

MULTI-MODAL EXPRESSION RECOGNITION

by

SRIVARDAN CHANDRAPATI

B. Tech., Dr Babasaheb Ambedkar Technological University, 2005

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Mechanical and Nuclear Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2008

Approved by:

Major Professor
Dr Akira Tokuhira

Copyright

SRIVARDHAN CHANDRAPATI

2008

Abstract

Robots will eventually become common everyday items. However before this becomes a reality, robots would need to learn to be socially interactive. Since humans communicate much more information through expression than through actual spoken words, expression recognition is an important aspect in the development of social robots. Automatic recognition of emotional expressions has a number of potential applications other than just social robots. It can be used in systems that make sure the operator is alert at all times, or it can be used for psycho-analysis or cognitive studies. Emotional expressions are not always deliberate and can also occur without the person being aware of them. Recognizing these involuntary expressions provide an insight into the person's thought, state of mind and could be used as indicators for a hidden intent. In this research we developed an initial multi-modal emotion recognition system using cues from emotional expressions in face and voice. This is achieved by extracting features from each of the modalities using signal processing techniques, and then classifying these features with the help of artificial neural networks. The features extracted from the face are the eyes, eyebrows, mouth and nose; this is done using image processing techniques such as seeded region growing algorithm, particle swarm optimization and general properties of the feature being extracted. In contrast features of interest in speech are pitch, formant frequencies and mel spectrum along with some statistical properties such as mean and median and also the rate of change of these properties. These features are extracted using techniques such as Fourier transform and linear predictive coding. We have developed a toolbox that can read an audio and/or video file and perform emotion recognition on the face in the video and speech in the audio channel. The features extracted from the face and voices are independently classified into emotions using two separate feed forward type of artificial neural networks. This toolbox then presents the output of the artificial neural networks from one/both the modalities on a synchronized time scale. Some interesting results from this research is consistent misclassification of facial expressions between two databases, suggesting a cultural basis for this confusion. Addition of voice component has been shown to partially help in better classification.

Table of Contents

List of Figures.....	viii
List of Tables.....	x
Acknowledgements.....	xi
Dedication	xii
CHAPTER 1 - INTRODUCTION.....	1
CHAPTER 2 - FACIAL EXPRESSION RECOGNITION (FER).....	4
FACE DATABASE	6
Japanese Female Facial Expression Database (JAFFE).....	6
MMI Database	7
FEATURE POINT EXTRACTION FROM SNAPSHOTS	8
Face Region Separation.....	9
Feature Point Location	12
Particle Swarm Optimization.....	12
Cost Functions.....	14
Other Techniques used	15
Results of Feature Point Extraction.....	16
Construction of Vector	17
FEATURE POINT TRACKING IN VIDEOS.....	19
NEURAL NETWORKS FOR CLASSIFICATION.....	20
Training of ANN.....	25
Testing of ANN.....	27
RESULTS of FER	27
Training on JAFFE and Testing on JAFFE	27
Training on JAFFE and Testing on MMI.....	28
Training on MMI and Testing on JAFFE.....	31
Testing on Elderly Faces	32
ANALYSIS of FER	34
CHAPTER 3 - EMOTION RECOGNITION IN SPEECH	38

SPEECH DATABASE.....	38
SPEECH PROCESSING.....	41
Spectrogram.....	41
Word Separation	43
Mel Frequency Spectrum.....	44
Formant Extraction using LPC	46
Speech Generation in Humans.....	46
Filter Designing using Linear Predictive Coding (LPC).....	50
Formant Extraction.....	52
CLASSIFICATION OF EXPRESSION.....	52
Input Vector used with ANN	52
Training, Validation and testing sets.....	53
Results and Discussion.....	53
CHAPTER 4 - MULTI-MODAL EXPRESSION RECOGNITION.....	56
MULTIMODAL FUSION	57
Signal level	57
Feature level.....	57
Decision or Conceptual level.....	58
EMOTION RECOGNITION TOOLBOX	59
Instruction Manual	59
TESTING and ANALYSIS	63
CHAPTER 5 - Conclusion	67
References And Bibliography	72
Appendix A - MATLAB Code.....	78
Filename : integrate.m.....	78
Filename : mmread.m	86
Filename : distanceDiffVector.m.....	94
Filename : distanceDirVector1.m.....	94
Filename : setAudio.m	98
Filename : featureExtraction.m	104
Filename : computeSpectrum.m.....	107

Filename : wordSeperation.m.....	109
Filename : melfiltermatrix.m.....	110
Filename : freq2mel.m	113
Filename : mel2freq.m	113
Filename : Peak.m.....	114
Filename : computeFormant.m.....	115
Filename : loglimit.m.....	118
Filename : computeMelSpectrum.m.....	119
Filename : setVideo.m	120
Filename : getFaceRegion.m.....	158
Filename : checkBackground.m	161
Filename : seedPixel.m	162
Filename : check3.m	163
Filename : testFrame.m.....	164
Filename : Pso2_eye.m	172
Filename : cost_function7subReg.m.....	177
Filename : getRightEyeCorners.m.....	182
Filename : Pso_eyebrow.m	184
Filename : cost_functionEyebrow.m	188
Filename : getRightEyebrowCorner.m	191
Filename : getLeftEyeCorners.m.....	193
Filename : getLeftEyebrowCorner.m	195
Filename : Pso2_mouth_left.m.....	197
Filename : cost_functionMouthL2.m.....	203
Filename : Pso2_mouth_right.m.....	205
Filename : mouth_fix.m	211
Filename : getMouthLipsCoordinates.m.....	214
Filename : Pso_nose.m.....	216
Filename : cost_functionNose.m	221
Filename : getNoseCorners.m	224
Filename : run2.m.....	227

Filename : Pso2_eye_run.m	237
Filename : getRightEyeCorners2.m.....	241
Filename : Pso_eyebrow_run2.m	243
Filename : getLeftEyeCorners2.m.....	249
Filename : Pso2_mouth_left_run.m.....	251
Filename : Pso2_mouth_right_run.m.....	257
Filename : cost_functionMouthR2.m	264
Filename : Pso_nose_run_vector.m.....	267
Appendix B - Facial Action Coding System.....	272
Appendix C - Neural Network as a Useful Tool for Real-Time Facial Expression Recognition	279
Appendix D - Defense Presentation.....	290

List of Figures

Figure 1.1 Temporal Characteristics of Emotional Categories [14].....	1
Figure 2.1 Setup used to photograph facial expression for JAFFE database.....	7
Figure 2.2 Example of facial expression form JAFFE database	7
Figure 2.3 Example of facial expression images from MMI database	8
Figure 2.4 Seed Pixel Neighbor configuration; (a) 8 neighbor (b) 4 neighbor	10
Figure 2.5 Blur Image and Edges on Image.....	10
Figure 2.6 Original Image and Edges on Image.....	11
Figure 2.7 Background Separation	11
Figure 2.8 Face Region	12
Figure 2.9 Left Mouth Corner Template.....	15
Figure 2.10 Eye Corner Estimation	15
Figure 2.11 Eyebrow Corner Estimation	16
Figure 2.12 Upper and Lower Lip Estimation.....	16
Figure 2.13 Feature Points	17
Figure 2.14 Feature Mask.....	18
Figure 2.15 Evolution of Facial Expressions Over Time.....	19
Figure 2.16 Biological Neuron	20
Figure 2.17 Basic Component of Artificial Neural Network.....	21
Figure 2.18 Identity Function.....	22
Figure 2.19 Step Function.....	22
Figure 2.20 Sigmoid Function ($\alpha = 1$)	23
Figure 2.21 Hyper-Tangent Function ($\alpha = 1$).....	23
Figure 2.22 Recurrent Network.....	24
Figure 2.23 Three Layered Feed-Forward Network [63]	25
Figure 2.24 Normalizing Expression Recognition in Video Sequences.....	30
Figure 2.25 Examples of Images of the Elderly Gentleman	32
Figure 2.26 Expression Recognition in Elderly Faces.....	33

Figure 2.27 Comparing Anger and Disgust Expressions.....	36
Figure 2.28 Comparing Surprise and Fear Expressions.....	37
Figure 3.1 Photograph during recording in anechoic chamber at TU-Berlin. [7]	39
Figure 3.2 Recognition rate for various emotions in database [7].....	40
Figure 3.3 A Spectrogram	41
Figure 3.4 Power plot with word separation	44
Figure 3.5 Mel frequency Scale.....	45
Figure 3.6 Mel frequency filterbank [58].....	45
Figure 3.7 Mel frequency filter bank as a matrix	46
Figure 3.8 Schematic of the Human Speech Production System	47
Figure 3.9 Linear Speech Production Model.....	48
Figure 3.10 Speech Feature Extraction [48].....	53
Figure 3.11 Emotion Recognition in Speech.....	54
Figure 4.1 Ideal Emotion Recognition System.....	56
Figure 4.2 Multimodal fusion at Signal level.....	57
Figure 4.3 Multimodal fusion at Feature level	58
Figure 4.4 Multimodal fusion at Decision level	58
Figure 4.5 Selecting a Media File	59
Figure 4.6 Reading a media file.....	60
Figure 4.7 Audio Toolbox.....	61
Figure 4.8 Video Toolbox	62
Figure 4.9 Emotion Recognition Toolbox	63
Figure 4.10 Comparing Multi-Modal Results for Disgust and Anger	66
Figure B.1 Anatomy of Facial Muscles	272

List of Tables

Table 2.1 True Positive rate for ANN trained and tested on JAFFE database.....	27
Table 2.2 True Positive rate for ANN trained on JAFFE and tested on MMI database.....	29
Table 2.3 True Positive rate for ANN trained on MMI and tested on JAFFE database.....	31
Table 2.4 Confusion Matrix for ANN trained on JAFFE and tested on MMI database.....	35
Table 2.5 Confusion Matrix for ANN trained on MMI and tested on JAFFE database.....	36
Table 3.1 Sentences in speech database [7]	40
Table 3.2 Confusion Matrix for Emotion recognition in Voice	54
Table 3.3 Recognition Rate for Emotions in Voice.....	55
Table B.1 Facial Action Coding System.....	273

Acknowledgements

I take this opportunity to firstly thank my major professor Dr. Akira Tokuhira for his continuous support and guidance through these two years, without which I would have not been able to complete this research.

I would also like to thank Dr. Dale Schinstock, Associate Professor, Mechanical and Nuclear Engineering and Dr. David Gustafson, Professor, Computing and Information Sciences for agreeing to be on my committee and helping me with my research whenever I needed their assistance.

I thank the members of the IDEAS lab and fellow graduate students for being really awesome friends and making it a fun and challenging place to work and learn, and always pushing the bar higher in the spirit of healthy competition.

I grateful to the faculty and staff at the Department of Mechanical and Nuclear Engineering for assisting me in their own capacities. Also to all the other members of the K-State family who make working at K-State so much easier.

I am greatly obliged to the K-State Center on Aging and its director Dr Gayle Doll, Dr. William Dunn, Department of Mechanical and Nuclear Engineering at K-State and M2 Technologies for their financial support during the course of this research.

