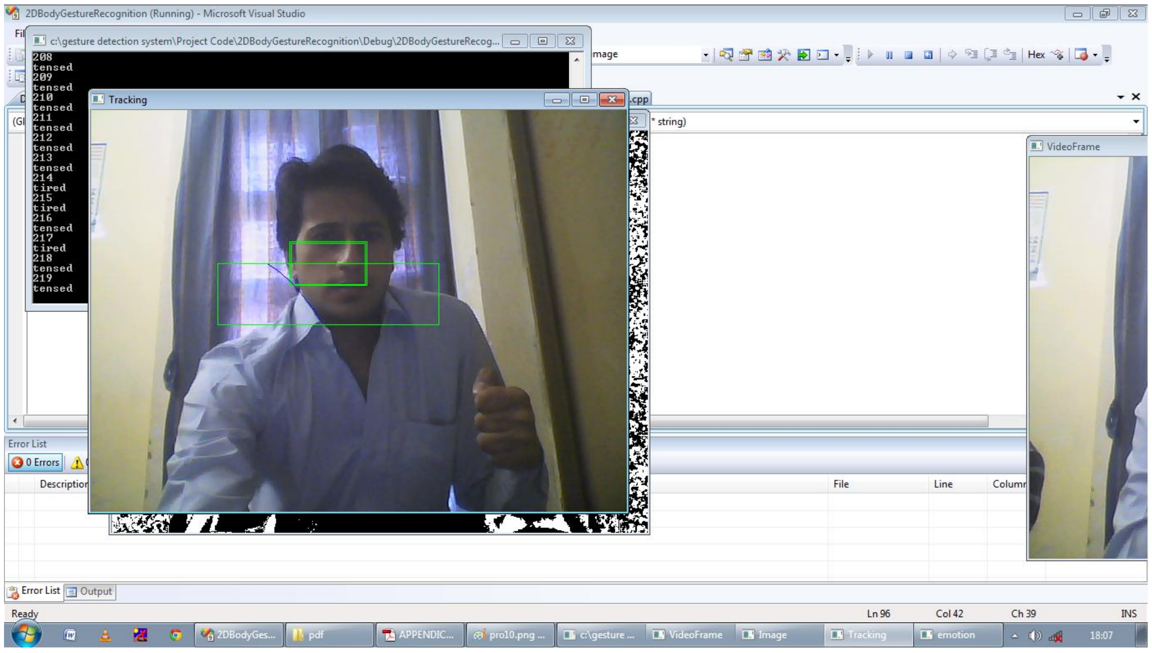
```
//Update model(Background subtrac
cvUpdateBGStatModel(videoFrame,bg
/* ... */
GetSkinMask(videoFrame, skinFrame
```

```
on.exe': Loaded 'C:\WINDOWS\system32\vsnpstd3.
on.exe': Loaded 'C:\WINDOWS\system32\qedit.dll
on.exe': Loaded 'C:\WINDOWS\system32\comdlg32.
on.exe': Loaded 'C:\WINDOWS\system32\ddraw.dll
on.exe': Loaded 'C:\WINDOWS\system32\dciman32.
on.exe': Loaded 'C:\WINDOWS\system32\d3dim700.
```

Fig 9. Connected Component Analysis

TRACKING

These contours are passed onto the tracker which represents each component in a histogram format to apply camshift tracking. The connected components just gives the camshift the initial Bounded boxes of face and hands and then camshift tracks these components using the camshift algorithm described in Chapter 2.


2DBodyGestureRecognition (Running) - Microsoft Visual Studio
c:\gesture detection system\Project Code\2DBodyGestureRecognition\Debug\2DBodyGestureRecog...
Tracking
VideoFrame
Error List
0 Errors
Description
File
Line
Column
Output
Ready
Ln 96
Col 42
Ch 39
INS
18:07

```
208
tensed
209
tensed
210
tensed
211
tensed
212
tensed
213
tensed
214
tired
215
tired
216
tensed
217
tired
218
tensed
219
tensed
```

Fig 10. Tracking

SHOULDER DETECTION & TRACKING

- ✓ The next step was to detect shoulders from the video frame and track their position. For this we first defined a region of Interest where shoulders are assumed to be confined. This is a rectangle under the face center with 3 X 1.5 times the face size.
- ✓ We then extracted shoulder candidate points which are the points with y- gradients more than a certain threshold value. These y- gradient points are calculated by convolving the shoulder ROI with the y- direction sobel operator.
- ✓ Then we apply Weighted Hough Transform technique to detect the shoulder from the candidate points. We use Hough Transform with a, b, c as its dimensions and the accumulator is updated for each candidate point. A small difference from the standard Hough transform is that the *y*-gradient associated with each candidate point is used as weight (instead of 1 for all candidates) to update the accumulator, so that the real edge points on the shoulder are emphasized.
- ✓ The value *a*, *b*, *c* associated with the highest peak in the accumulator is the estimated parabola parameters.

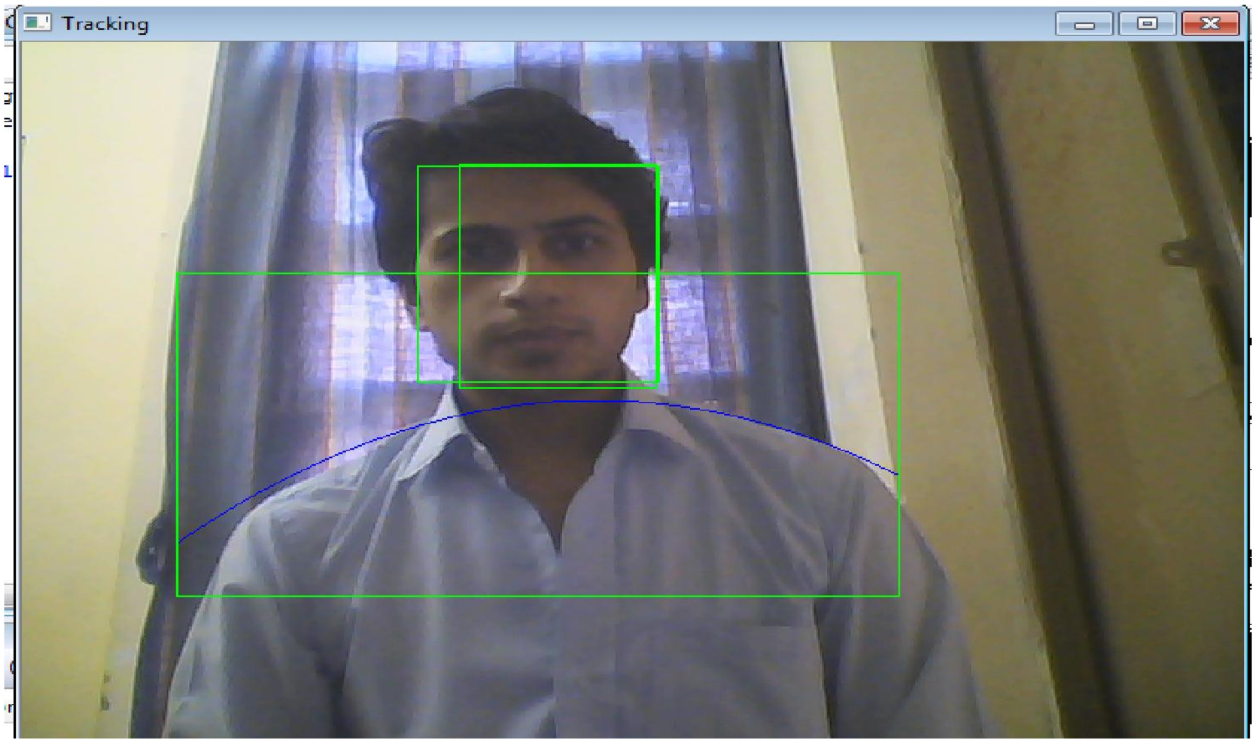


Fig 11. Shoulder Detection and tracking

EMOTION DETECTION AND CLASSIFICATION

We implemented a **SVM classifier** for emotion classification. We perform affect recognition by **frame based classification**.

SVM Training

The parameters used for classification are as follows:

- Shoulder Parameters:
  - a, b, c of shoulder equation: $ax^2 + bx + c$
- Distance of face centroid from hand1 centroid.
- Distance of face centroid from hand2 centroid.

The SVM tracker is trained to detect the following emotions:

- Neutral
- Tiredness
- Puzzlement

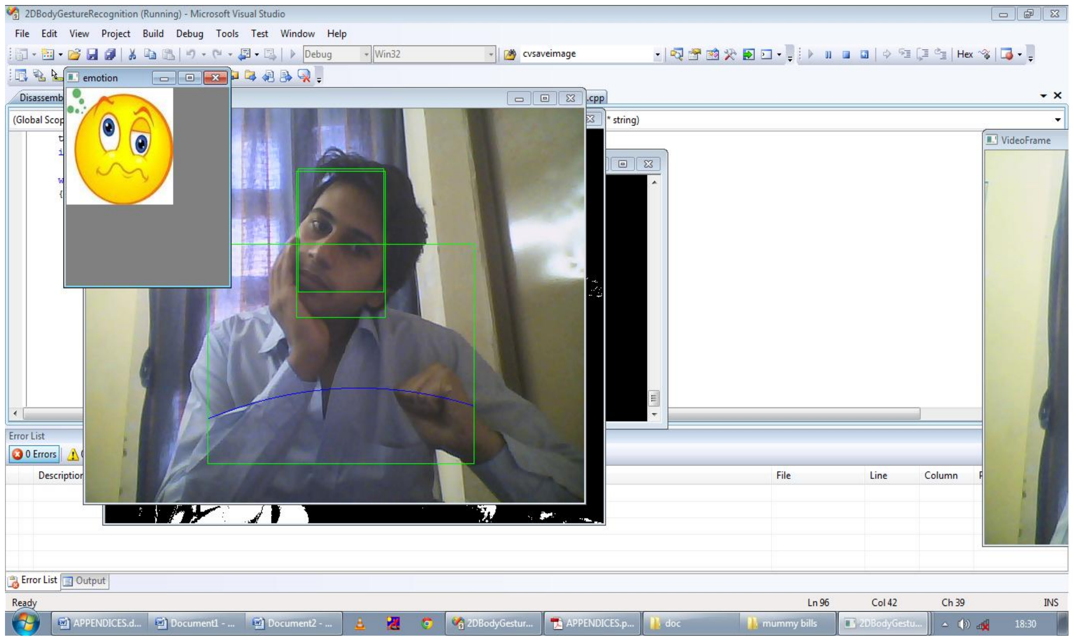
- Uncertainty

The SVM tracker is trained for 50 tuples of each type of emotion. All these tuples are stored in a file “train.txt”. The svm-train function then uses this train file for training of the classifier.