I would like to thank Ms. Victoria Bañales, who volunteered to provide us with some video clips of herself, so that we could perform some preliminary tests on the system we developed.

Last but not the least, to family and friends for being very supportive and have made these two years I spent in Manhattan fun-filled and very memorable.

Dedication

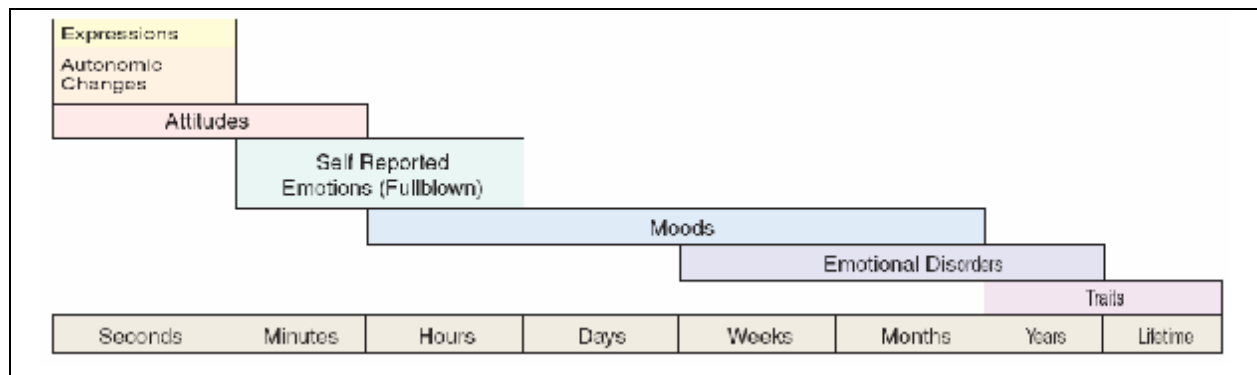
I dedicate this thesis to my parents and sister to whom I owe my success.

CHAPTER 1 - INTRODUCTION

Robot as defined by Kaplan “*is an object that possesses the following three properties: It is a Physical object, it functions in an Autonomous and Situated manner*”. By situated manner he means that the physical and social environment perceived by the robot has a direct influence on it [32]. A lot of research has gone into integrating the robot with its physical environment. With the various sensors, such as proximity sensors, accelerometers and strain gauges, the physical integration of the robot and environment is incrementally becoming a reality. However robots today are still limited mostly to industrial applications or as toys for the technically inclined. To make robots more universally acceptable so that they can coexist with human in the same environment their ability to interact socially or at least understand the social environment is important. In face to face human interaction, facial expressions carry approximately, 55%, and voice intonations, 38%, of the message, while only 7% percent of the message is carried by the actual words [45]. Therefore for robots to be socially acceptable they need to be able to recognize emotions revealed by facial expressions and voiced intonations.

Research has shown that human expressions are short lived emotional states and the changes in expressions are indicators to the subjective feeling and action tendencies towards the current issues. For example if an assistive feeding robot sees that a person has a look of disgust seeing a particular entrée it could refrain from feeding it. Other emotional states can last from a few seconds such as expressions to attitude lasting minutes, moods sometimes lasting for weeks or months, emotional disorders lasting for years and traits lasting a lifetime (Figure 1.1).

Figure 1.1 Temporal Characteristics of Emotional Categories [14]



Emotions are a group of processes involving five different components, comprising of subjective feeling, cognition, motor expression, action tendencies and neurological processes [17]. Traditional research in emotion detection consists of stimulating the subjective feeling or cognition of the subject (person). For example, showing graphic images to induce changes in motor expression and neurological processes from an apriori state. Along with the general curiosity about human nature, this research was also directed towards detection of deception [23].

Humans have tried to detect deception and/or intent to deceive for many years. For example in western Africa people suspected of a crime were made to pass a bird's egg from one nest to another and if they broke it they were considered guilty. In ancient China a suspect was made to hold a handful of rice in their mouth during the prosecutor's speech, and if there was any rice left at the end of the speech then he/she was declared guilty. All these methods were meant to be indicators of anxiety that accompanies the telling of lies and/or trying to deceive in varying degrees. The African method was a measure of how nervous the subject was when confronted the truth; the Chinese method relied on the idea that the mouth went dry during times of emotional anxiety [60].

Today law enforcement, security, intelligence and other related agencies rely on an age old technology developed in 1931 called the polygraph test to determine truthful/deceitful accounts. The polygraph test measures various physiological states of the subject namely blood pressure, heart rate and skin conductivity in response to a series of carefully controlled questions. The results of this test are largely based on the skill of the test supervisor [60]. The US Supreme Court has left it to individual jurisdictions whether to accept polygraphs as admissible evidence, and in 2007 it was admitted in 19 states at the discretion of the trial judge and with stipulations. In most European jurisdictions it is not used by the police force, as they do not consider them to be reliable.

Research on deception detection using cues from expression of emotion [67] [23] and advances in computing technology and processing speed make it possible to consider an automated deception detection system. Before we can build a system to successfully detect deception, we have to first build a robust automated expression recognition system that can recognize emotions from various modalities such that information from one "channel" can be verified and validated by another, and also to accommodate "failure" of one channel by another.

In this research we make an effort to build a multi-modal emotion recognition system using cues from facial expression recognition and voiced characteristics, that until recently were left to the judgment of the person conducting the polygraph test. We anticipate that an automated emotion recognition system would facilitate the development of human computer interaction. In fact applications such as monitoring operator efficiency in critical situations or developing first generation social robots are near-term goals [61]. Our task is thus to recognize the six basic expressions (happiness, anger, disgust, fear, surprise and sadness) that are common in the human experience [19]. All other expressions are learned from the environment [15].

The objectives of this research are as follows:

- To build a digital facial expression recognition system
- Build a system for recognition of emotions in digitally recorded voice
- Integrate the two into a multi-modal emotion recognition system
- Build an user friendly interface for the multi-modal emotion recognition system

This thesis is divided into 3 parts. In the Chapter 2 we will discuss facial expression recognition. We will firstly discuss the databases used, and then talk about feature extraction from the images and formation of feature mask. After which we will briefly discuss the basics of artificial neural networks, and their training and testing. Then we discuss the results and analysis of the same. We then proceed to expression recognition in speech (Chapter 3) where we will first talk about the database used. After which we will discuss word separation using spectrogram and power plot. After which we will see how to convert from frequency scale to mel scale and compute mel energy. We will now have a look at the human speech production system, draw analogies to a source filter model and filter design using linear predictive coding. Once we are done designing a vocal tract filter using linear predictive coding, we then extract features from the speech signal. After which we build a feature vector of expression classification in speech. We will then discuss results from expression recognition in speech before having a look at multimodal emotion recognition. In the third part (Chapter 4) we will study the various levels of multimodal fusion, and see how to use the emotion recognition toolbox we develop. After which we discuss how to classify expressions using the multimodal data.

CHAPTER 2 - FACIAL EXPRESSION RECOGNITION (FER)

Verbal communication is voluntary and controlled by the speaker, but this conveys only a portion of the full meaning of the message. Along with verbal communication there is also a significant amount of information that is conveyed through non-verbal channels such as facial expressions (approximately 50 percent of the effect of the message), while vocal intonations contain some 40 percent of the effect [45]. Being able to develop a system to recognize these expressions automatically would facilitate human computer interaction. That is in effect, better understand the needs of the person interacting with the computer. This improved man-machine communication would also make it easier for people with disabilities to use computers. There has been some research with focus many on facial expression recognition and others that are more specific, such as tracking the eyeball [38] or a whistling user interface [65].

It is self-evident that humans have very well developed communication skills, and thus developing an automated system to demonstrate all that a person can do, is a challenge in computational intelligence. One of these skills that we take for granted is the ability to recognize facial expressions. The range of human expressions and the cognitive states to which the expression is attached is large [28]. Hence it is not surprising that even for one expression, say a smile linked to a state of happiness, that every person has a slightly different way of expressing this emotion. An automated expression recognition system must be able to accommodate these variations. Additionally, there is evidence of differences for selected emotions (fear and surprise) amongst different ethnic groups and cultures [21]. While others, like happiness (smile) appear to be largely universal. However, as humans are able to distinguish between basic expressions/emotions (anger, disgust, happy and sad) irrespective of ethnicity and culture [21], it is reasonable to assert that there exists a common functional modality for many expressions. Discovering this modality and developing a tool to accurately extract an inherent “pattern” from an expression is the key to building flexible and robust facial expression recognition software.

Gladwell [27] described a face reader as a, “*professional person who has a gifted ability to pick up voluntary or involuntary facial expressions occurring within a very short time*”. With humans one has to consider the question of degree of truthful intent. By using a digital camera to

record the changes/changing expressions we want to be sure that we have a system that is reliable and accurate [29].

In fact Ekman [24] has reported that a ‘true’ smile can be held longer in time than a ‘forced’ smile. In fact herein lies one of the difficult challenges/issues in facial expression recognition and emotion.

Facial expression recognition systems can be classified into two categories: static recognition which classifies expressions in snapshots or individual frames of a video, and dynamic recognition that works on video sequences as a whole. The principal methods of static recognition are wavelet transformation, neural network based classification and principal component analysis. These methods are capable of analyzing only one frame at a time. The dynamic methods work by extracting the changes in the video, such as changing intensity within a particular region of the video or movement of features in the video, to achieve expression recognition. Two important methods of extracting these dynamics are optical flow models and hidden Markov models [71].

Another way to classify various automated facial expression recognition systems is via their method, namely a geometric feature based approach, and a holistic template matching method. In the geometric feature based approach a set of key feature points are located and the geometric relation of these points with each other is used to classify the expression. In contrast, the holistic template matching technique consists of a template of desired feature points superimposed on the image of interest and then deformed so as to match the image. The deformation of the template is then used to classify the image. A notable detailed survey of the various methods is that given by Fasel and Luetin [25] and also by Pantic and Rothkrantz [50]. However, as the number of feature points correlates to facial details, most of the reported approaches are computationally intensive and thus less than applicable in a real time environment. An exception perhaps is that developed by Littlewort et al. [40]. Considerable computing is needed for detailed image processing and feature point extraction. If an expression recognition system is to be deployed in the field it needs to perform its analysis in real time or quasi-real time.

Our objective has been to reduce the amount of computation required (relative to above) for the image preprocessing and create an input vector for an artificial neural network capable of recognizing facial expressions in real time with a relative accuracy in recognition, while leaving

more computational capacity to process speech for emotion recognition. There are a number of challenges involved in making the processing quicker; firstly we cannot extract a large number of facial feature points to form a dense grid for tracking subtle changes. Instead we search for the most distinguishable points on the face that can consistently be tracked spatiotemporally. We also exercise the option to limit the search for feature points if an initial “pass” does not easily reveal them. Thus, relative to facial expression recognition reported above, we expect at the onset, a lower overall accuracy in facial expression recognition due to the lower dimensionality of the vector representing the expression. This also presents a tougher learning task for the artificial neural network since there is inherently more noise in the data. Even with all these challenges, lower accuracy and more noise in the data, we believe that adding the voice component will maintain lower computation load yet contribute to near real time emotion recognition.