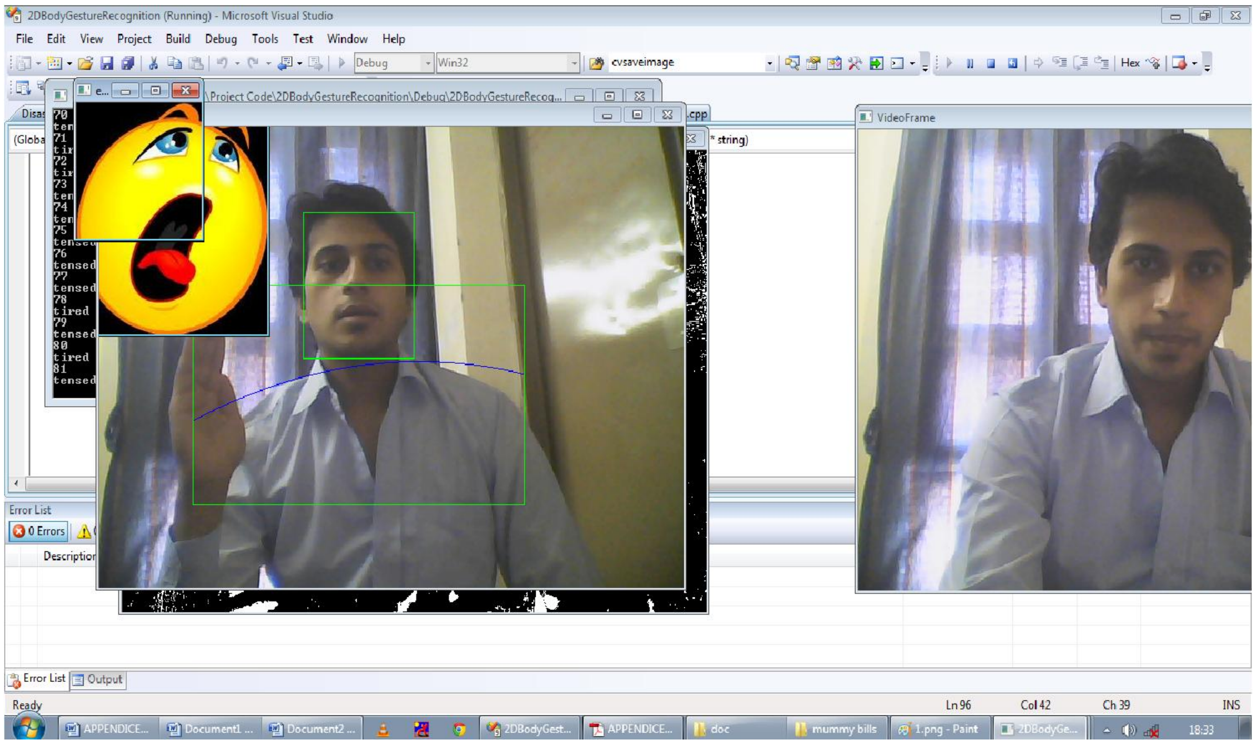
SVM Prediction

The subject has to sit in front of the camera and then the system after all the preprocessing steps the SVM classifier identifies the emotional state of the subject. The support vector is extracted from the current frame and then sends to the classifier algorithm where it is first parsed to extract the current values of the parameters.

It also displays an emoticon corresponding to the affective state of the subject.



(A)


2DBodyGestureRecognition (Running) - Microsoft Visual Studio
File Edit View Project Build Debug Tools Test Window Help
Debug
Win32
cvsaveimage
VideoFrame
Error List
0 Errors
Output
Ready
Ln 96
Col 42
Ch 39
INS

```
70
ten
71
tir
72
tir
73
ten
74
ten
75
tensed
76
tensed
77
tensed
78
tired
79
tensed
80
tired
81
tensed
```

(B)

Fig 12. Emotion Classification : (A) Tense (B) Tired

4.1.2 Debugging

Identify the Bug: A bug is unexpected and undesirable behaviour by a program. Some specific bugs encountered by us in our program were:

Bug#1: The working of background subtraction is not good as the functionality provided by the OpenCV platform is useful in detecting fast moving object and not sitting human beings.
Bug#2: Camshift algorithm sometimes looses track because of mergers of two or more components.
Bug#3: Sometimes only left or right shoulder is detected and the other one is not due to changes in luminosity of the scene.
*Bug#4***:** Inaccurate classification by SVM

Replicate the Bug*:* The first step in fixing a bug is to replicate it. This means to recreate the undesirable behaviour under controlled conditions. The goal is to find a precisely specified set of steps which demonstrate the bug. In many cases this is straightforward. You run the program on a particular input, or you press a particular button on a particular dialog, and the bug occurs.

Locate the bug: **We used the debugging functionality of Microsoft Visual Studio to locate several bugs in our code. We used the break point functionality to exactly locate the code area or the statement causing the undesirable behaviour. We also used the print statement of our coding language.**

Locate the error: **Once we have found the code which causes the bad behaviour, we need to identify the actual coding error. Often they are the same code--that is, the coding error directly causes the bad behaviour.**

Fix the bug: ***The final step in the debugging process is, of course, to fix the bug.***

Bug#1: ***Fixed by developing a background subtraction algorithm and not using the functionality provided by OpenCV.***

Bug#2: Cannot be fixed.

Bug#3: bug active only in extreme cases. The program is made more robust by making it indifferent to changes in lightning.

Bug#4: most dangerous bug. This bug was due to difference in ranges of the values of parameters. This bug was fixed by scaling both training and testing data and using scaling parameters.

4.1.3 Error and Exception Handling

Error#1: Source and destination images not compatible.
This error is due to difference in number of channels of both the images fixed by accessing individual channels.
Error#2: Memory index out of range.
This error was addressed by dynamic allocation of memory.
Error #3:

4.2 Risk Management

Risk Statements

- **RS1:** Personnel and Performance: Irregularity in the work
- **RS2:** Performance: System as delivered might be hard to maintain
- **RS3:** Software Process: Time for implementing Hough Transform algorithm is high
- **RS4:** Software Process: Unpredictable Camshift algorithm
- **RS5:** Hardware and Performance: Insufficient processing speed on small memory systems
- **RS6:** Hardware and Performance: Dependence on camera quality
- **RS7:** External Input and Performance: Malfunction due to the effect of intensity of light and light effects along with the video input on Tracking algorithm
- **RS8:** Budget (Time / Cost) : Schedule: Total Time of 1 year
- **RS9:** Budget (Time / Cost) : Since this is an academic project, Budget is limited
- **RS10:** Budget: Requirements are only partly known, when the project starts
- **RS11:** Project Scope: Features listed may be beyond what team can build in time
- **RS12**: Project Scope: Operating System Dependent
- **RS13:** Personnel: Team members don't agree for the same thing during project

Risk of our project as follows:

- **Performance:** Every risk is related to the performance of the project. For example, if there is problem with skewed data or the time taken for clustering and some of the personnel issues also affect the performance of the project.
- **Budget:** A high budget would helps to get access to latest updated hardware. If the hardware used is good, the performance of the system automatically increases. For example, the camera used is not of very good quality which limits project performance.
- **External Input:** The input video should satisfy the necessary constraints in order for the project to function efficiently.
- **Hardware**: Our project is highly hardware dependent. It requires efficient Camera and system with high memory capacity and processing speed.
- **Personnel:** Every project is very dependent on personal skills of people who implement it.
- **Project Scope**

- **Software Process**

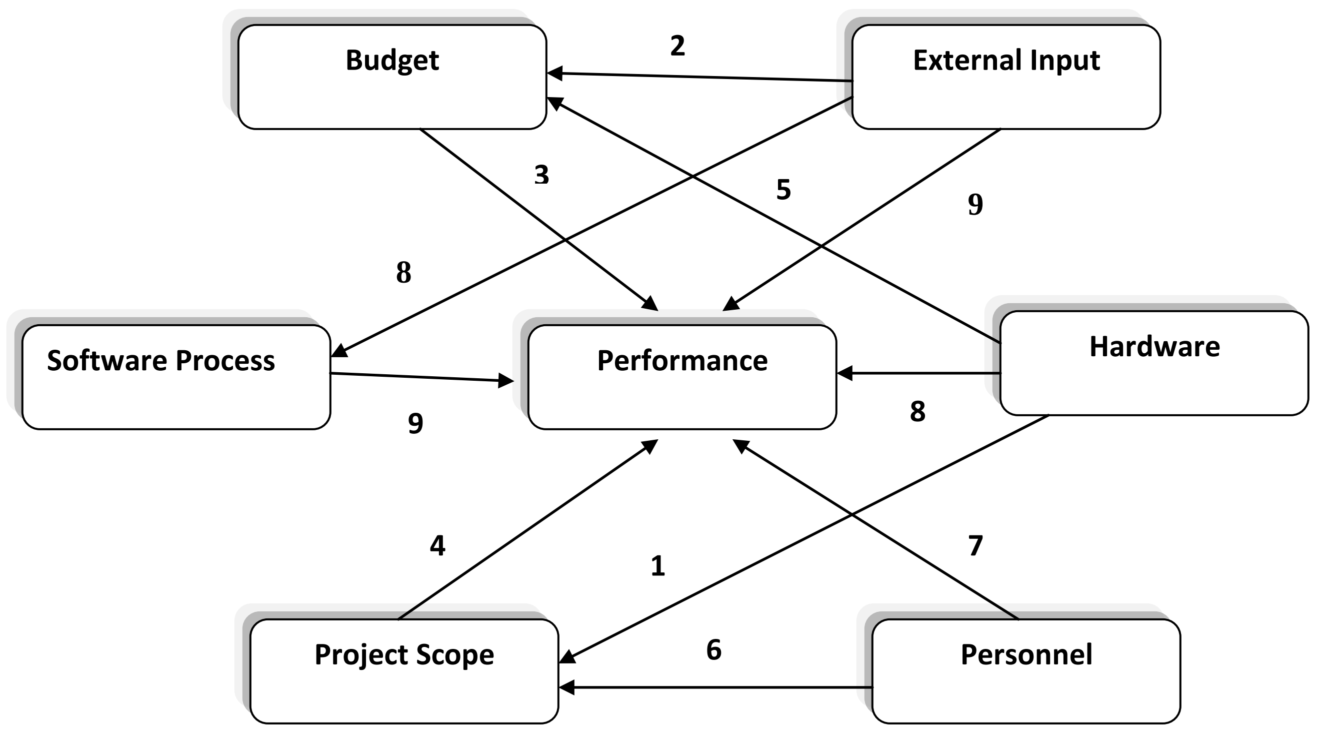


Fig 13. Weighted Interrelationship Graph of Project Risks

Risk Area Wise Total Weighting Factor

S. no.	**Risk Area**	**# of Risk Statement**	**Risk Statements**	**Weights (In + Out)**	**Total Weight**	**Priority**
1	Performance	5	RS1,RS2,RS5, RS6,RS7	3+9+8+7+4+9	40	1
2	External Input	1	RS7	2+8+9	19	2
3	Software Process	2	RS3,RS4	8+9	17	3
4	Hardware	2	RS5,RS6	5+8+1	14	4
5	Personnel	2	RS1,RS13	7+6	13	5
6	Budget	3	RS8,RS9,RS10	2+5+3	10	6
7	Project Scope	2	RS11,RS12	1+4	5	7

Risk Statement Table

Risk ID	**Risk Statement**	**Risk Area**	**Priority of Risk Area in IG**
1	Irregularity in the work	Personnel and Performance	1 and 5
2	System as delivered might be hard to maintain	Performance	1 and 5
3	Time for implementing Hough Transform algorithm is high	Software Process	3
4	Unpredictable Camshift algorithm	Software Process	3
5	Insufficient processing speed on small memory systems	Hardware and Performance	4 and 1
6	Dependence on camera quality	Hardware and Performance	4 and 1
7	Malfunction due to the effect of intensity of light and light effects along with the video input on Tracking algorithm	External Input and Performance	2 and 1
8	Schedule: Total Time of 5 months	Budget (Time / Cost)	6
9	Since this is an academic project, Budget is limited	Budget (Time / Cost)	6
10	Requirements are only partly known, when the project starts	Budget	6
11	Features listed may be beyond what team can build in time	Project Scope	7
12	Operating System Dependent	Project Scope	7
13	Team members don’t agree for the same thing during project	Personnel	5

4.3 Testing

4.3.1 Test Plan Identifier

- Webcam working properly
- Real time face tracking
- Real time face and hands tracking
- Handling Mergers
- Accurate Emotion Classification
- Accurate output

4.3.2 Test Items

- Camera
- Memory
- Human present
- Face in the frame
- Face and hands in the frame
- Real time movement tracking
- Pose variance
- Light variance
- Shrug
- Mood/Emotion

4.3.3 Features to be tested

- Proper skin segmentation
- Proper contour extraction
- Real time movement tracking
- Pose variance
- Shrug
- Accurate Emotion Detection
- Accurate output

4.3.4 Features not to be tested

- Camera
- Memory
- Human present
- Face in the frame
- Light variance

4.3.5 Test cases

Refer to Appendix C.