FACE DATABASE

Databases are usually collected in a controlled environment and hence suitable for development of a system before the system is tested in the field. Working with standard databases also gives us an opportunity to directly compare our results with those obtained by other researchers. Further a pre-coded database (facial expression and emotion), standardizes characterization or labeling such that it sidesteps subjective labeling.

Japanese Female Facial Expression Database (JAFFE)

The Japanese female facial expression database (JAFFE) was compiled by Lyons et. al. [41]. This database consists of a total of 219 images of 10 Japanese female subjects six basic expressions of angry, disgust, fear, happy, sad and surprise [20], and a neutral face.

Each of the subjects took their own pictures using a set up as shown in the Figure 2.1. The setup consisted of a semi-reflective plastic mirror placed in front of a camera, the subjects took their photographs while looking in this mirror. Tungsten lights were used to evenly illuminate the faces of the subjects. The camera was placed in a dark box so as to reduce back reflection in the semi-reflective mirror. The images were digitized using a flat bed scanner, after being printed in monochrome.

Figure 2.1 Setup used to photograph facial expression for JAFFE database

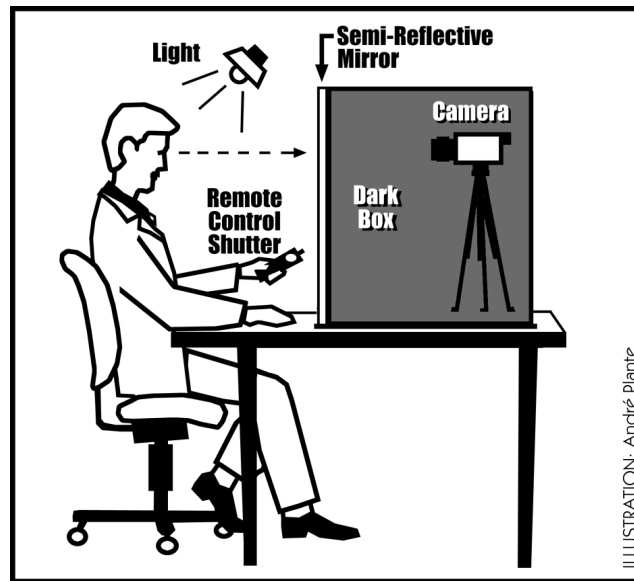
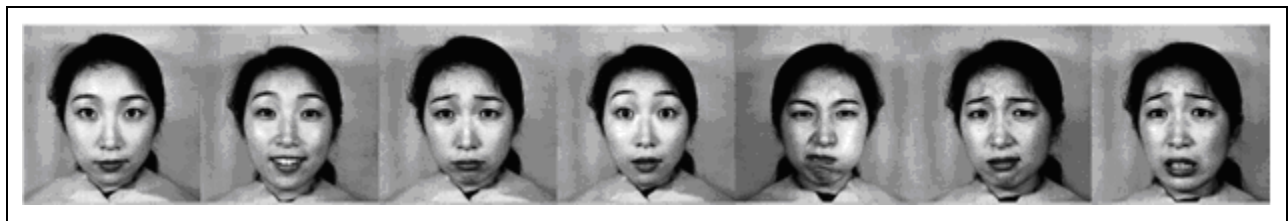


Figure 2.2 Example of facial expression form JAFFE database



The images were then rated for degree of each component of the expression by another group of 92 Japanese females. The rating group was divided into four subgroups; the first group of 31 was asked to rate 108 images for 6 basic expressions, while the second group of 31 was asked to rate the remaining 111 images. The third group and fourth group of 15 subjects each, were not shown the fear images and asked to rate the images for five expressions. Here, similar to the first two groups, the images shown to the third and fourth groups were also mutually exclusive.

MMI Database

Pantic et al. developed and compiled the MMI face database to address the lack of availability of an easily accessible standard database for researchers in the fields of facial expression recognition. The database consists in excess of 1500 still and video image sequences of various expressions in both frontal and profile view. The videos in this database are shot at a

standard rate of 24 frames per sec with the length of the video varying from 40 to 520 frames. These videos were shot in a under consistent lighting form two high intensity lamps which was diffused using reflective umbrellas. The subjects of the database were asked to display a number of different action units both individually and in combination with other action units. The complete database was then action unit coded [52], according to the facial action coding system (FACS) attributed to Ekman and Friesen [19] [22].

The facial action coding system is a technique developed to recognize and score action units. Action units represent facial muscular activity that momentarily changes the facial expression. Each expression is a combination of a number of action units that occur simultaneously. Tracking these changes in action units can be used for expression recognition, but will only add to computation since these changes are minute and would require a thorough preprocessing of the image. Appendix B contains several reference images of the facial actions units.

Figure 2.3 Example of facial expression images from MMI database



FEATURE POINT EXTRACTION FROM SNAPSHOTS

It is at the preprocessing stage when extracting the feature points that we significantly differ from Littlewort et al. Littlewort et al. began with 48 X 48 pixel images to obtain 92,160 possible features using 5 spatial and 8 orientations of the gabor filter across the image. The feature set was reduced by first extracting the best feature set for each emotion independently, the various feature sets were put together into a 538 dimensional vector [39].

We are looking at a computationally lighter system. Hence based on Bassili's work, we chose instead to locate selected points on the face. Bassili in experimentally recorded video of actors expressing various emotions, with their faces painted black and placed about 100 white markers on the face. These videos were then shown to other people, such that only the white markers were visible. With this experiment he showed that people could recognize emotions by

just looking at the movement in the cluster of points without the complete facial image. The mean accuracy of recognition of emotions reported by Bassili based on feature point motion was 33.33%, ranging from 75% for happy to 6% for fear. A chi square test was performed for statistical significance and the results were found to be significant compared to a random guess [5]. While the recognition rate is not high it does demonstrate that even with a “sparse” image the emotion can be recognized for select expressions and linked emotions. We extract 17 feature points that could be automatically extracted from the face, as shown by Chennamsetty in his M.S. thesis [12]. An assumption we make while extracting the feature points is that the image is of the frontal face and approximately centered in the frame. As a first generation system we decided to not to take into consideration out of plane rotation of the head. While an out of plane rotation provides us with a profile view of the person, it drastically reduces the number of feature points that can be extracted. It does give us a few additional more points such as the ears and the chin, but these points would be more important for identification purposes, than for expression recognition. Since one of the applications that this system of this system is in deception detection, the video recording will be in a controlled environment where frontal face images can be obtained.

Face Region Separation

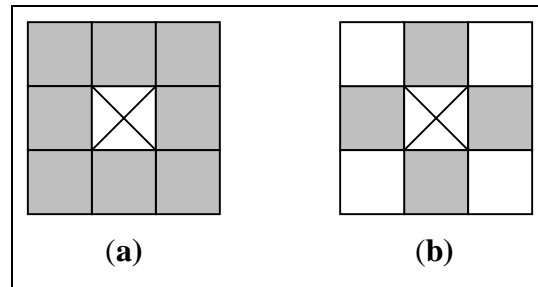
Face region separation is the first step in processing the image, which eventually leads to feature point location and then emotion recognition. In this stage the face is separated from the background of the image, so that at a later stage we can limit our search to the face region. The face region separation is achieved using a seeded region growing algorithm. The seeded region algorithm was first proposed by Adams and Bischof in 1994 [1]. We chose this method for its simplicity in implementation, and ability to accurately separate the complete face region, without including the background. This method begins with selection of the initial seeds such that they are located in the area of interest. We then begin with one of these seeds and test its neighboring pixels to see if the following threshold condition is satisfied. It is described by,

$$|I_s - I| < T$$

where, I_s is the intensity of the seed pixel, I is the intensity of the pixel under consideration, and T is the threshold intensity. In case a neighboring pixel satisfies the criteria it is added to the set of seed pixels and acts like a seed pixel in the next iteration. We do this until either none of the

seeds have a neighboring pixel that satisfies the criteria or we reach the borders of the image. There are two commonly used sets of neighbors, firstly the eight neighbor scheme where all the eight neighbors are considered, and secondly the four neighbor scheme where we look at only the pixels to the immediate right, left, top and bottom, and not at the diagonal pixels.

Figure 2.4 Seed Pixel Neighbor configuration; (a) 8 neighbor (b) 4 neighbor



Before we begin with the actual seeded pixel growing we have to first preprocess the image. Preprocessing begins with first scaling the image such that we use the full range of the gray scale, i.e. [0 255]. We then blur the image, using a Gabor filter, to get rid of the soft edges (Fig 2.5), because at the next stage when we use an edge detector, it is very sensitive to soft edges (Fig 2.6) and thus picks up a lot of edges. We use the canny edge detection algorithm [61][9] for this purpose; this is an inbuilt function in Matlab.

Figure 2.5 Blur Image and Edges on Image

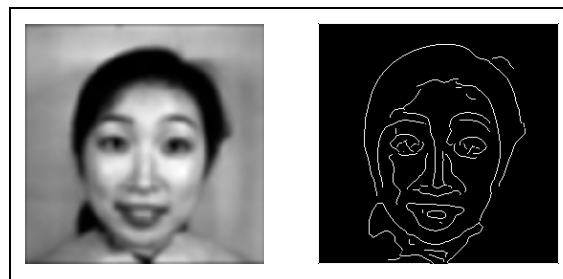
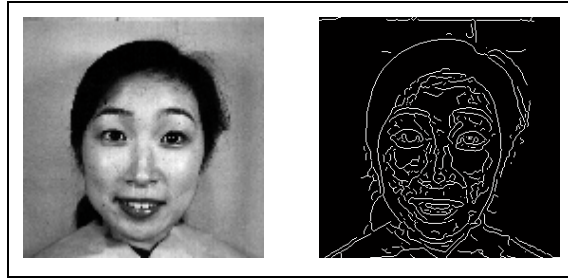
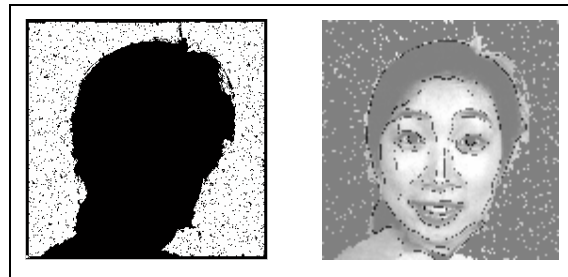


Figure 2.6 Original Image and Edges on Image



Once we have the edges, we subtract them from the original image so as to make all the edges black in the original image. This creates a gradient in the image and prevents the region from growing across the edge. However, this could grow around the edge if the regions are actually connected. Once we have done this we first use the seeded region growing to locate as much of the background as we can by using seed pixels at each of the four corners of the image and then painting the background black on the original image (Figure 2.7). This step helps when we do the seeded region growing for the face, since if the region attempts to grow outside the face, it will be limited by the black background and not grow to encompass the whole image.

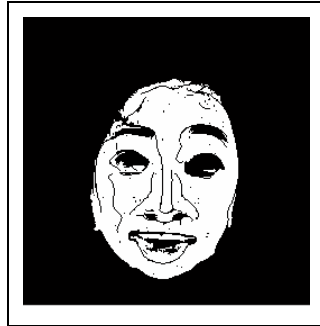
Figure 2.7 Background Separation



Now that we have an image where we have painted the background black and the edges more clearly delineated, we can go ahead and use the seeded region growing algorithm over the face region to locate the face in the image. For the region growing algorithm at this stage we select five seed pixels such they form a cross about the center of the image. These seeded regions are grown to get the final face region. However, in case the seeded regions grow out-of-bounds of the image then the algorithm is rerun after lowering the threshold and reinitializing the seeds, either until a region completely within the bounds of the image is found or the seeds do not grow at all. If such a region is found then it is considered to be the face. If the seeds do not grow or

after a number of repeated iterations still grow beyond the image then it returns an error declaring that face could not be detected.

Figure 2.8 Face Region



Feature Point Location

Once we have the face region located, we divide it into sub-regions. We then use various search mechanisms to locate the feature points. We primarily divide the face region into four quadrants, and locate the one most easily distinguishable point in each of the four quadrants i.e. two eyes and the two corners of the mouth. These key feature points are extracted using the particle swarm optimization technique, explained below. After we locate these key feature points we locate the rest of the points using the key feature as the reference. These features are extracted using the properties of the respective features, i.e. location with respect to the key features and intensity gradient

Particle Swarm Optimization

Particle swarm optimization (PSO) algorithm was developed in 1995 by Kennedy and Eberhart, for solving optimization problems of continuous non-linear functions. It is mainly been inspired by the artificial life and social psychology of flocking birds or schooling fish [33].

In this method the population members called “particles” are flown through the solution space. Upon initialization the particles are randomly located in the solution space and assigned random velocities. At each iteration the particle's velocities is so adjusted that it has a weighted acceleration, towards its previous best position and towards the global best [34]. The best position is decided by a function called the ‘cost function’ or a ‘fitness function’; this could be the function of the solution space if looking for a maxima or a minima, or some other function

that defines the characteristics of the optimal solution (in this case the characteristics of the feature). For example for the eye the cost function is defined so as to look for a dark region (eyeball), with lighter regions on the left and right (cornea) and another darker region (eyebrow) above the pixel being investigated. We define similar cost functions for each of the key features. These iterations are then continued until all the particles converge upon the optimal solution. The algorithm can be better understood using the pseudo code below [34].

```

Loop
  For i=1 to number of particles
    If  $G(\bar{x}_i) > G(\bar{p}_i)$  then do
      For d = 1 to dimensions
         $p_{id} = x_{id}$ 
      Next d
    End do

    g = i
    For j = indexes of neighbors
      if  $G(\bar{p}_j) > G(\bar{p}_g)$  then g = j
    Next j

    For d = 1 to number of dimensions
       $v_{id}(t) = v_{id}(t-1) + r_1(p_{id} - x_{id}(t-1)) + r_2(p_{gd} - x_{id}(t-1))$ 
       $v_{id} \in (-v_{max}, +v_{max})$ 
       $x_{id}(t) = x_{id}(t-1) + v_{id}(t)$ 
    Next d
  Next i
Until criterion

```

Cost Functions

Cost function also known as fitness function, is what defines an optimal solution. It could be any continuous function, designed such that it reaches a maximum at the optimal solution. The cost functions that we used for various features are described below.

Eye: The cost function of the eye is a sum of four terms.

$$G(x, y) = A + B + C + D$$

A = Sum of Variance across rows + Sum of Variance across columns

B = Sum of intensities above and below the particle minus two times the intensity at the particle; if the average intensity below the particle is more than that on the particle

OR

$$= -800$$

C = 255 minus mean intensity about the particle

D = -800; if the particle is beyond the region boundaries for the eye

OR

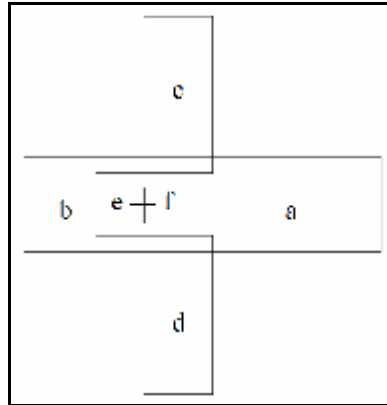
$$= 0$$

Eyebrow: The cost function for the eyebrow is 255 minus the average intensity of the image around the particle. It is the PSO algorithm that makes sure that the particle does not go out of the region of interest, which in this case was in the region above the eyes. A pixel with an intensity of 0 represents a black pixel and one with 255 represents a white pixel. When we define the function above we are looking at the dark region above the eye, which is the eyebrow. We need to look at the average of the region because otherwise we could make a mistake by picking a stray black pixel, and an eyebrow is surely thicker than a single pixel.

Mouth: The cost function for the left mouth corner can be represented by the template shown in Figure 2.9. If the mean intensity of area (a) is more than the threshold and the mean intensity of area (b) is less than the threshold then the cost function is the difference in intensity between these two areas, i.e. a darker right side and a brighter left side. There is also a bonus if the difference in intensity between areas (c) and (d) and (e) and (f) is greater than 50. This works on the understanding that the lips are darker than the rest of the cheeks, and for the left corner the lips are to its right. The lips also get narrow as they

move towards the corner and hence the bonus. A mirror image of this template can be used for the right corner of the mouth.

Figure 2.9 Left Mouth Corner Template

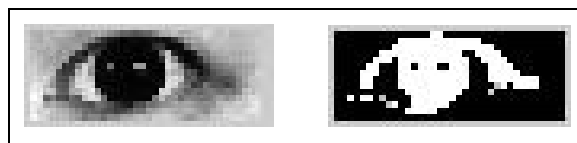


Nose: The cost function for the nose is a sum of two terms. The first term compares the intensity at the particle to those on the lower left and right of the particle; this is based on the concept that the tip of the nose is always brighter than the nostrils. While the second term ensures that the nose is close to the center of the face region, therefore a point closer to the center of the face would have higher fitness compared to that of a particle away from the center.

Other Techniques used

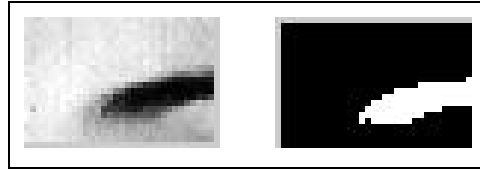
The eye corners are located next after the eyes are located using the PSO. The eye corners are located using a threshold to convert the region around the eye into a monochrome image (Figure 2.10) and then looking for the largest contiguous block of white pixels. This is based on the observation that in most cases within a small region around the eyes the eyelashes are the only continuous dark object and they begin and end at either corner of the eye. Further the threshold for converting to monochrome is calculated based on the mean intensity of the image around the eye. The first and last columns of this cropped image are then scanned to find the corners of the eye.

Figure 2.10 Eye Corner Estimation



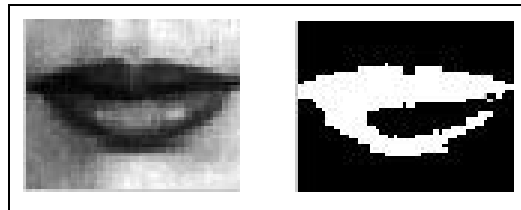
The eyebrow corners are located using a technique similar to that used for locating the eye corners, except that the search is performed in the region around the eyebrow. This is because the eyebrow is also a continuous feature and can easily be separated using a threshold (Figure 2.11).

Figure 2.11 Eyebrow Corner Estimation



A very similar technique is used to locate the upper and lower lips. The threshold is decided based on the whole region around the mouth while the actual search is performed only midway between the two lip corners. This is because in the mouth is symmetric and the relative position of the lips is always in between the mouth corners. For example in Figure 2.12 the topmost and the bottommost white pixel in the center column will be picked as the upper and lower lips respectively.

Figure 2.12 Upper and Lower Lip Estimation



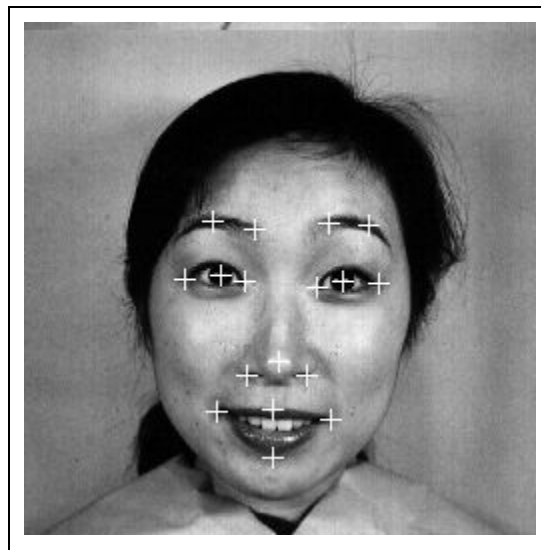
Results of Feature Point Extraction

Using the methods described above we locate seventeen feature points in all, as follows: eyes, eye corners, eyebrows, eyebrow corner, mouth corners, upper and lower lip, nose and nostrils. While extracting these features, we relied upon thresholds. These were initially some arbitrary numbers we identified by trial and error, but upon further trials we came to realize that the thresholds were not universal, and had to be customized to the lighting conditions, color of the eyes, skin color and hair color. Therefore we devised a graphical user interface which gives the user some flexibility with respect to the thresholds while not straying too far away from the

optimal values we found. All of these thresholds are based upon the mean intensity of the image around the desired feature.

We see that a location of each feature depends upon the previous. The eyes are the first feature we locate, but the accuracy of location of eye depends upon the identification of the face in the image. If we get this wrong, we would deploy the particles for the PSO in the wrong region and see a cascade effect of these errors at all points. Hence the location of the face is as important as location of the feature points. Just in case the face region separation algorithm does not work as expected, the GUI also has a manual face region selection option. This saves time on occasion when we are dealing with a small number of images with different lighting conditions.

Figure 2.13 Feature Points



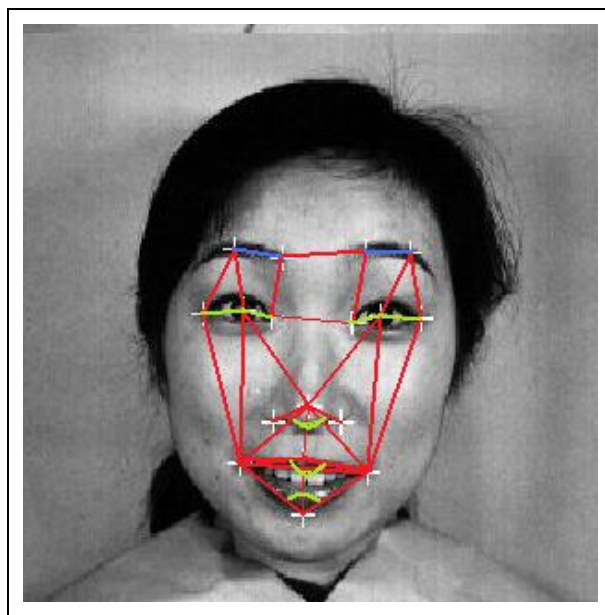
Construction of Vector

At this point we have located the seventeen feature points on the image (Figure 2.13). We need to figure out a way to input this vector into an artificial neural network for classification. Since a backpropagation neural network only accepts vectors as inputs and not a series of vectors we had to find a way to convert the coordinates of the feature points into a vector. In our initial attempts we tried forming a vector by simply putting the entire x and y coordinates one after another to form a column vector. However, this had a number of drawbacks, and was not suitable for classification of emotion.