4.4 Results

The camshift algorithm could track face and hand movements in real time.

Shoulder detection algorithm could detect shoulder in real time.

SVM- classifier could classify and accurately recognize 5 emotions viz. – Neutral, Positive surprise, Uncertainty, Puzzlement/ Tension, Tiredness.

To calculate accuracy, We used 250 frames for training and 375 frames for testing.

Confusion Matrix

S. No.	**Emotions**	**Time (s)**	**No. of frames input**	**No. of frames / s**	**No. of frames correctly classified**
1	Neutral	10.26	80	7.79	76
2	Shrug/ Uncertainty	10.27	68	6.62	63
3	Tension	10.25	72	7.02	68
4	Tired	10.30	74	7.18	71

Calculating Accuracy

Neutral: 76/80 * 100 = 95 %

Shrug/ Uncertainty: 63/68 * 100 = 92.64 %

Tension: 68/72 * 100 = 94.44%

Tired: 71/74 * 100 = 95.94%

Chapter 5: Conclusion and Future Work

5.1 Conclusion

The following are the findings of the work we have done so far:

- In case of computer vision projects it is often better to use optimized libraries rather than writing codes our self.
- Even small differences in the face orientation can cause significant change in the detection process.
- It is a redundant to apply light dependant algorithms like optical flow because light independent algorithms already cover all the conditions.
- Light dependant algorithms are very unpredictable.
- The background and environment of the user can significantly hamper the detection process if not controlled properly.
- Stauffer's Background subtraction available in OpenCV works well for constantly moving objects.
- Tracking algorithms are very computationally expensive and hence have limited utility in real-time applications, for example, the "camshift" algorithm is the least computationally expensive and still is managed to hang our system several times.
- For an automated system based on vision, it is easier to model and recognize affective states from global body and head region movements and their relationship between each other.
- The proposed shrug detector is able detect the shrug action correctly and efficiently, but sometimes it fails to tolerate the large in-class variation caused by different subjects different action speed, illumination, partial occlusion, and background clutter. Therefore the proposed real-time shrug detector is promising in video analysis under a controlled environment.
- The real-time processing ability of the system based on the proposed architectural framework opens up its applicability to a wide range of applications where the proposed framework can be customized based on specific performance requirements in speed of processing, noise tolerance, fuzzy interpretation, etc.
- This work presented preliminary results of a vision-based system that extracts features automatically from expressive upper-body display. This is an attempt towards recognizing expressive face and upper-body gesture for affective Human Computer Interaction.
- CamShift can be used for general-purpose object tracking using a background- weighted histogram and arbitrary quantized color features of the target.

- An interface analyzing body gestures will find use in a range of areas such as video surveillance, monitoring of human activity and virtual environments and help in transmitting video for teleconferencing and improve man-machine interaction. However, due to being a fairly new research area, there still exist problems to be solved and issues to be considered in order to develop a robust, adaptive, context -sensitive analyzer of body gestures using computer vision and machine learning techniques.

5.2 Future Work

- Following constraints are there in the project which need to be removed:
  - Presence of a dark background behind the human body.
  - Absence of noisy circumstances for camshift to track properly.

  Consequently, future investigations towards more robust detection and tracking of the upper body and hands, possibly by using dynamic programming for tracking, which also allows partial trackbacks (e.g., one or two frames),are necessary yet possible.

- In natural Human Computer Interaction settings, gestures are continuous. Due to this, gesture recognition requires spotting of the gesture (i.e., determining the start and endpoints of a meaningful gesture pattern from a continuous stream or time segmentation. Our system currently does not perform spotting of the gesture due to the particular nature of the data at. However, the system could be extended to analyze a gesture continuum, determine the start and endpoints of a meaningful gesture pattern, and subsequently apply recognition.
- If dynamic cameras could be adopted, the range of applications the proposed framework can support will be even higher.
- The project can be extended to multimodal feature extraction by fusing body features with face features from monomodal feature extraction. Multimodal systems provide the possibility of combining different modalities that occur together to function in a more efficient and reliable way in diverse human-computer interaction applications.
- The proposed framework can be customized and optimized to various applications:

- Monitoring of public gathering places such as supermarkets, airports, sportsvenues, etc and identifying abnormal (or suspicious) behavior through gesture recognition and interpretation.
- Monitoring of hospital wards and in homes of disabled people to monitor them and taking care of their needs.
- Designing of safety systems for building such as factories. Assume a scenario where a fire breaks out in a factory and as it is quite difficult to individually notify all workers of details of the accident a warning alarm is used. To activate warning alarms and open safe exists, relevant operators will have to enter the building inspite of the fire. There may be instances where damage would occur very fast as in chemical factories before alarm is activated. In such scenarios, it will be extremely useful to have warning alarms and protective measures activated based on human gestures such as rapid waving the hands over the head while quickly running away from a location.

Chapter 6: Code Implementation

6.1 Camshift Analysis

Camshift.cpp

```
#include "cv.h"
#include <stdio.h>


// Parameters
int   nHistBins = 30;                 // number of histogram bins
float rangesArr[] = {0,180};          // histogram range
int vmin = 65, vmax = 256, smin = 55; // limits for calculating hue


// File-level variables
IplImage * pHSVImg  = 0; // the input image converted to HSV color mode
IplImage * pHueImg  = 0; // the Hue channel of the HSV image
IplImage * pMask    = 0; // this image is used for masking pixels
IplImage * pProbImg = 0; // the face probability estimates for each pixel
CvHistogram * pHist = 0; // histogram of hue in the original face image

IplImage * h1HSVImg  = 0; // the input image converted to HSV color mode
IplImage * h1HueImg  = 0; // the Hue channel of the HSV image
IplImage * h1Mask    = 0; // this image is used for masking pixels
IplImage * h1ProbImg = 0; // the hand probability estimates for each pixel
CvHistogram * h1Hist = 0; // histogram of hue in the original hand image

IplImage * h2HSVImg  = 0; // the input image converted to HSV color mode
IplImage * h2HueImg  = 0; // the Hue channel of the HSV image
IplImage * h2Mask    = 0; // this image is used for masking pixels
IplImage * h2ProbImg = 0; // the hand probability estimates for each pixel
CvHistogram * h2Hist = 0; // histogram of hue in the original hand image

CvRect prevFaceRect;  // location of face in previous frame
CvBox2D faceBox;      // current face-location estimate

CvRect prevHand1Rect;  // location of face in previous frame
CvBox2D hand1Box;      // current face-location estimate

CvRect prevHand2Rect;  // location of face in previous frame
CvBox2D hand2Box;      // current face-location estimate
```

```
int nFrames = 0;
// Declarations for internal functions
void updateHueImage(const IplImage * pImg);
////////////////////////////////
// createTracker()
//
int createTracker(const IplImage * pImg)
{
	// Allocate the main data structures ahead of time
	float * pRanges = rangesArr;
	pHSVImg  = cvCreateImage( cvGetSize(pImg), 8, 3 );
	pHueImg  = cvCreateImage( cvGetSize(pImg), 8, 1 );
	pMask    = cvCreateImage( cvGetSize(pImg), 8, 1 );
	pProbImg = cvCreateImage( cvGetSize(pImg), 8, 1 );

	pHist = cvCreateHist( 1, &nHistBins, CV_HIST_ARRAY, &pRanges, 1 );

	float * h1Ranges = rangesArr;
	h1HSVImg  = cvCreateImage( cvGetSize(pImg), 8, 3 );
	h1HueImg  = cvCreateImage( cvGetSize(pImg), 8, 1 );
	h1Mask    = cvCreateImage( cvGetSize(pImg), 8, 1 );
	h1ProbImg = cvCreateImage( cvGetSize(pImg), 8, 1 );

	h1Hist = cvCreateHist( 1, &nHistBins, CV_HIST_ARRAY, &h1Ranges, 1 );

	float * h2Ranges = rangesArr;
	h2HSVImg  = cvCreateImage( cvGetSize(pImg), 8, 3 );
	h2HueImg  = cvCreateImage( cvGetSize(pImg), 8, 1 );
	h2Mask    = cvCreateImage( cvGetSize(pImg), 8, 1 );
	h2ProbImg = cvCreateImage( cvGetSize(pImg), 8, 1 );

	h2Hist = cvCreateHist( 1, &nHistBins, CV_HIST_ARRAY, &h1Ranges, 1 );

	return 1;
}
////////////////////////////////
// releaseTracker()
//
void releaseTracker()
{
	// Release all tracker resources
	cvReleaseImage( &pHSVImg );
	cvReleaseImage( &pHueImg );
	cvReleaseImage( &pMask );
	cvReleaseImage( &pProbImg );
```

```
        cvReleaseHist( &pHist );
         cvReleaseImage( &h1HSVImg );
        cvReleaseImage( &h1HueImg );
        cvReleaseImage( &h1Mask );
        cvReleaseImage( &h1ProbImg );

        cvReleaseHist( &h1Hist );

        cvReleaseImage( &h2HSVImg );
        cvReleaseImage( &h2HueImg );
        cvReleaseImage( &h2Mask );
        cvReleaseImage( &h2ProbImg );

        cvReleaseHist( &h2Hist );

}
//////////////////////////////////
// startTracking()
//
void startTracking(IplImage * pImg, CvSeq* ROIs)
{
        float maxVal = 0.f;
        CvSeqReader reader;
        cvStartReadSeq(ROIs, &reader,0);
        CvRect pFaceRect, h1Rect, h2Rect;
        //memcpy( pFaceRect, reader->ptr, ROIs->elem_size );
        CV_READ_SEQ_ELEM(pFaceRect, reader);
        CV_READ_SEQ_ELEM(h1Rect, reader);
        CV_READ_SEQ_ELEM(h2Rect, reader);

        // Make sure internal data structures have been allocated
        if( !pHist||!h1Hist ) createTracker(pImg);

        // Create a new hue image
        updateHueImage(pImg);

        // Create a histogram representation for the face
  cvSetImageROI( pHueImg, pFaceRect );
  cvSetImageROI( pMask, pFaceRect );
  cvCalcHist( &pHueImg, pHist, 0, pMask );
  cvGetMinMaxHistValue( pHist, 0, &maxVal, 0, 0 );
  cvConvertScale( pHist->bins, pHist->bins, maxVal? 255.0/maxVal : 0, 0 );
  cvResetImageROI( pHueImg );
  cvResetImageROI( pMask );
```

```
        // Store the previous face location
        prevFaceRect = pFaceRect;
  cvSetImageROI( h1HueImg, h1Rect );
  cvSetImageROI( h1Mask, h1Rect );
  cvCalcHist( &h1HueImg, h1Hist, 0, h1Mask );
  cvGetMinMaxHistValue( h1Hist, 0, &maxVal, 0, 0 );
  cvConvertScale( h1Hist->bins, h1Hist->bins, maxVal? 255.0/maxVal : 0, 0 );
  cvResetImageROI( h1HueImg );
  cvResetImageROI( h1Mask );

        // Store the previous face location
        prevHand1Rect = h1Rect;

        cvSetImageROI( h2HueImg, h2Rect );
  cvSetImageROI( h2Mask, h2Rect );
  cvCalcHist( &h2HueImg, h2Hist, 0, h2Mask );
  cvGetMinMaxHistValue( h2Hist, 0, &maxVal, 0, 0 );
  cvConvertScale( h2Hist->bins, h2Hist->bins, maxVal? 255.0/maxVal : 0, 0 );
  cvResetImageROI( h2HueImg );
  cvResetImageROI( h2Mask );

        // Store the previous face location
        prevHand2Rect = h2Rect;

}
//////////////////////////////////
// track()
//
CvRect trackFace(IplImage * pImg )
{
        CvConnectedComp componentsf;


        // Create a new hue image
        updateHueImage(pImg);

        // Create a probability image based on the face histogram
        cvCalcBackProject( &pHueImg, pProbImg, pHist );
  cvAnd( pProbImg, pMask, pProbImg, 0 );


        // Use CamShift to find the center of the new face probability
  cvCamShift( pProbImg, prevFaceRect,
          cvTermCriteria( CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 10, 1 ),
          &componentsf, &faceBox );


        // Update face location and angle
        prevFaceRect = componentsf.rect;

        faceBox.angle = -faceBox.angle;
        return prevFaceRect;
}

CvRect trackHand1(IplImage * pImg)
{

        CvConnectedComp componentsh;

        // Create a new hue image
        updateHueImage(pImg);

        cvCalcBackProject( &h1HueImg, h1ProbImg, h1Hist );
  cvAnd( h1ProbImg, h1Mask, h1ProbImg, 0 );

        // Use CamShift to find the center of the new hand probability
  cvCamShift( h1ProbImg, prevHand1Rect,
          cvTermCriteria( CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 10, 1 ),
          &componentsh, &hand1Box );

        // Update location and angle
  prevHand1Rect = componentsh.rect;

        hand1Box.angle = hand1Box.angle;

        return prevHand1Rect;
}
CvRect trackHand2(IplImage * pImg)
{
        CvConnectedComp componentsh2;

        // Create a new hue image
        updateHueImage(pImg);

        cvCalcBackProject( &h2HueImg, h2ProbImg, h2Hist );
  cvAnd( h2ProbImg, h2Mask, h2ProbImg, 0 );

        // Use CamShift to find the center of the new hand probability
  cvCamShift( h2ProbImg, prevHand2Rect,
            cvTermCriteria( CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 10, 1 ),
            &componentsh2, &hand2Box );

          // Update location and angle
   prevHand2Rect = componentsh2.rect;
          hand2Box.angle = -hand2Box.angle;

          return prevHand2Rect;
}
//////////////////////////////////
// updateHueImage()
//
void updateHueImage(const IplImage * pImg)
{
          // Convert to HSV color model
          cvCvtColor( pImg, pHSVImg, CV_BGR2HSV );

          cvCvtColor( pImg, h1HSVImg, CV_BGR2HSV );

          cvCvtColor( pImg, h2HSVImg, CV_BGR2HSV );


          // Mask out-of-range values
          cvInRangeS( pHSVImg, cvScalar(0, smin, MIN(vmin,vmax), 0),
                    cvScalar(180, 256, MAX(vmin,vmax) ,0), pMask );

          // Mask out-of-range values
          cvInRangeS( h1HSVImg, cvScalar(0, smin, MIN(vmin,vmax), 0),
                    cvScalar(180, 256, MAX(vmin,vmax) ,0), h1Mask );

          // Mask out-of-range values
          cvInRangeS( h2HSVImg, cvScalar(0, smin, MIN(vmin,vmax), 0),
                    cvScalar(180, 256, MAX(vmin,vmax) ,0), h2Mask );

          // Extract the hue channel
          cvSplit( pHSVImg, pHueImg, 0, 0, 0 );

          // Extract the hue channel
          cvSplit( h1HSVImg, h1HueImg, 0, 0, 0 );

          // Extract the hue channel
          cvSplit( h2HSVImg, h2HueImg, 0, 0, 0 );
}
//////////////////////////////////
// setVmin()
```

```
//
void setVmin(int _vmin)
{ vmin = _vmin; }
////////////////////////////////
// setSmin()
//
void setSmin(int _smin)
{ smin = _smin; }
```

6.2 Skin Segmentation Analysis

Skin.cpp

```
#include "cv.h"
#include "highgui.h"
#include <stdio.h>
#include <conio.h>
#include "cxcore.h"
void GetSkinMask(IplImage * src_RGB, IplImage * mask_BW, int erosions=1, int dilations=7)
{
        CvSize size;

        CvSize sz = cvSize( src_RGB->width & -2, src_RGB->height & -2);
        //get the size of input_image (src_RGB)

        IplImage* pyr = cvCreateImage( cvSize(sz.width/2, sz.height/2), 8,3 ); //create 2 temp-
images

        IplImage* src = cvCreateImage(cvGetSize(src_RGB), IPL_DEPTH_8U ,3);
        cvCopyImage(src_RGB, src);

        IplImage* tmpYCR = cvCreateImage(cvGetSize(src), IPL_DEPTH_8U , 3);

        cvPyrDown( src, pyr, 7 );
        //remove noise from input
        cvPyrUp( pyr, src, 7 );

        cvCvtColor(src ,tmpYCR , CV_RGB2YCrCb);
        uchar Y;
        uchar Cr;
        uchar Cb;
        CvPixelPosition8u pos_src;
        CvPixelPosition8u pos_dst;
        int x =0;
        int y =0;

        CV_INIT_PIXEL_POS(pos_src,(unsigned char *) tmpYCR->imageData,        tmpYCR-
>widthStep,cvGetSize(tmpYCR),x,y,tmpYCR->origin);

        CV_INIT_PIXEL_POS(pos_dst,(unsigned char *) mask_BW->imageData,     mask_BW-
>widthStep,    cvGetSize(mask_BW), x,y,mask_BW->origin);

        uchar * ptr_src;
        uchar * ptr_dst;
        for( y=0;y<src-> height; y++)
        {
                for ( x=0; x<src->width; x++)
                {
                        ptr_src = CV_MOVE_TO(pos_src,x,y,3);
                        ptr_dst = CV_MOVE_TO(pos_dst,x,y,3);

                        Y = ptr_src[0];
                        Cb= ptr_src[1];
                        Cr= ptr_src[2];

                        If
( Cr > 138 && Cr < 178 &&     Cb + 0.6 * Cr >200 && Cb + 0.6 * Cr <215)
                        {
                                ptr_dst[0] = 255;
                                ptr_dst[1] = 255;
                                ptr_dst[2] = 255;
                        }
                        else
                        {
                                ptr_dst[0] = 0;
                                ptr_dst[1] = 0;
                                ptr_dst[2] = 0;
                        }
                }
        }

        if(erosions>0) cvErode(mask_BW,mask_BW,0,erosions);
                if (dilations>0) cvDilate(mask_BW,mask_BW,0,dilations);
                        cvReleaseImage(&pyr);

        cvReleaseImage(&tmpYCR);
        cvReleaseImage(&src);
}
```

```
int apple(int argc, char** argv)
{
		IplImage *img, *imgSkin;
   img = cvLoadImage("mikki2.jpg"); /* loads the image from the command line */
		imgSkin = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 3);
		GetSkinMask(img, imgSkin, 1, 7);
		cvSaveImage("result2.jpg", imgSkin);
		return 0;
}
```