Since the coordinates are just a set of numbers representing the location of points on the image, they would change with the location of the face in the image. This could be solved by using one of the feature points as the reference. However we then face a problem of scaling, since the distance of each point from the other would change not just based on the expression as we would like it to, but also on the size of the face in the image. For example the features would be closer to each other if the picture is taken from a distance and further apart if it is a close-up image. To offset the effects of zoom in the image we would have to normalize the feature points extracted. We solve this by scaling the coordinates of the feature points such that the distance between the two eyes is equal to one, this is similar to what other researchers [4] [26] [13] have used for normalization. By doing so we would scale each image to the same size while still allowing for the movement of features on the face.

The elements of the vectors were still numbers representing points on the image, but did not represent the physical relation between various feature points within a given feature or among features. Hence this would not classify the expression satisfactorily. We then began testing vectors that consisted of distances between various features, angles subtended by them, polar representations, difference vectors and various combinations of these. We eventually built a vector that consisted of a combination of distances, angles and difference vectors. The mask formed by the final vector is shown in Figure 2.14.

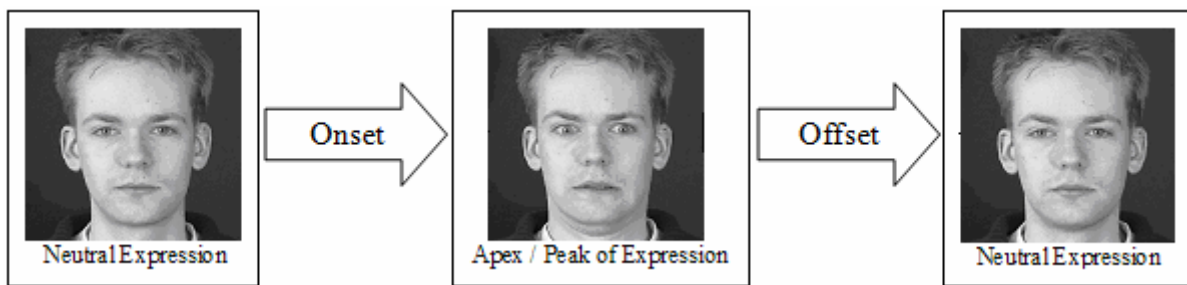
Figure 2.14 Feature Mask



FEATURE POINT TRACKING IN VIDEOS

Facial expressions when seen over time evolve from the onset of expression to the apex, when the expression is at its peak, and then the offset (Figure 2.15). One phase instance of an expression is the complete cycle from onset to offset of expression, in this method we classify the whole phase as an expression. This method has of classification has not been used because it is difficult to say when exactly has the expression has begun or ended, and to be able to locate the beginning and end of an expression we would need to classify each frame to look for deviation from a neutral face. Classifying each frame in a video for facial expression is called the frame instance type of classification. The frame instance type classification when plotted temporally can show the onset and offset of the expression [30].

Figure 2.15 Evolution of Facial Expressions Over Time



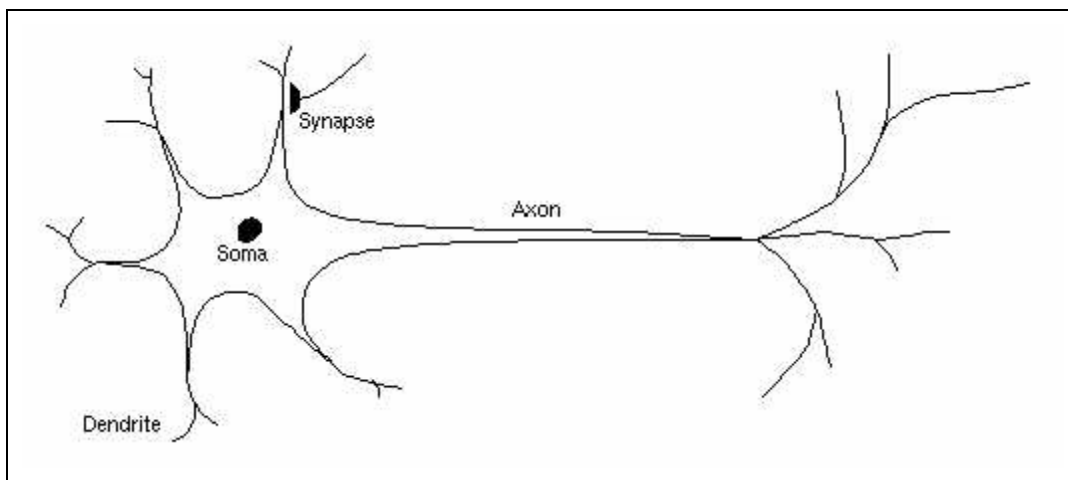
We use the frame instance wherein at each frame we recalculate the feature points and the vector for each frame. In this stage while computing the feature points for each frame, we no longer locate the face region again for each frame. As a video camera normally captures video at 30 frames per sec the changes from one frame to the next in an unedited video are usually not significant. Hence we deploy the PSO for the feature points within a small region surrounding the corresponding feature point in the previous frame. We construct the vector for input to the artificial neural network by taking the difference of the vectors for the image under consideration and a neutral frame of the sequence. By default the first frame of the sequence is considered the neutral frame, because most of the databases consist of clips with the expression beginning with a neutral face, going to apex and back. But the GUI does have a provision to select any other frame as a neutral frame in case the clip began with a different expression.

NEURAL NETWORKS FOR CLASSIFICATION

Machine learning methods are algorithms that allow computers to learn and extract rules and patterns from large data sets. These techniques are usually employed in areas where it is difficult to define a set of rules but large amount of data along with its correlated expected results are available. For example in areas like robot locomotion, speech recognition, object recognition, data mining and stock market analysis to name a few. Our problem of facial expression recognition has similar properties, where it is difficult to have a set of rules for expression recognition and we have a database of labeled images of facial expressions. Hence we use one such technique known as artificial neural networks for classification of facial expressions.

Initial interest in the area of artificial neural networks was sparked in 1943 after McCulloch and Pitts introduced the concept of simplified neurons [35]. A neuron in its biological sense is the basic unit of the brain. It typically consists of four parts, namely: dendrites, the synapse, the cell body/soma and the axons. Dendrites act as the antennae for the neuron, receiving signals from the surrounding neurons and passing the signal to the cell body (Figure 2.16). Once the sum of the signals received at the cell body crosses a threshold, the cell fires a signal to its surrounding neurons through its axons [36].

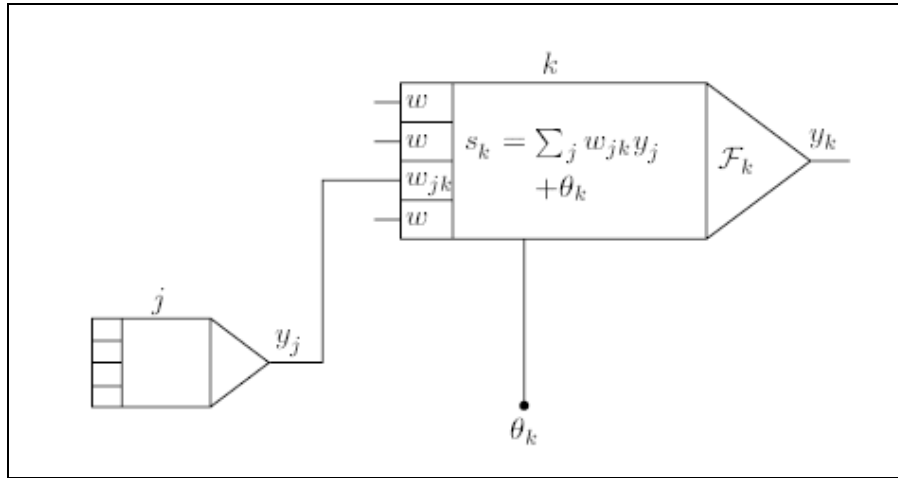
Figure 2.16 Biological Neuron



The primary unit of an artificial neural network is called a 'neuron' similar to its biological counterpart. These function similar to the biological neuron, by receiving inputs from

the neighboring neurons or external inputs, for input layer; and passing it onto the next layer or the output.

Figure 2.17 Basic Component of Artificial Neural Network

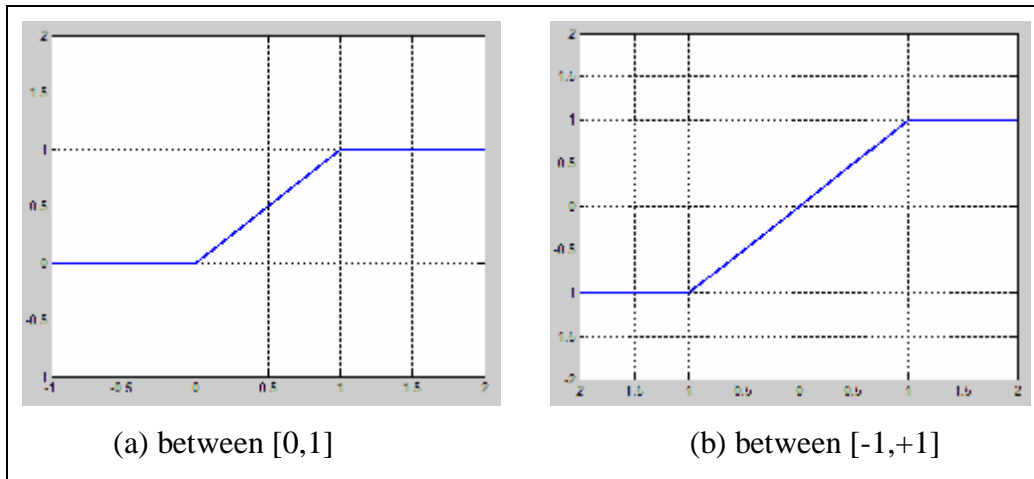


Some of the defining aspects of a neuron are: the weights (w) for each of its inputs (j), its bias (θ), and its output transfer function (F) or the activation function as seen in Figure 2.17. The weights define the importance of each of the input to that neuron. The bias also known as offset acts as the threshold for the neuron. The activation function defines the output characteristics of the neuron, they are usually non-linear. Some common activation functions are explained below.