6.3 Segmentation Analysis

Segmentation.cpp

```
#include "cv.h"
#include "highgui.h"
#include <ctype.h>
#include "cxcore.h"
#include "cvaux.h"
#include "CamShift.h"
#include <stdio.h>
#include "Houghtransform.cpp"
#include "svm-predict-new.c"
//#include "Classifier.cpp"

#define maxBins 512
#define PI 3.1415

int ConnectedComponent[640*480][2];
int i =0, j =0;

int checkForSkinColor(int x, int y, IplImage *img);
int connectedComp(int x, int y, IplImage *img);

extern void GetSkinMask(IplImage * src_RGB, IplImage * mask_BW, int erosions, int dilations);
extern void on_trackbar( IplImage* original, IplImage* originalThr,IplImage* displayedImage,
int param1 = 0,int param2 = 1000  );
extern CvSeq* CCA(IplImage *I1, IplImage* I2, CvMemStorage *mem);
//extern double svmpredict(char *s);
void getROIFrame(IplImage *vFrame, IplImage* frame, CvRect R);
void updateFrame(IplImage* vFrame, IplImage *frame, CvBox2D box);
int minimum(int a, int b);
int maximum(int a, int b);
```

```
void Print(IplImage *I, char *string);

void main(int argc, char ** argv)
{
        CvCapture* capture = cvCreateCameraCapture( 0 );
        assert(capture);

   /* print a welcome message, and the OpenCV version */

        //   CV_VERSION,
        //    CV_MAJOR_VERSION, CV_MINOR_VERSION, CV_SUBMINOR_VERSION);

   /* Capture 1 video frame for initialization */
        IplImage* videoFrame = NULL;
        IplImage* BGFrame = NULL;
        //BGFrame = cvQueryFrame(capture);
        videoFrame = cvQueryFrame(capture);

   if(!videoFrame)
        {
                printf("Bad frame \n");
                exit(0);
        }
        // Create windows


        // Select parameters for Gaussian model.
        /*CvGaussBGStatModelParams* params = new CvGaussBGStatModelParams;

        params->win_size=2;
   params->n_gauss=3;
        params->bg_threshold=0.9;
        params->std_threshold=.5;
        params->minArea=15;
        params->weight_init=0.05;
        params->variance_init=30;

   CvBGStatModel* bgModel = cvCreateGaussianBGModel(videoFrame ,params);
        */

        IplImage  *skinFrame,  *skinFrameGray,  *ccImage,  *  originalThr,  *skinFrameRGB,
*sobelFrame, *sobelFrameConverted ;
        IplImage* faceFrame, * hand1Frame,* hand2Frame, *videoFramecopy;
        skinFrame = cvCreateImage(cvGetSize(videoFrame), IPL_DEPTH_8U, 3);
        videoFramecopy = cvCreateImage(cvGetSize(videoFrame), IPL_DEPTH_8U, 3);
        sobelFrame = cvCreateImage(cvGetSize(videoFrame), IPL_DEPTH_16S, 1);
        sobelFrameConverted = cvCreateImage(cvGetSize(videoFrame), IPL_DEPTH_8U, 1);
        skinFrameRGB = cvCreateImage(cvGetSize(skinFrame), IPL_DEPTH_8U, 3);
        skinFrameGray = cvCreateImage(cvGetSize(skinFrame), IPL_DEPTH_8U, 1);
        ccImage = cvCreateImage(cvGetSize(skinFrame), IPL_DEPTH_8U, 3);
        originalThr = cvCreateImage(cvGetSize(skinFrame), IPL_DEPTH_8U, 1);
        faceFrame = cvCreateImage(cvGetSize(videoFrame), IPL_DEPTH_8U, 3);
        hand1Frame = cvCreateImage(cvGetSize(videoFrame), IPL_DEPTH_8U, 3);
        hand2Frame = cvCreateImage(cvGetSize(videoFrame), IPL_DEPTH_8U, 3);
        CvMemStorage *mem, *mem2;
        CvSeq* RegionOfInterests = NULL;
        CvSeq* RegionOfInterests2 = NULL;
        mem = cvCreateMemStorage(0);
        mem2 = cvCreateMemStorage(0);
int key=-1;
        cvNamedWindow("BG", 1);
        cvNamedWindow("VideoFrame", 1);
while(1)
        {
                // Grab a frame
                videoFrame = cvQueryFrame(capture);
                cvShowImage("BG", videoFrame);
                if( !videoFrame )
                        break;
                if( 'b'==cvWaitKey(1) )
                {
                        videoFrame = cvQueryFrame(capture);
                        cvSaveImage("videoFrame.jpg", videoFrame);
                        cvDestroyWindow("BG");
                        cvReleaseCapture(&capture);
                        break;
                }
        }
        capture = cvCreateCameraCapture( 0 );
        assert(capture);
        while(1)
        {
                BGFrame = cvQueryFrame(capture);//Update model(Background subtraction)
                if( 'c'==cvWaitKey(1) )
                        break;
        }

        //cvCvtColor(backg,backg,CV_RGB2YCrCb);

cvNamedWindow("FG", 1);
```

```
        cvNamedWindow("Image",1);
        cvNamedWindow("SkinFrame", 1);
        while(1)
        {
                BGFrame = cvQueryFrame(capture);//Update model(Background subtraction)
                cvSaveImage("BGFrame.jpg",BGFrame);
                //cvCvtColor(BGFrame,BGFrame,CV_RGB2YCrCb);

                //cvUpdateBGStatModel(videoFrame,bgModel);
                IplImage *img1 = cvLoadImage("BGFrame.jpg",FALSE);
                IplImage *img2 =  cvLoadImage("videoFrame.jpg",FALSE);

                //code to extract the byte array of the pixels from the image

                //initialize byte array as zero
                BYTE *Pixel1=0;
                BYTE *Pixel2=0;
                //extract pixels using the OpenCV function cvGetRawData
                cvGetRawData(img1,&Pixel1,0,0);
                cvGetRawData(img2,&Pixel2,0,0);

                //get height and width using OpenCV functions
                const int &rows = img1->height;
                const int &cols = img2->width;

                //register int to increase the speed
                register  int r,ri,c;

                //to find diffrence of 2 images by pixel to pixel comparision
                for(r = 0, ri = 0; r < rows; r++, ri += cols)
                {
                        for(c = 0; c < cols; c++)
                        {
                                        //get the difference in pixels
                                Pixel1[ri + c] = Pixel1[ri + c] - Pixel2[ri + c];

                                //set threshold value as 100 for comparision, it can be changed to
values between 50 and 200, for getting binary image
                                if(Pixel1[ri + c] < 100)
                                {
                                        Pixel1[ri + c]=0;
          }
                                else
            Pixel1[ri + c]=255;
```

```
				}//for c

			}//for r, ri*/

			//create a named window to display the image  with only foreground
			cvShowImage("Image",img1);
			cvSaveImage("sub.jpg", img1);
			//Detect The Skin
			GetSkinMask(BGFrame, skinFrame, 1, 7);
			cvShowImage("SkinFrame", skinFrame);
			cvCvtColor(skinFrame, skinFrameRGB, CV_YCrCb2BGR);
			cvCvtColor(skinFrameRGB, skinFrameGray, CV_BGR2GRAY);
			cvThreshold(skinFrameGray, skinFrameGray, 155, 255, CV_THRESH_BINARY);
			BYTE *Pixel3=0;
			BYTE *Pixel4=0;
			//extract pixels using the OpenCV function cvGetRawData
			cvGetRawData(img1,&Pixel3,0,0);
			cvGetRawData(skinFrameGray,&Pixel4,0,0);

			//get height and width using OpenCV functions
			const int &rows2 = img1->height;
			const int &cols2 = skinFrameGray->width;

			//register int to increase the speed
			register  int r2,ri2,c2;

			//to find diffrence of 2 images by pixel to pixel comparision
			for(r2 = 0, ri2 = 0; r2 < rows2; r2++, ri2 += cols2)
			{
				for(c2 = 0; c2 < cols2; c2++)
				{
		//get the difference in pixels

		//set threshold value as 100 for comparision, it can be changed to values between 50
	and 200, for getting binary image
		if(Pixel3[ri2 + c2]!=255||Pixel4[ri2+c2]!=255)
		{
					Pixel4[ri2 + c2]=0;
		}
				 else
					Pixel4[ri2 + c2]=255;

				}//for c

			}//for r, ri*/
```

```
                    if( (char)27==cvWaitKey(1) ) exit(0);

                    //Find the first Connected Component
                    RegionOfInterests = CCA(skinFrameGray, ccImage, mem);

            // Display results
//          cvShowImage("BG", bgModel->background);
                    cvShowImage("FG", skinFrameGray);

                    cvShowImage("VideoFrame", BGFrame);

                    if(&RegionOfInterests)
                            break;
    }
            cvReleaseImage( &skinFrame );
            cvReleaseImage( &ccImage );
            cvReleaseImage( &skinFrameGray );
            cvReleaseImage( &skinFrameRGB );
            cvDestroyWindow("BG");
    cvDestroyWindow("FG");
            cvDestroyWindow("ConnectedComponent");
            cvDestroyWindow("SkinFrame");

            cvNamedWindow("Tracking", 1);
            videoFrame = cvQueryFrame(capture);
            //Initialize Tracking
            startTracking(videoFrame, RegionOfInterests);
            //FILE *fp = fopen("tired.txt","w");
            FILE *fptest = fopen("test.txt","w");
            FILE *fpout = fopen("output.txt","w");
            //Classifier *classifier = new Classifier();
            int count =0;
            //cvNamedWindow("emotion");
            while(1)
            {
                    CvRect faceBox, hand1Box, hand2Box, shoulderBox;
                    double output;
                    //cvShowImage("Tracking2", faceFrame);
                    //cvShowImage("Tracking4", hand1Frame);
                    // Grab a frame
                    videoFrame = cvQueryFrame(capture);
                    videoFramecopy = cvCloneImage(videoFrame);
                    faceBox = trackFace(videoFrame);
                    //cvShowImage("Tracking3", faceFrame);
```

```
                hand1Box = trackHand1(videoFrame);
                //cvShowImage("Tracking5", hand1Frame);
                //hand2Box = trackHand2(videoFrame);
                // outline face ellipse
                cvRectangle(videoFrame,cvPoint(faceBox.x,faceBox.y),
            cvPoint(faceBox.x+faceBox.width,
            faceBox.y+ faceBox.height),
            cvScalar(0,255,0),0);

                cvRectangle(videoFrame,cvPoint(hand1Box.x,hand1Box.y),
            cvPoint(hand1Box.x+hand1Box.width,
            hand1Box.y+ hand1Box.height),
            cvScalar(0,255,0),0);
                /*cvRectangle(videoFrame,cvPoint(hand2Box.x,hand2Box.y),
            cvPoint(hand2Box.x+hand2Box.width,
            hand2Box.y+ hand2Box.height),
             cvScalar(0,255,0),0);
                /* Shoulder Detection */
                shoulderBox = cvRect(faceBox.x - faceBox.width, faceBox.y+(0.5*faceBox.height),
3 * faceBox.width, 1.5* faceBox.height);
                cvRectangle(videoFrame,cvPoint(shoulderBox.x,shoulderBox.y),
            cvPoint(shoulderBox.x+shoulderBox.width,
            shoulderBox.y+ shoulderBox.height),
            cvScalar(0,255,0),0);
                IplImage    *shoulderFrame    =    cvCreateImage(cvGetSize(videoFrame),
IPL_DEPTH_8U, 1);

                /** Extracting ShoulderFrame ****/

                CvPixelPosition8u pos_src;
                CvPixelPosition8u pos_dst;

                int x =0;
                int y =0;

                CV_INIT_PIXEL_POS(pos_src,(unsigned  char  *)  videoFramecopy->imageData,
        videoFramecopy->widthStep,cvGetSize(videoFramecopy),x,y,videoFramecopy->origin);

                CV_INIT_PIXEL_POS(pos_dst,(unsigned  char  *)  shoulderFrame->imageData,
        shoulderFrame->widthStep,  cvGetSize(shoulderFrame),  x,y,shoulderFrame->origin);

                uchar * ptr_src;
                uchar * ptr_dst;

                for( y=0;y<videoFramecopy-> height; y++)
```

```
				{
					for ( x=0; x<videoFramecopy->width; x++)
					{
						ptr_src = CV_MOVE_TO(pos_src,x,y,3);
						ptr_dst = CV_MOVE_TO(pos_dst,x,y,1);


		if(x>shoulderBox.x&&x<=shoulderBox.x+shoulderBox.width&&y>shoulderBox.y&&y<=s
houlderBox.y+shoulderBox.height)
						{
							ptr_dst[0] = ptr_src[0];
						//	ptr_dst[1] = ptr_src[1];
						//	ptr_dst[2] = ptr_src[2];
						}
						else
						{
							ptr_dst[0] = 0;
						//	ptr_dst[1] = 0;
						//	ptr_dst[2] = 0;
						}
					}
				}
				cvSobel(shoulderFrame, sobelFrame, 0, 1);
				cvConvertScaleAbs(sobelFrame, sobelFrameConverted);
				cvThreshold(sobelFrameConverted,		sobelFrameConverted,100,		255,
CV_THRESH_BINARY);

				Hough* hough = new Hough(sobelFrameConverted);
				IplImage	*parabola	=	hough->HoughTransform(sobelFrameConverted,
shoulderBox, videoFrame);
				cvShowImage( "Tracking", videoFrame );
		//		cvShowImage( "Shoulder", sobelFrameConverted);
				cvSaveImage("resultShoulder1.jpg", sobelFrameConverted);
				float	centroid_diff_x_h1	=	abs((faceBox.x+faceBox.width/2)	-
(hand1Box.x+hand1Box.width/2));
				float	centroid_diff_y_h1	=	abs((faceBox.y+faceBox.height/2)	-
(hand1Box.y+hand1Box.height/2));
				float	centroid_diff_x_h2	=	abs((faceBox.x+faceBox.width/2)	-
0);//(hand2Box.x+hand2Box.width/2));
				float	centroid_diff_y_h2	=	abs((faceBox.y+faceBox.height/2)	-
0);//(hand2Box.y+hand2Box.height/2));
				//cvSaveImage("resultShoulder.jpg", parabola);
		/*		if(count<50&&'a'==cvWaitKey(1))
				{
```

```
				fprintf(fp,   "%d\t1:%f\t2:%f\t3:%f\t4:%f\t5:%f\t6:%f\t7:%f\n",5,   hough->getAmax(),  hough->getBmax(),  hough->getCmax(),  centroid_diff_x_h1  ,  centroid_diff_y_h1, centroid_diff_x_h2 , centroid_diff_y_h2);
				count++;
			}
			else if(count>=50)
			{
				fclose(fp);
				cvReleaseCapture(&capture);
				break;
			}
		*/
			char str[200];
			int n;
			n = sprintf(str, "0\t1:%f\t2:%f\t3:%f\t4:%f\t5:%f\t6:%f\t7:%f", hough->getAmax(), hough->getBmax(), hough->getCmax(), centroid_diff_x_h1 , centroid_diff_y_h1, centroid_diff_x_h2 , centroid_diff_y_h2);
			output = svmpredict(str);
			//output = classifier->classify(videoFramecopy,hough->getAmax(), hough->getBmax(), hough->getCmax(), centroid_diff_x, centroid_diff_y);
			count++;
			printf("%d\n", count);
			//fprintf(fptest, "0\t1:%f\t2:%f\t3:%f\t4:%f\t5:%f\t6:%f\t7:%f\n", hough->getAmax(), hough->getBmax(), hough->getCmax(), centroid_diff_x_h1 , centroid_diff_y_h1, centroid_diff_x_h2 , centroid_diff_y_h2);

			if(output==2)
			{
				IplImage *shock = cvLoadImage("shock-4.jpg");
				cvShowImage("emotion", shock);
				//Print(videoFrame, "Positive Surprised");
				printf("shrug\n");
				fprintf(fptest, "%s\n","shrug");
			}

			else if(output==1)
			{
				IplImage *neutral = cvLoadImage("Neutral.jpg");
				cvShowImage("emotion", neutral);
				printf("neutral\n");
				fprintf(fptest, "%s\n","neutral");
			}
			else if(output==3)
			{
				//IplImage *neutral = cvLoadImage("Neutral.jpg");
```

```
                        //cvShowImage("emotion", neutral);
                        printf("positive\n");
                        fprintf(fptest, "%s\n","surprise");
                }
                else if(output==4)
                {
                        IplImage *emotion = cvLoadImage("tensed.jpg");
                        cvShowImage("emotion", emotion);
                        printf("tensed\n");
                        fprintf(fptest, "%s\n","tense");
                }
                else if(output==5)
                {
                        IplImage *emotion = cvLoadImage("tired.jpg");
                        cvShowImage("emotion", emotion);
                        printf("tired\n");
                        fprintf(fptest, " %s\n","tired");
                }

                if( (char)27==cvWaitKey(1) )
                {
                        cvReleaseCapture(&capture);
                        cvReleaseImage( &videoFrame );
                        fclose(fptest);
                        //cvReleaseBGStatModel( &bgModel );
                        break;
                }
        }
}

void Print(IplImage* I, char* string)
{
        cvPutText(I, string, cvPoint(250, 80), &cvFont(10), cvScalar(0, 0, 255));

}
```

6.4 Hough Transform

hough.cpp

```
#include "cv.h"

#include <stdio.h>
```

```
#include "highgui.h"

class Hough

{

private:

      double aMin, bMin, cMin, aMax, bMax, cMax;

      int n1, n2, n3;

      double *aArray ;

      double *bArray ;

      double *cArray ;

      int ***HoughSpace;

      int maxR ,maxL, maxM, maxV;

      double aBin ;

      double bBin;

public:

      Hough(IplImage * I)

{           aMin = 0.0010;

            aMax = 0.0500;

            bMin = 0;

            bMax = -I->width/100;

            aBin = 0.001;

            bBin = 0.1;

            n1= (int)((aMax - aMin)/ aBin);
```

```
n2 = (int)((bMin - bMax)/bBin);

n3 = I->height;

aArray = new double[n1];

bArray = new double[n2];

cArray = new double[n3];

HoughSpace = new int**[n1];



for(int l=0; l<n1; l++)

{

      HoughSpace[l] = new int*[n2];

for(  int m=0; m<n2;m++)

      {

                  HoughSpace[l][m] = new int[n3];

            for(int v=0; v<n3; v++)

            {

                  HoughSpace[l][m][v] = 0;



            }

      }

}

maxR=0;

maxL=0;
```

```
            maxM=0;

            maxV=0;

      }



      IplImage* HoughTransform(IplImage* I, CvRect Rect, IplImage *result)

      {

      //    IplImage *result = cvCreateImage(cvGetSize(I), IPL_DEPTH_8U, 1);



            uchar R;

            CvPixelPosition8u pos_src;

            CvPixelPosition8u pos_dst;

            int x =0;

            int y =0;

            int i=0;

            CV_INIT_PIXEL_POS(pos_src,(unsigned   char   *)I->imageData,   I-
>widthStep, cvGetSize(I), x, y, I->origin);

            CV_INIT_PIXEL_POS(pos_dst,(unsigned   char   *)result->imageData,
result->widthStep, cvGetSize(result),     x, y, result->origin);

            double a = 0, b =0, xcenter;

            int c = 0, value = 0;

            uchar * ptr_src;

            uchar * ptr_dst;
            for( y=0;y<I->height; y++)

            {

                  for ( x=0; x<I->width; x++)

                  {

                        ptr_src = CV_MOVE_TO(pos_src,x,y,1);

                        ptr_dst = CV_MOVE_TO(pos_dst,x,y,1);

                        R = ptr_src[0];

                        if(R==255&&  y>Rect.y  +  5  &&  y<  Rect.y+Rect.height  -
5)

                        {

                              a = aMin;

                              i =0;

                              while(a<=aMax&&i<n1)

                              {

                                    aArray[i] = a;

                                    a = a + aBin;

                                    i++;

                                    }

                              //n1 = i;

                              b = bMin;

                              i=0;

                              while(b>=bMax&&i<n2)
                              {

                                    bArray[i] = b;

                                    b = b - bBin;

                                    i++;

                              }

                              i=0;

                              while(c<I->height)

                              {

                                    cArray[i] = c;

                                    c= c+1;

                                    i++;

                              }

                              // = i;

                              for(int j=0; j< n1; j++)

                              {

                                    for(int k=0; k<n2; k++)

                                    {

                                          value=(int)(y   -   aArray[j]*x*x   -
bArray[k]*x);

                                          xcenter = -bArray[k]/(2*aArray[j]);

                                          if(value>0&&value<I-
>height&&(xcenter>Rect.x)&&(xcenter<Rect.x+Rect.width))

                                          {
```

```
                                                    HoughSpace[j][k][value]      =
HoughSpace[j][k][value]+ R;
                                            }
                                    }
                            }
                    }
            }
    }
    for(int l=0; l<n1; l++)
    {
    for(int m=0; m<n2; m++)
            {
            for(int v=0; v<n3; v++)
                    {
                            if(maxR<HoughSpace[l][m][v])
                            {
                                    maxR = HoughSpace[l][m][v];
                                    maxL= l;
                                    maxM = m;
                                    maxV = v;
                            }
                    }
```

```
                        }

                }

                for(int x=0; x<result->width; x++)

                {

                        y     =     (int)(aArray[maxL]*x*x     +     bArray[maxM]*x     +
cArray[maxV]);


        if(y>Rect.y&&y<Rect.y+Rect.height&&x>Rect.x&&x<Rect.x+Rect.width)

                        cvSet2D(result, y, x, cvScalar(255));

                }

                return result;

        }

        double getAmax()

        {

                return aArray[maxL];

        }

        double getBmax()

        {

                return bArray[maxM];

        }

double getCmax()

        {

                return cArray[maxV];
```

```
}


int main4(int argc, char *argv[])

{

      IplImage *imgThr; /*IplImage is an image in OpenCV*/

      CvCapture* capture = cvCreateCameraCapture( 0 );

      assert(capture);


    /* print a welcome message, and the OpenCV version */


      //   CV_VERSION,

      //    CV_MAJOR_VERSION, CV_MINOR_VERSION, CV_SUBMINOR_VERSION);


    /* Capture 1 video frame for initialization */

      IplImage* img = NULL;

      img = cvQueryFrame(capture);


      imgThr = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 3);

    cvThreshold(img, imgThr, 155, 255, CV_THRESH_BINARY);

      //IplImage *parabola = HoughTransform(imgThr,);

//    cvSaveImage("resultShoulder.jpg", parabola);

      return 0;
```

```
        }


};
```

6.5 Classifier

Classifier.cpp

```
#include "cv.h"

#include <stdio.h>

#include "highgui.h"

extern double svmpredict(char *s);

class Classifier {

private:

public:

        void train(FILE *fp,double a, double b, double c, double rcx, double rcy)

        {

                fprintf(fp, "%d\t%lf\t%lf\t%lf\t%lf\t%lf\n", 1, a, b, c, rcx, rcy);

        }



        double classify(IplImage *I, double a, double b, double c, double cdx, double cdy)

        {
                char s[200];

                int n;

                n = sprintf(s, "0\t1:%f\t2:%f\t3:%f\t4:%f\t5:%f", a, b, c, cdx, cdy);

                double d = svmpredict(s);

                return d;

        }

};

/*int main6(int argc, char** argv)

{

        char *argv1[3] = {"", "train.txt", "model.txt"};

        main0(3,argv1);

        return 0;

}*/
```

6.6 Connected component Analysis

Cca.cpp

```
#include "cv.h"

#include "highgui.h"

#include <stdio.h>

#include <conio.h>
```

```
// Main blob library include

//#include "BlobResult.h"



char wndname[] = "Blob Extraction";

char tbarname1[] = "Threshold";

char tbarname2[] = "Blob Size";



// The output and temporary images

//int param1 = 0,param2 = 1000;

// threshold trackbar callback

/*void  on_trackbar(  IplImage*  original,  IplImage*  originalThr,IplImage*
displayedImage, int param1 = 255,int param2 = 1000  )

{

      if(!originalThr)

      {

            originalThr = cvCreateImage(cvGetSize(original), IPL_DEPTH_8U,1);

      }

      if(!displayedImage)

      {

            displayedImage        =        cvCreateImage(cvGetSize(original),
IPL_DEPTH_8U,3);

      }
```

```
		// threshold input image
	//cvThreshold( original, originalThr, param1, 255, CV_THRESH_BINARY );


	// get blobs and filter them using its area
	CBlobResult blobs;
	int i;
	CBlob *currentBlob;


	// find blobs in image
	blobs = CBlobResult( original, NULL, 255 );
	blobs.Filter( blobs, B_EXCLUDE, CBlobGetArea(), B_LESS, param2 );


	// display filtered blobs
	//cvMerge( originalThr, originalThr, originalThr, NULL, displayedImage 
);
	cvMerge( original, original, original, NULL, displayedImage );


	for (i = 0; i < blobs.GetNumBlobs(); i++ )
	{
		currentBlob = blobs.GetBlob(i);
		currentBlob->FillBlob( displayedImage, CV_RGB(255,0,0));
	}
```

```
   //cvSaveImage("result5.jpg", displayedImage);

}

int main2(int argc, char **argv)

{

      IplImage* originalThr = 0;

      IplImage* original = cvLoadImage("mikki2.jpg",0);

      IplImage* originalSkin = 0;

      IplImage* displayedImage = 0;

      on_trackbar(original, originalThr, displayedImage);

      return 0;

}

*/

 //int main(int argc, char *argv[])

CvSeq* CCA(IplImage* img, IplImage* cc_color,   CvMemStorage *mem)

{

   //cvNamedWindow("Thr",1);

      IplImage *imgThr; /*IplImage is an image in OpenCV*/

         CvSeq *contours, *ptr;

   // img = cvLoadImage("mikki2.jpg", 0); /* loads the image from the command
line */

      cc_color = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 3);
      imgThr = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);

    cvThreshold(img, imgThr,  155, 255, CV_THRESH_BINARY);

    cvFindContours(imgThr, mem, &contours, sizeof(CvContour), CV_RETR_CCOMP,
CV_CHAIN_APPROX_SIMPLE, cvPoint(0,0));

      //CvSeq* result = cvApproxPoly(contours, sizeof(CvContour), mem,
CV_POLY_APPROX_DP, 2, 0);

      double max =0, max2 = 0, max3 = 0;

      CvSeq* c = contours, *c2 = contours, *c3 = contours;

      CvRect boundRect, boundRect2, boundRect3;

      for (ptr = contours; ptr != NULL; ptr = ptr->h_next)

      {

        //CvScalar color = CV_RGB( rand()&255, rand()&255, rand()&255 );

            //cvDrawContours(cc_color, ptr, color, CV_RGB(0,0,0), -1,
CV_FILLED, 8, cvPoint(0,0));

            boundRect = cvBoundingRect(ptr,0);

            if(max<boundRect.height*boundRect.width)

            {

                  max = boundRect.height*boundRect.width;

                  c =ptr;

            }

/*          cvRectangle(cc_color,cvPoint(boundRect.x,boundRect.y),

                    cvPoint(boundRect.x+boundRect.width,

                    boundRect.y+boundRect.height),
                    cvScalar(255,0,0),0);
*/
    }
      boundRect = cvBoundingRect(c,0);
      for (ptr = contours; ptr != NULL && ptr!=c; ptr = ptr->h_next)
      {
        //CvScalar color = CV_RGB( rand()&255, rand()&255, rand()&255 );
            //cvDrawContours(cc_color,   ptr,   color,   CV_RGB(0,0,0),   -1,
CV_FILLED, 8, cvPoint(0,0));
            boundRect2 = cvBoundingRect(ptr,0);
                  if(max2<boundRect2.height*boundRect2.width)
                  {
                        max2 = boundRect2.height*boundRect2.width;
                        c2 = ptr;
                  }
    }
      boundRect2 = cvBoundingRect(c2,0);
      for  (ptr  =  contours;  ptr  !=  NULL  &&  ptr!=c  &&ptr!=c2;  ptr  =  ptr-
>h_next)
      {
        //CvScalar color = CV_RGB( rand()&255, rand()&255, rand()&255 );
            //cvDrawContours(cc_color,   ptr,   color,   CV_RGB(0,0,0),   -1,
CV_FILLED, 8, cvPoint(0,0));
```

```
            boundRect3 = cvBoundingRect(ptr,0);

                  if(max3<boundRect2.height*boundRect2.width)

                  {

                        max3 = boundRect2.height*boundRect2.width;

                        c3 = ptr;

                  }

    }

      boundRect3 = cvBoundingRect(c3,0);

      cvRectangle(cc_color,cvPoint(boundRect.x,boundRect.y),

                    cvPoint(boundRect.x+boundRect.width,

                    boundRect.y+boundRect.height),

                    cvScalar(0,255,0),0);

      cvRectangle(cc_color,cvPoint(boundRect2.x,boundRect2.y),

                    cvPoint(boundRect2.x+boundRect2.width,

                    boundRect2.y+boundRect2.height),

                    cvScalar(255,0,0),0);

      cvRectangle(cc_color,cvPoint(boundRect3.x,boundRect3.y),

                    cvPoint(boundRect3.x+boundRect3.width,

                    boundRect3.y+boundRect3.height),

                    cvScalar(0,0,255),0);

      CvMemStorage * storage = cvCreateMemStorage(0);

      CvSeqWriter writer;
```

```
        cvStartWriteSeq( 0, sizeof(CvSeq ), sizeof(CvRect), storage, &writer );

        CV_WRITE_SEQ_ELEM( boundRect, writer );

        CV_WRITE_SEQ_ELEM( boundRect2, writer );

        CV_WRITE_SEQ_ELEM( boundRect3, writer );

        CvSeq* ROIs = cvEndWriteSeq( &writer );

        cvSaveImage("result.jpg", cc_color);

      //cvReleaseImage(&img);

      //cvReleaseImage(&cc_color);

      return ROIs;

}
```