- a. Identity Function: This is also known as the linear transfer function. It passes the inputs as it is to the next layer. At times it is also programmed to saturate the output of the neuron to $[0, 1]$ (Figure 2.18a) or $[-1, 1]$ (Figure 2.18b).

$$F(x) = \begin{cases} 0 & x \leq 0 \\ x & 0 < x < 1 \\ 1 & x \geq 1 \end{cases} \quad \text{OR} \quad F(x) = \begin{cases} -1 & x \leq -1 \\ x & -1 < x < 1 \\ 1 & x \geq 1 \end{cases}$$

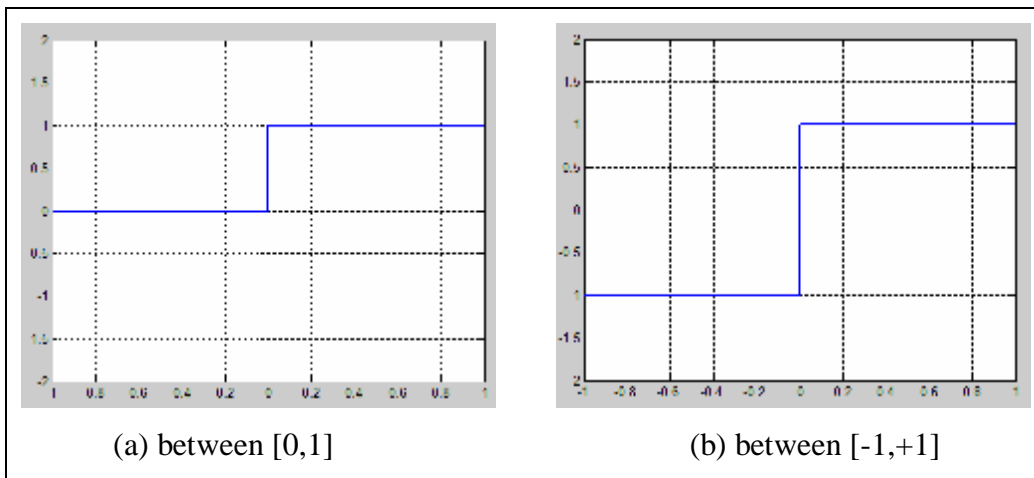
Figure 2.18 Identity Function



- b. Step Function: This function has outputs of either 0 or 1 (Figure 2.19a), or either -1 or +1 (Figure 2.19b).

$$F(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases} \text{ OR } F(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

Figure 2.19 Step Function

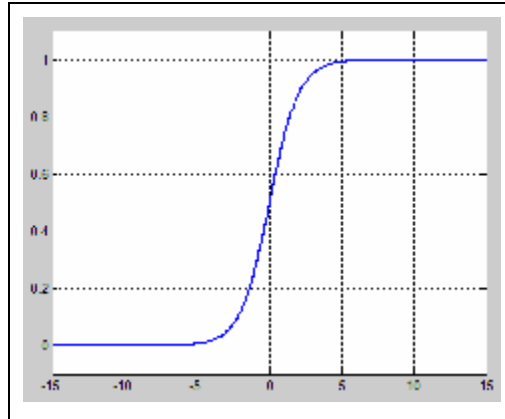


- c. Sigmoid Function: This is the most preferred transfer function because of its smooth and bounded nature (Figure 2.20). It also has a simple first derivative. This gave the neuron an advantage of scaling the output from the neuron to anything between 0 and 1 in a continuous manner.

$$F(x) = \frac{1}{1 + e^{-ax}}$$

where a is a constant.

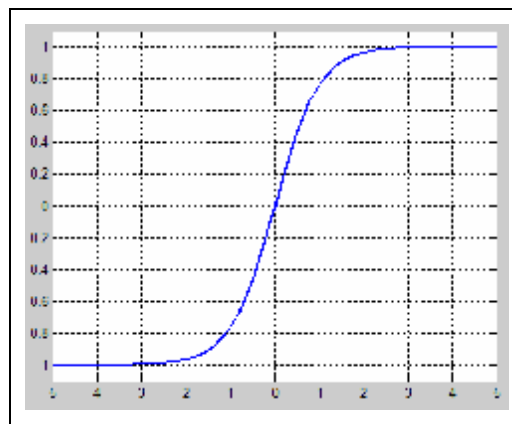
Figure 2.20 Sigmoid Function ($\alpha = 1$)



d. Hyper-tangent Function: This is similar to the sigmoid function (Figure 2.21).

$$F(x) = \frac{1 - e^{-ax}}{1 + e^{ax}}$$

Figure 2.21 Hyper-Tangent Function ($\alpha = 1$)



These weights and the biases are initiated to some small random number, and then modified using learning rules so as to learn the trends present in the training data.

An artificial neural network is then formed by placing a number of such neurons in parallel to form a layer, and then having a number of such layers connected either in series or

parallel or both. A neural network is defined by the number of neurons in each layer, the number of layers, their type of connection between layers and the output transfer function used by the neurons in each layer. The networks can be classified into two categories based on the how the neurons are interconnected, namely recurrent and non-recurrent networks. Recurrent networks (Figure 2.22) have connections from their outputs to their inputs and are best used for time varying data or prediction purposes, where data information from previous instances is useful. In non-recurrent networks there is no feedback from the output of the network to its inputs, example a feed-forward network (Figure 2.23). These kinds of networks are more appropriate for applications where each set of inputs has a different solution and is not linked to the results obtained in the previous iteration, for example object recognition or character recognition. We used a feed-forward type of neural network for our purposes in this research, since the JAFFE database consisted of only snapshots and not continuous video with temporal information.

Figure 2.22 Recurrent Network

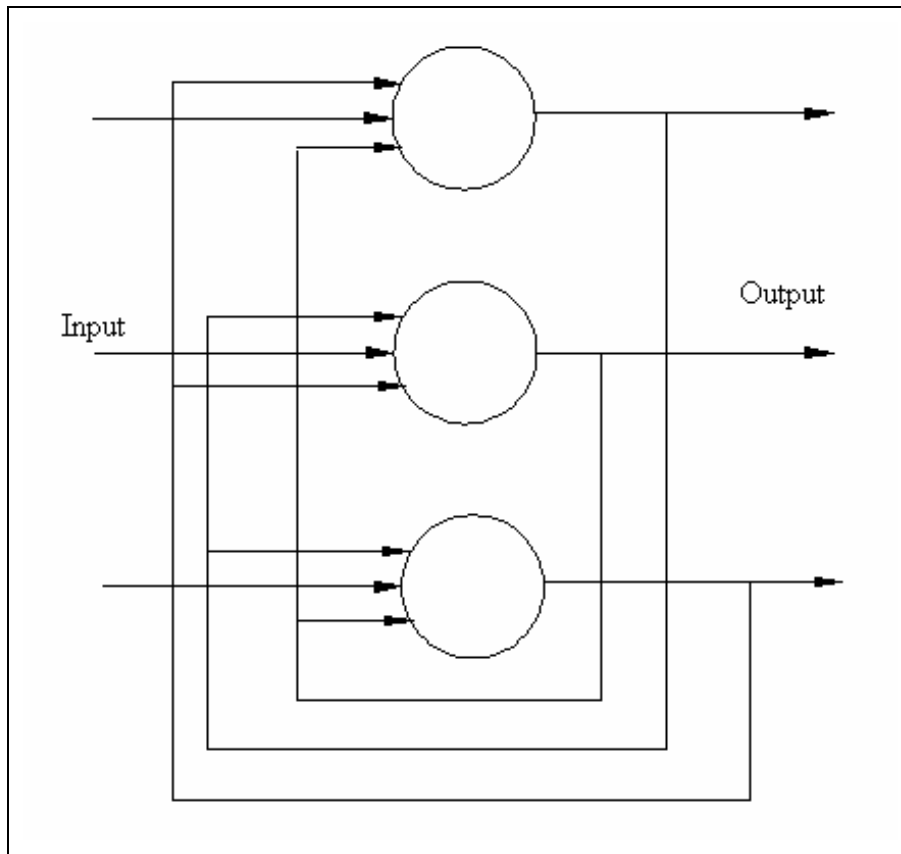
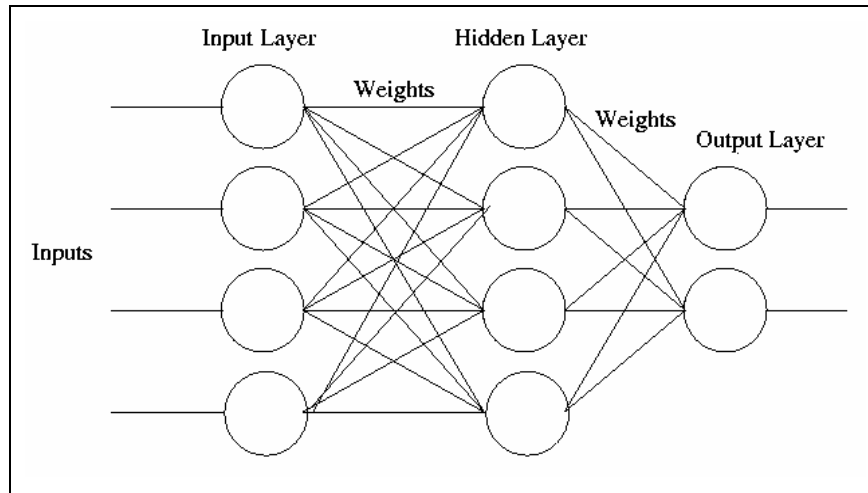


Figure 2.23 Three Layered Feed-Forward Network [63]



Training of ANN

We use the backpropagation training for the neural network. This is a widely and most commonly used method for training of ANN. It was first introduced by Paul Werbos in his PhD dissertation at Harvard in 1974, as “dynamic feedback”. Backpropagation effectively is an efficient method to calculate derivatives of large and complex systems represented by smaller subsystems that are defined by differentiable functions. Hence its use is not limited to ANN, but to a number of other fields such as pattern recognition, dynamic modeling, control of systems over time, fluid dynamic modeling, etc... [69]. Since backpropagation can only be used on systems defined by differentiable function, we cannot use step function as the transfer function for the neuron. But a sigmoid function approaches a step function as $\alpha \rightarrow \infty$, and can be used instead.

Backpropagation actually stands for ‘backpropagation of errors’ and is exactly what is done during the training of the ANN. The mean squared error is computed at each iteration and then the weights of the network adjusted such that the error is minimized. Since the only variables in the network are the weights that can be changed to make the error low, and all the nodes of the network have a continuous and differentiable activation function, we can say that the error is essentially a continuous and differentiable function of weights, $E = F(w_1, w_2, \dots, w_l)$, and training a problem of minimization.

To begin with training we would need a training set consisting of p ordered pairs of n and m dimensional vectors $\{(x_1, t1), (x_2, t2), \dots, (x_p, tp)\}$, where n is the dimension of the input vector

and m the dimension of the output vector. Here x_i would be a sample input pattern and t_i the corresponding output, also known as target. The error formula for the network would then be

$$E = \frac{1}{2} \sum_{i=1}^p (o_i - t_i)^2$$

where, o_i is the actual output of the network as opposed to t_i which is the desired output.

We can then minimize error, E , using gradient descent to compute the new weights. Gradient descent is an optimization algorithm, used to find the local minima of the function by moving in the direction of the negative slope i.e. towards the local downhill [68]. For this we have to calculate the gradient given by,

$$\nabla E = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \dots, \frac{\partial E}{\partial w_l} \right)$$

Weights are then updated using the increment

$$\Delta w_i = -g \frac{\partial E}{\partial w_i}$$

$$w_i = w_i + \Delta w_i = w_i - g \frac{\partial E}{\partial w_i} \quad \text{for } i=1, 2, \dots, l$$

where, γ is the learning rate [55].