6.7 SVM

Svmpredictnew

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include "svm.h"
#include <float.h>

struct svm_node *x;
int max_nr_attr = 64;

struct svm_model* model;
int predict_probability=0;

double lower=-1.0,upper=1.0,y_lower,y_upper;
int y_scaling = 0;
double *feature_max;
double *feature_min;
double y_max = -DBL_MAX;
double y_min = DBL_MAX;
int max_index;
long int num_nonzeros = 0;
long int new_num_nonzeros = 0;

static char *line = NULL;
static int max_line_len;

#define max(x,y) (((x)>(y))?(x):(y))
#define min(x,y) (((x)<(y))?(x):(y))

static char* readline(FILE *input)
{
	int len;

	if(fgets(line,max_line_len,input) == NULL)
		return NULL;

	while(strrchr(line,'\n') == NULL)
	{
		max_line_len *= 2;
		line = (char *) realloc(line,max_line_len);
		len = (int) strlen(line);
		if(fgets(line+len,max_line_len-len,input) == NULL)
			break;
	}
	return line;
}

void exit_input_error(int line_num)
{
	fprintf(stderr,"Wrong input format at line %d\n", line_num);
	exit(1);
}
double predict(char *string)
//void predict(FILE *input, FILE *output)
{
	int correct = 0;
	int total = 0;
	double error = 0;
	double sump = 0, sumt = 0, sumpp = 0, sumtt = 0, sumpt = 0;

	int svm_type=svm_get_svm_type(model);
	int nr_class=svm_get_nr_class(model);
	double *prob_estimates=NULL;
	int j;

/*	if(predict_probability)
	{
		if (svm_type==NU_SVR || svm_type==EPSILON_SVR)
			printf("Prob. model for test data: target value = predicted
value + z,\nz: Laplace distribution e^(-
|z|/sigma)/(2sigma),sigma=%g\n",svm_get_svr_probability(model));
		else
		{
			int *labels=(int *) malloc(nr_class*sizeof(int));
			svm_get_labels(model,labels);
			prob_estimates = (double *)
malloc(nr_class*sizeof(double));
			fprintf(output,"labels");
			for(j=0;j<nr_class;j++)
				fprintf(output," %d",labels[j]);
			fprintf(output,"\n");
			free(labels);
		}
	}
*/
	//max_line_len = 1024;
	//line = (char *)malloc(max_line_len*sizeof(char));
	//while(readline(input) != NULL)
	//{
	int i = 0;
	double target_label, predict_label;
	char *idx, *val, *label, *endptr;
	int inst_max_index = -1; // strtol gives 0 if wrong format, and 
precomputed kernel has <index> start from 0
	//label = strtok(line," \t\n");
	label = strtok(string," \t\n");
	if(label == NULL) // empty line
		exit_input_error(total+1);
	target_label = strtod(label,&endptr);
	if(endptr == label || *endptr != '\0')
		exit_input_error(total+1);
	while(1)
	{
		if(i>=max_nr_attr-1)	// need one more for index = -1
		{
			max_nr_attr *= 2;
			x = (struct svm_node *) realloc(x,max_nr_attr*sizeof(struct 
svm_node));
		}
		idx = strtok(NULL,":");
		val = strtok(NULL," \t");
		if(val == NULL)
			break;
		errno = 0;
		x[i].index = (int) strtol(idx,&endptr,10);
		if(endptr == idx || errno != 0 || *endptr != '\0' || x[i].index 
<= inst_max_index)
			exit_input_error(total+1);
		else
			inst_max_index = x[i].index;
			errno = 0;
		x[i].value = strtod(val,&endptr);
		if(endptr == val || errno != 0 || (*endptr != '\0' && 
!isspace(*endptr)))
			exit_input_error(total+1);

		++i;
	}

	x[i].index = -1;
	if (predict_probability && (svm_type==C_SVC || svm_type==NU_SVC))
	{
		predict_label = svm_predict_probability(model,x,prob_estimates);
		//fprintf(output,"%g",predict_label);
		//for(j=0;j<nr_class;j++)
		//	fprintf(output," %g",prob_estimates[j]);
		//fprintf(output,"\n");
	}
	else
	{
		predict_label = svm_predict(model,x);
		//fprintf(output,"%g\n",predict_label);
	}

	if(predict_label == target_label)
		++correct;
	error += (predict_label-target_label)*(predict_label-target_label);
	sump += predict_label;
	sumt += target_label;
	sumpp += predict_label*predict_label;
	sumtt += target_label*target_label;
	sumpt += predict_label*target_label;
	++total;

	if (svm_type==NU_SVR || svm_type==EPSILON_SVR)
	{
		printf("Mean squared error = %g (regression)\n",error/total);
		printf("Squared correlation coefficient = %g (regression)\n",
			((total*sumpt-sump*sumt)*(total*sumpt-sump*sumt))/
			((total*sumpp-sump*sump)*(total*sumtt-sumt*sumt))
			);
	}
	//else
		//printf("Accuracy = %g%% (%d/%d) (classification)\n",
			// (double)correct/total*100,correct,total);
	if(predict_probability)
		free(prob_estimates);
	return predict_label;
}

void exit_with_help()
{
	printf(
	"Usage: svm-predict [options] test_file model_file output_file\n"
	"options:\n"
	"-b probability_estimates: whether to predict probability estimates, 0
or 1 (default 0); for one-class SVM only 0 is supported\n"
	);
	exit(1);
}

void calc_max_index(char * restore_filename)
{
	FILE *fp_restore = fopen(restore_filename, "r");
	int idx, c;

	fp_restore = fopen(restore_filename,"r");
	if(fp_restore==NULL)
	{
		fprintf(stderr,"can't open file %s\n", restore_filename);
		exit(1);
	}
	c = fgetc(fp_restore);
	if(c == 'y')
	{
		readline(fp_restore);
		readline(fp_restore);
		readline(fp_restore);
	}
	readline(fp_restore);
	readline(fp_restore);
	while(fscanf(fp_restore,"%d %*f %*f\n",&idx) == 1)
		max_index = max(idx,max_index);
	rewind(fp_restore);
	fclose(fp_restore);
}

void calc_feature_range(char * restore_filename)
{
	FILE *fp_restore = fopen(restore_filename, "r");
	int idx, c;
	double fmin, fmax;

	if((c = fgetc(fp_restore)) == 'y')
	{
		fscanf(fp_restore, "%lf %lf\n", &y_lower, &y_upper);
		fscanf(fp_restore, "%lf %lf\n", &y_min, &y_max);
		y_scaling = 1;
	}
	else
		ungetc(c, fp_restore);

	if (fgetc(fp_restore) == 'x') {
		fscanf(fp_restore, "%lf %lf\n", &lower, &upper);
		while(fscanf(fp_restore,"%d %lf %lf\n",&idx,&fmin,&fmax)==3)
		{
			if(idx<=max_index)
			{
				feature_min[idx] = fmin;
				feature_max[idx] = fmax;
			}
		}
	}
	fclose(fp_restore);
}

void output_target(double value, char *str)
{
	int n;
	if(y_scaling)
	{
		if(value == y_min)
			value = y_lower;
		else if(value == y_max)
			value = y_upper;
		else value = y_lower + (y_upper-y_lower) *
				(value - y_min)/(y_max-y_min);
	}
	n = sprintf(str, "%g ",value);
	//printf(" %s", str);
	//printf("%g ",value);

}

void outputfunc(int index, double value, char *str)
{
	/* skip single-valued attribute */
	char *s  = (char *)malloc(1024*sizeof(char));
	if(feature_max[index] == feature_min[index])
```

```
            return ;

      if(value == feature_min[index])
            value = lower;
      else if(value == feature_max[index])
            value = upper;
      else
            value = lower + (upper-lower) *
                  (value-feature_min[index])/
                  (feature_max[index]-feature_min[index]);

      if(value != 0)
      {
            int n = sprintf(s, "%d:%g ",index, value);
            strcat(str,s);
      //    printf("%d ",n);
            new_num_nonzeros++;
      }
      //printf("%s", str);

}

double svmpredict(char *string)
//int main(int argc, char **argv)
{
      FILE *input, *output;
      char * p=string;
      char *scaled_string = (char *)malloc(1024*sizeof(char));


      int i;
      double d;
      int next_index=1, index;
      double target;
      double value;
#define SKIP_TARGET\
      while(isspace(*p)) ++p;\
      while(!isspace(*p)) ++p;

#define SKIP_ELEMENT\
      while(*p!=':') ++p;\
      ++p;\
      while(isspace(*p)) ++p;\
      while(*p && !isspace(*p)) ++p;

      //input = fopen("input", "w");
      //fprintf(input, "%s", string);
      max_index = 0;
      calc_max_index("range");
      feature_max = (double *)malloc((max_index+1)* sizeof(double));
      feature_min = (double *)malloc((max_index+1)* sizeof(double));

      if(feature_max == NULL || feature_min == NULL)
      {
            fprintf(stderr,"can't allocate enough memory\n");
            exit(1);
      }

      for(i=0;i<=max_index;i++)
```

```
	{
		feature_max[i]=-DBL_MAX;
		feature_min[i]=DBL_MAX;
	}

	calc_feature_range("range");

	/*Scale */
	sscanf(p,"%lf",&target);
	//printf("%s", p);
	SKIP_TARGET
	//output_target(target);
	output_target(target, scaled_string);
	//printf("Scaled_STRING: %s", scaled_string);
	while(sscanf(p,"%d:%lf",&index,&value)==2)
	//sscanf(p,"%d:%lf",&index,&value);
	{
	//	printf("\nindex: %d", index);
		//printf("\nvalue: %lf", value);
		for(i=next_index;i<index;i++)
			outputfunc(i,0, scaled_string);

		outputfunc(index,value, scaled_string);

		//outputfunc(index, value);
		SKIP_ELEMENT
		next_index=index+1;
	}

	for(i=next_index;i<=max_index;i++)
		//outputfunc(i, 0);
		outputfunc(i,0, scaled_string);

	//printf("%s\n", scaled_string);
	// parse options
	/*for(i=1;i<argc;i++)
	{
		if(argv[i][0] != '-') break;
		++i;
		switch(argv[i-1][1])
		{
			case 'b':
				predict_probability = atoi(argv[i]);
				break;
			default:
				fprintf(stderr,"Unknown option: -%c\n", argv[i-
1][1]);
				exit_with_help();
		}
	}


	if(i>=argn-2)
		exit_with_help();

	input = fopen(argv[i],"r");
	if(input == NULL)
	{
		fprintf(stderr,"can't open input file %s\n",argv[i]);
```