Since the backpropagation algorithm is so famous, a number of variants of it have been developed. One of the most famous and commonly used variant is ‘‘Gradient Descent with Momentum’’, a method that we have used in training our ANN. This method has the advantage of not getting stuck at local minima early on in the training by making the weight increment in the N^{th} iteration dependent on the $(N-1)^{\text{th}}$ iteration.

$$\Delta w_i(N) = -g \frac{\partial E}{\partial w_i} - \eta \Delta w_i(N-1)$$

where, η is a constant known as the momentum [46]. This is similar to letting a roller coaster go free after taking it to the top, the momentum of the train takes it through the up and downs in the ride, finally bringing it to rest at the lowest point. In a similar way the using the method of gradient descent with momentum, helps the errors move through the local minima in search of the global minima. As it approaches the global minima, the error might oscillate a little before finally coming to rest at the global minima.

Testing of ANN





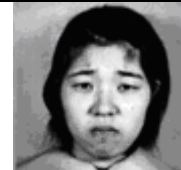

Each database was divided into two main sections, the training and the testing group. The groups we so divided that the training set was bigger than the testing, while the testing set had diversity in emotions and subjects. This was usually accomplished by leaving the subject with the most diverse range of images out of the training set. That way the ANN had a whole array of expression on a number of different subjects to train on. But then we would also have a face that the network has never seen before and also a range of expressions, that way we can test performance on both unseen faces and also on each expression. For example in the JAFFE database we used nine subjects for training and one for testing, similarly in the MMI database subject 39 was used for testing while the other seventeen subjects were used for training. We also tested the ANN on different databases, so in that case the one face that formed a testing set for the original database was used as a baseline to compare performance of the ANN on other databases.

RESULTS of FER

Training on JAFFE and Testing on JAFFE

We first present our results of training the ANN on the JAFFE database and testing it on the JAFFE database. We present these results first, because we anticipate that testing the ANN on non-JAFFE images would potentially yield either misclassification of facial expressions or a strong indication of the goodness of a minimal feature point set that we used per facial expression. In fact, this baseline should yield expectedly *good* results. The following results were achieved by using ANN with one hidden layer containing 30 neurons. We trained 10 different networks and then averaged the results.

Table 2.1 True Positive rate for ANN trained and tested on JAFFE database

TP	Angry	Disgust	Fear	Happy	Sad	Surprise
58.43%	57.33%	42.00%	50.89%	74.71%	44.22%	90.00%
						

As shown in Table 2.1, the average, overall accuracy of the trained ANN was 58.43%, but ranges from a low of 42% for “disgust” to as high of 90% for “surprise”. This average figure of merit (FOM) may seem low when compared to higher figures reported by other studies ranging from 65% to 95% [51]. However, we emphasize the limited number of feature points extracted per facial expression in contrast to these studies. In fact, other studies when tested with images beyond its training database show a comparable true positive rate of 60% [39]. Thus, in the task of classifying six different expressions, we conditionally accepted this figure of merit. This figure is consistent with the drop in accuracy of facial expression recognition by human subjects [5]. Also Table 2.1 shows accuracies for each expression. From the results above we can clearly see that our methodology is suited for classifying facial expressions with clear movement of feature points, i.e. “happy” and “surprise” relative to the neutral FE. We assert that these expressions are well-suited for the ANN to learn and classify. On the other hand, facial expressions such as “angry, disgust, fear and sad” are “weakly” characterized by our descriptor and therefore not as well-suited for the ANN to learn. In fact the occurrence of weakly characterized expression suggests a need for a second biometric and/or other means of secondary validation.

Training on JAFFE and Testing on MMI

It is impractical to assume that we will always have associated training dataset available for a FE which we want to analyze. Therefore to evaluate the ability of our ANN-based application to recognize facial expressions on unseen (external) faces, we first trained the ANN with the JAFFE database and then test it on images taken from the MMI database. In fact, not only are the training and test sets different, they also span ethnic and cultural characteristics. Thus, again using the same ANN, we averaged the results of 10 networks. Validation based early stopping was used while training the ANN. Under normal circumstances training of an ANN continues until either a preset number of iterations/epochs are completed, or the error on the training set attains a level reaches below a threshold. While using a validation based early stopping, a portion of the complete database is assigned to the validation set. After every epoch the ANN is tested using the validation set and its error computed. Any increase in error in the validation set stops further training. In this case we used a sample of the MMI database as the

validation set so as to improve performance. This also prevents the ANN from overfitting, which is when the ANN begins to memorize the training set rather than identifying a pattern.

Table 2.2 True Positive rate for ANN trained on JAFFE and tested on MMI database

TP	Angry	Disgust	Fear	Happy	Sad	Surprise
55.96%	57.38%	17.22%	13.05%	90.80%	50.24%	92.00%

Table 2.2 shows that the average accuracy achieved on a different database was 55.96%, which is comparable to the accuracy we achieved using the JAFFE database. This is also comparable to the methods developed by Littlewort et al. who reported 93% recognition of facial expression on their training; however, this FOM dropped to ~60% when tested on a different database [39].

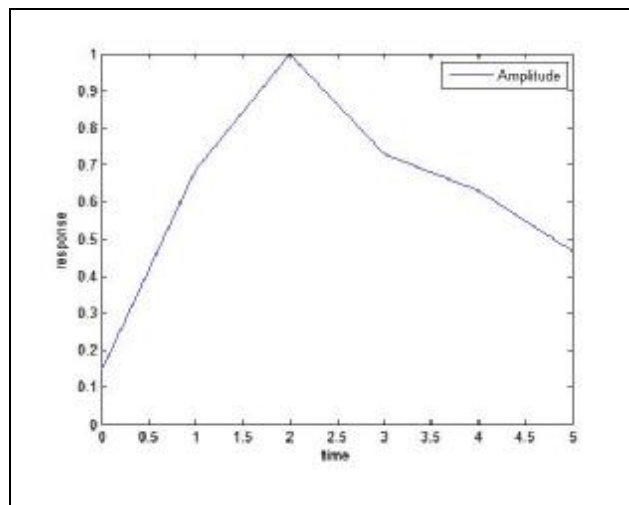
One can again see that “surprise” and “happy” facial expressions consistently produce higher figures of merit and support the view that these facial expressions are easiest to train on and classify. Interestingly, the ANN is some 15% better at recognizing the happy expression in the MMI than in JAFFE on which it was trained. Though limited, we learned through our own facial expressions database development that subjects from some ethnicities and cultures (here Asians) do not express a sense of “happy” with the same “intensity” as from other backgrounds. However, the vectoral descriptor appears to be consistent for both JAFFE and MMI. We can also see that “angry” and “sad” facial expressions are still relatively difficult for the ANN to classify; that is, the descriptor for these MMI expressions is not particularly distinct. Interestingly the true positive rates for “disgust” and “fear” exhibited a significant decrease relative to Table 2.1. In fact, in the case of “fear”, the rate is worse than that achieved by random guess. So overall the performance of the ANN trained on the JAFFE and tested on the MMI database appears to be consistent for ‘angry’, ‘happy’, ‘sad’, and ‘surprise’ expressions and (provisionally) suspect for ‘disgust’ and ‘fear’, relative to Table 2.1.

The phase instance classification rate of video sequences can be improved using the method to be described. If we define the amplitude of an FE as the averaged displacement of all (extracted) feature points on a FE relative to their positions on a reference neutral FE, we observe that this amplitude increases as the FE appears on the face, reaches a relative maximum as the FE is held in time and subsequently, decreases as the FE dissipates or transitions to

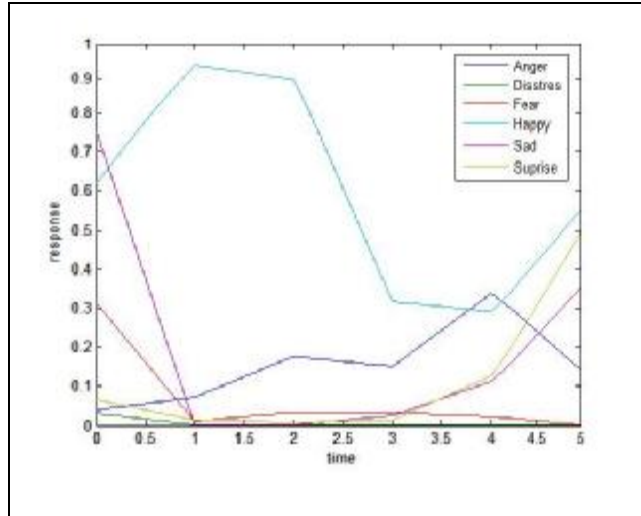
another FE. Logically when the amplitude is small at the onset and end of the sequence, the calculated vectoral descriptor is not definitive and as expected, presents itself as a weak (and difficult) test example for the ANN. However, the amplitude increases the calculated descriptor and becomes more definitive (thus characterizing the particular FE). Here, those examples near the peak amplitude associated with a FE are more likely to be correctly classified.

Using this characterization of the spatiotemporal facial dynamics we plot the response of our ANN for all six expressions over the video sequence and in addition, normalize the response for each frame relative to the maximum amplitude. This normalization not only elucidated the peak amplitude but importantly relative to the six normalized FEs. Thus in Figure 2.24, we see the change in amplitude over time for a given FE. Starting from a neutral FE, a peak develops and then subsides for some FEs (Figure. 2.24a, “happy”) while large amplitudes are attained only at the start and end for other FEs (Figure. 2.24b). Here Figure. 2.24a depicts the raw response whereas Figure 2.24c shows the (maximum) normalized amplitude for a given FE. From tracking the FEs of MMI subjects, we first see similarities and differences among the six FEs. Note that there are FEs with larger amplitude at the start/end of the video (“Su, Fe, Sa”). Surprise, though similar in trend (amplitude) to “sad”, has a vectoral descriptor that distinguishes it in training and testing across the JAFFE and MMI database. On the other hand, for “sad” and “fear”, the vectoral descriptor is less definitive and seemingly more specific to the training and testing database.

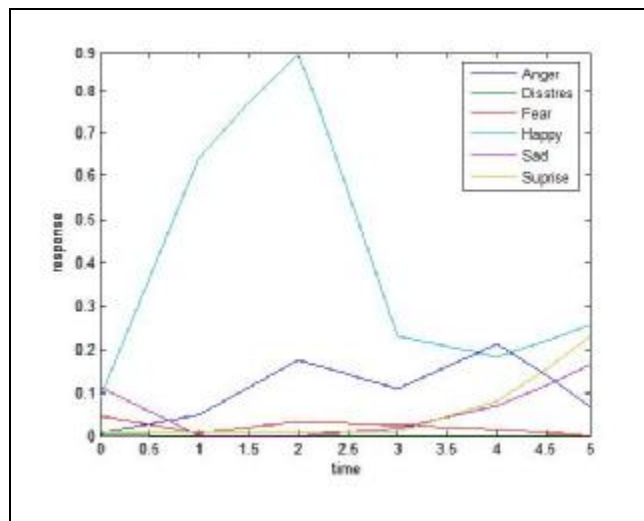
Figure 2.24 Normalizing Expression Recognition in Video Sequences



(a) Amplitude of expression



(b) Raw response of ANN



(c) Normalized response of ANN

Training on MMI and Testing on JAFFE

We then also tried training an ANN with the MMI database and testing it with the JAFFE database. The results for this combination of training and test set are shown in the table below.