```
            exit(1);
        }

        output = fopen(argv[i+2],"w");
        if(output == NULL)
        {
            fprintf(stderr,"can't open output file %s\n",argv[i+2]);
            exit(1);
        }
        */

        if((model=svm_load_model("model.txt"))==0)
        //if((model=svm_load_model(argv[i+1]))==0)
        {
            printf("can't open model file %s\n");
            exit(1);
        }

        x = (struct svm_node *) malloc(max_nr_attr*sizeof(struct svm_node));
        if(predict_probability)
        {
            if(svm_check_probability_model(model)==0)
            {
                //fprintf(stderr,"Model does not support probabiliy 
estimates\n");
                printf("Model does not support probabiliy estimates\n");
                exit(1);
            }
        }
        else
        {
            if(svm_check_probability_model(model)!=0)
                printf("Model supports probability estimates, but disabled 
in prediction.\n");
        }
        d = predict(scaled_string);
        svm_free_and_destroy_model(&model);
        free(x);
        free(line);
        //fclose(input);
        //fclose(output);
        return d;
}


/*int main10()
{
        char s[50] = "-1  1:0.002     2:-0.699999 3:221 ";
        double d = svmpredict(s);
        printf("%d", d);
        return 0;

}*/
```

REFERENCES

1. C.A.Bouman, “Digital Image Processing”, January10, 2011
2. Huazhong Ning,Tony X. Han,Yuxiao Hu, Zhenqiu Zhang,Yun Fu, and Thomas S. Huang, “A Real time Shrug Detector”, Proceedings of the 7th International Conference on Automatic Face and Gesture Recognition (FGR’06), 2006
3. Hatice Gunes and Massimo Piccardi, “Automatic Temporal Segment Detection and Affect Recognition From Face and Body Display”, IEEE Transactions on Systems, Man, and Cybernetics, February 2009
4. C.N.Joseph, S. Kokulakumaran, K. Srijeyanthan, A. Thusyanthan, C. Gunasekara, and C.D. Gamage, “A Framework for Whole-Body Gesture Recognition from VideoFeeds”, 5th International Conference on Industrial and Information Systems (ICIIS 2010), Jul 29- Aug 01, 2010
5. John G. Allen, Richard Y. D. Xu, Jesse S. Jin, “Object Tracking Using CamShift Algorithm and Multiple Quantized Feature Spaces”, Pan-Sydney Area Workshop on Visual Information Processing VIP2003, 2003
6. Mousa Mojarrad, Mashallah Abbasi Dezfouli, and Amir Masoud Rahmani, “Feature’s Extraction of Human Body Composition in Images by Segmentation Method”, World Academy of Science, Engineering and Technology, 2008
7. Hatice Gunes and Massimo Piccardi, “Fusing Face and Body Gesture for Machine Recognition of Emotions”, IEEE International Workshop on Robots and Human Interactive Communication, 2005
8. K. Sugawara. Weighted hough transform on a gridded image plane. In *ICDAR97*, pages 701–704, 1997.
9. D.M. Gavrila and L.S. Davis, “3-D model-based tracking of humans in action: a multi-view approach”, IEEE Computer Vision and Pattern Recognition, San Francisco, 1996.
10. Chi-Wei Chu, Isaac COHEN, ”Posture and Gesture Recognition using 3D Body Shapes Decomposition”, IEEE Workshop on Vision for Human-Computer Interaction (V4HCI),June 21, 2005.

APPENDICES

APPENDIX A: Tools Description

OpenCV is a library of programming functions mainly aimed at real time computer vision, developed by Intel and now supported by Willow Garage. It is free for use under the sources. The library is cross-platform. It focuses mainly on *real-time* image processing. If the library finds Intel's Integrated Performance Primitives on the system, it will use these commercial optimized routines to accelerate it.

OpenCV's application areas include:

- 2D and 3D feature toolkits
- Egomotion estimation
- Facial recognition system
- Gesture recognition
- Human-Computer Interface (HCI)
- Mobile robotics
- Motion understanding
- Object Identification
- Segmentation and Recognition
- Stereopsis Stereo vision: depth perception from 2 cameras

- Structure from motion (SFM)
- Motion tracking

To support some of the above areas, OpenCV includes a statistical machine learning library that contains:

- Boosting
- Decision tree learning
- Expectation-maximization algorithm
- k-nearest neighbor algorithm
- Naive Bayes classifier
- Artificial neural networks
- Random forest

Microsoft Visual Studio 2008

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It can be used to develop console and graphical user interface applications along with Windows Forms applications, web sites, web applications, and web services in both native code together with managed code for all platforms supported by Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework and Microsoft Silverlight.

Visual Studio supports different programming languages by means of language services, which allow the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Built-in languages include C/C++ (via Visual C++), VB.NET (via Visual Basic .NET), C# (via Visual C#), and F# (as of Visual Studio 2010). Support for other languages such as M,Python, and Ruby among others is available via language services installed separately. It also supports XML/XSLT, HTML/XHTML, Java Script and CSS. Individual language-specific versions of Visual Studio also exist which provide more limited language services to the user: Microsoft Visual Basic, Visual J#, Visual C#, and Visual C++.

LIBSVM is an integrated software for support vector classification, (C-SVC, nu-SVC), regression (epsilon-SVR, nu-SVR) and distribution estimation (one-class SVM). It supports multi-class classification.

Since version 2.8, it implements an SMO-type algorithm proposed in this paper: R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using second order information for training SVM. Journal of Machine Learning Research 6, 1889-1918, 2005. You can also find a pseudo code there. (how to cite LIBSVM)

Their goal is to help users from other fields to easily use SVM as a tool. LIBSVM provides a simple interface where users can easily link it with their own programs. Main features of LIBSVM include

- Different SVM formulations
- Efficient multi-class classification
- Cross validation for model selection
- Probability estimates
- Various kernels (including precomputed kernel matrix)
- Weighted SVM for unbalanced data
- Both C++ and Java sources
- GUI demonstrating SVM classification and regression

- Python, R, MATLAB, Perl, Ruby, Weka, Common LISP, CLISP, Haskell, LabVIEW, and PHP interfaces. C# .NET code and CUDA extension is available.
  It's also included in some data mining environments: Rapid Miner and PCP.
- Automatic model selection which can generate contour of cross valiation accuracy.

APPENDIX B: Quality Assurance

Quality assurance, or QA for short, is the systematic monitoring and evaluation of the various aspects of a project, service or facility to maximize the probability that minimum standards of quality are being attained by the production process. QA cannot absolutely guarantee the production of quality products. Two principles included in QA are: "Fit for purpose" - the product should be suitable for the intended purpose; and "Right first time" - mistakes should be eliminated.

Fit for purpose:

Our project is fit for the purpose of developing a conversational agent which can interact with an individual. It can accurately recognize the emotions of the individual and respond accordingly. It is a step forward to make computers emotionally intelligent so that they behave more like humans and become an integral part of our lives.

Right first Time:

Our project gives accurate results under appropriate conditions such as:

- The camshift works accurately only when the background does not have skin colored elements.
- The subject should be wearing full- sleeves for effective output.

Failure testing:

The project fails under the following conditions:

- If the background has skin coloured elements.
- Training of the classifier is not proper.

Programming style and testing

Various types of testing such as functional, white box and unit testing were carried out using debugging features of Microsoft Visual Studio 2008 (break point) and language tools such as printing the values of the variables.

Example:

```
while(sscanf(p,"%d:%lf",&index,&value)==2)


        {

        //        printf("\nindex: %d", index);
```

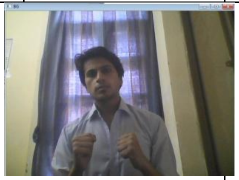
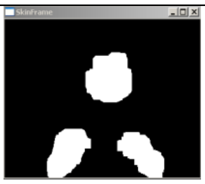
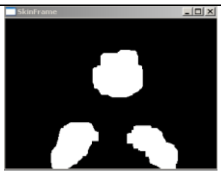
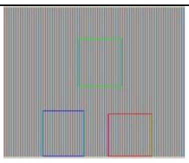
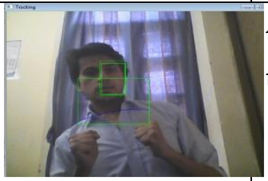
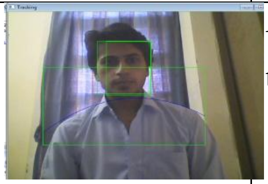
```
			//printf("\nvalue: %lf", value);

			for(i=next_index;i<index;i++)

				outputfunc(i,0, scaled_string);


			outputfunc(index,value, scaled_string);



			//outputfunc(index, value);

			SKIP_ELEMENT

			next_index=index+1;

		}
```

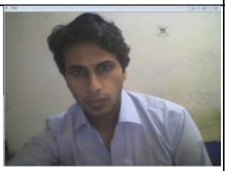
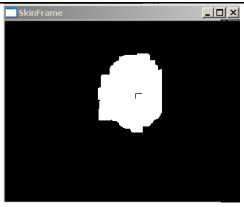
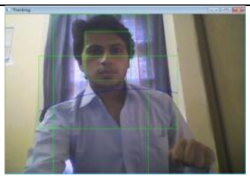
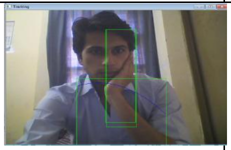
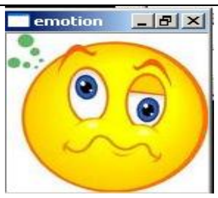
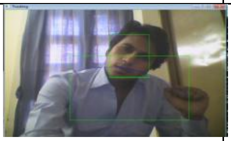
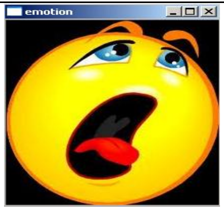
Apart from the above concepts we used the following tools and methods to for quality assurance:

- Extensive research work to know the detailed process of development
- Use of a log book to ensure proper requirements and documentation

- Make some policies of not to mix professionalism with other casual things.
- Invite feedbacks from student users to eliminate minor difficulties which were not visible to us as a developer.
- Learn from the mistake of other fellow students.
- Keep project on schedule to prevent haywire in the later stages.

APPENDIX C: Test cases

S.NO.	OBJECTIVE	PASS/ FAIL	DATA INPUT	EXPECTED OUTPUT	ACTUAL RESULT	REMARK
1	Proper skin segmentation	Pass	 Video Frame	Black and white image consisting of skin component	 Black and white image consisting of skin components	Proper skin segmentation achieved
2	Proper contour extraction	Pass	 Skin segmented image	Bounding boxes for head and hands	 Bounding boxes for head and hands	Proper contour extraction achieved
3	Pose Variance	Pass	Captured Frame	Detected face from every angle	Detected face from every angle	Pose Variance achieved
4	Real time movement tracking	Pass	Captured Frame	All components tracked efficiently	 All components tracked	**Accurate tracking**
5	Shoulder Detection and Tracking	Pass	Captured Frame	All components tracked efficiently	 Shoulder tracked efficiently	Accurate tracking

6	Light Intensity Variance	Fail	Captured Frame	All Components Tracked Efficiently	No effective tracking	No Effective tracking
7	Skin Segmentation with non uniform background	Pass		Skin Segmented effectively		Skin Segmented effectively
8	Tracking with non uniform background	Pass	Captured Frame	Accurate Tracking		Accurate Tracking
9	Tracking with non uniform background	Fail	Captured Frame	Accurate Tracking	Program break	Haphazard Tracking
10	Tensed Emotion classification	Pass		Emotion classified as Tension		Correct classification
11	Tired Emotion classification	Pass		Emotion classified as Tired		Correct Classification