Table 2.3 True Positive rate for ANN trained on MMI and tested on JAFFE database

TP	Angry	Disgust	Fear	Happy	Sad	Surprise
52.09%	45.11%	31.73%	39.27%	88.97%	51.67%	57.22%

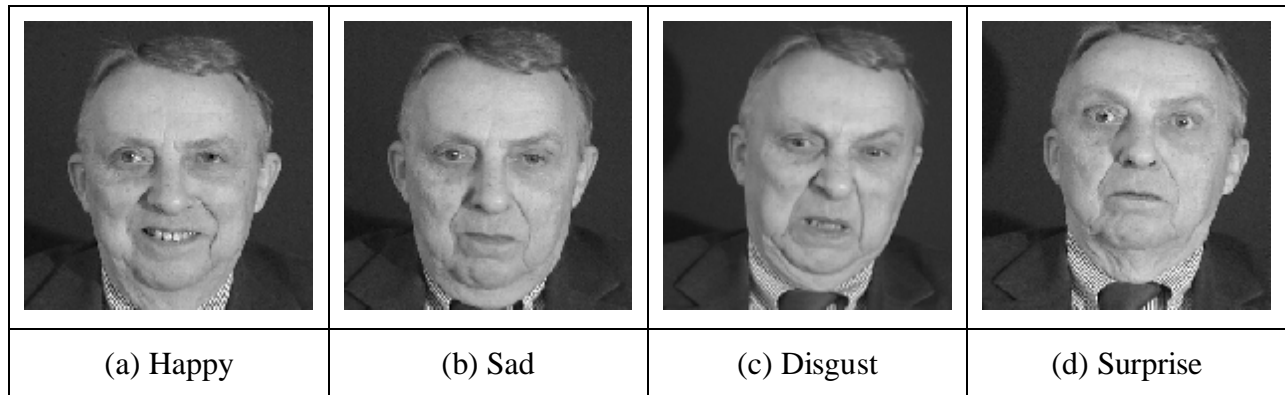
From Table above we see the overall recognition rate of training on one database and testing on another does not depend on what combination of training and testing databases we use.

But upon closer inspection we will see that training on the MMI database and testing on the JAFFE has significant differences as compared to the reverse. We observe the recognition rate for surprise has dropped drastically from 92.00% in the previous case to 57.22% at present. However at the same time we also observe that the two expressions of ‘disgust’ and ‘fear’ that saw a drastic drop in recognition in the previous case are recognized much better now. There is a change from 17.22% to 31.73% for disgust and 13.05% to 39.27% for fear. These results are comparable to those obtained when ANN was trained and tested on the JAFFE database. We will take a closer look at these discrepancies in the section to follow.

Testing on Elderly Faces

Some testing was performed on the images of the elderly gentleman (Figure 2.25) available in the MMI database. While trying to process these images we had trouble locating the eyebrows, because the eyebrows were essentially white and hence blended very well with the skin color of the gentleman. Hence we decided to go ahead with expression recognition without locating the eyebrows. But as expected this induced problems. Since the eyebrows were eliminated, so were any features that were calculated based on the eyebrows. This brought down the size of the input vector from 38 features to 27 features. To compensate for this we used Gabor wavelets with six different orientations in the regions around the eye and the mouth. A new neural network was generated to classify the images using this newly developed feature set.

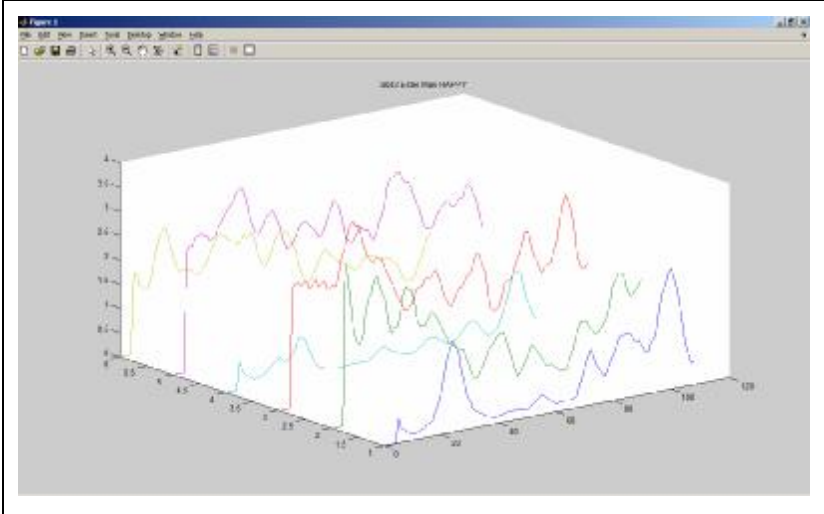
Figure 2.25 Examples of Images of the Elderly Gentleman



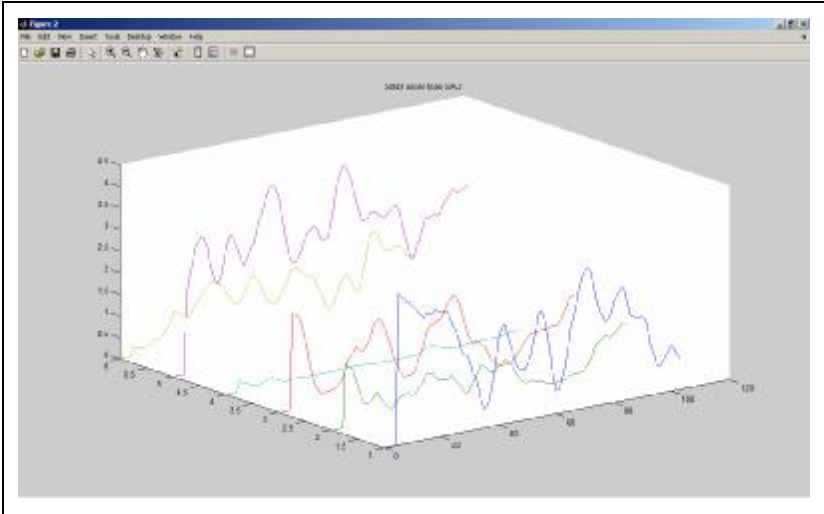
The database contained images of the elderly gentleman for only four emotions, namely happy, sad, disgust and surprise. We could only process happy, sad and disgust because surprise

consisted of large head movements; this caused problems in feature point location. Results from the other three expressions are presented in the Figure 2.26 below.

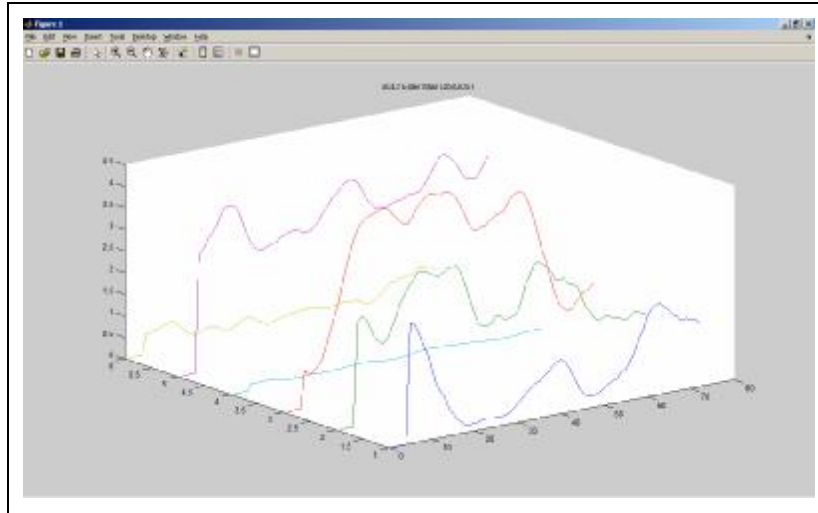
Figure 2.26 Expression Recognition in Elderly Faces



(a) Happy



(b) Sad



(c) Disgust

From the above plots we can see that in this method happy (light blue line, no 4) is hard to detect. The happy expression always has very low recognition, even when the face was actually supposed to be happy (Figure 2.26a). However the present approach shows promise with more difficult to detect expressions such as sad (purple line, no 5) and disgust (green line, no 2). In fact for the sad expression (Figure 2.26 b) we can see that it is recognized very well. Even disgust has a very strong signature compared to the previous networks we had developed.

This study on elderly gentleman was performed on behalf of the Center on Aging at Kansas State University. Any further discussion is limited to the original study. Although the study of elderly people is of interest, the small amount of data available prevents us from any significant study. However, our method is independent of age.

ANALYSIS of FER

Our descriptor works best with expressions in which there is clear and distinct movement of feature points with reference to their neutral position, i.e. happy and surprise, while the more subtle expressions seem to be difficult to classify. We know that our choice to locate fewer feature points prevents us from capturing the subtleties in face required to classify the rest of the expressions satisfactorily. This also introduces a chance of increased noise in the data feed to the ANN. But this is a compromise we were ready to make to make the system light on computation and also with the idea that once this system is deployed in the field the images are not going to

be very sharp, and these feature points are those that can be extracted even on low resolution images.

From the results we just saw, one thing that is apparent is that the happy and surprise expressions have been recognized most consistently, followed by anger and sad, whereas the recognition rate for fear and disgust has been poor. These results we see above for testing and training on the same JAFFE is not comparable to high recognition rates ranging from 65% to 98% published by other researchers [50], but if we look the recognition rates when some of these systems have been tested on a different from their training database these accuracies are in close comparison at 60% [39].

Upon closer inspection of the results we see that there are patterns in the misclassification. Here a confusion matrix is a great way of looking at the performance of the ANN. It also gives us an insight on the classification pattern. It is a matrix whose rows represent the true expression of the image/video, and its columns the actual results of classification. The values along the diagonal of the matrix represent the number of correct classifications.

Table 2.4 Confusion Matrix for ANN trained on JAFFE and tested on MMI database

True\Classif.	An	Di	Fe	Ha	Sa	Su
An	30	5	5	0	2	0
Di	26*	6	1	0	2	1
Fe	0	4	2	1	8	31*
Ha	0	1	2	44	2	1
Sa	3	6	3	0	23	6
Su	0	0	0	0	3	47

Table 2.5 Confusion Matrix for ANN trained on MMI and tested on JAFFE database

True/Classif.	An	Di	Fe	Ha	Sa	Su
An	51	29*	1	3	6	0
Di	19	38	4	1	25	0
Fe	19	14	40	7	14	2
Ha	4	0	1	82	0	0
Sa	24	8	13	4	39	2
Su	1	3	27*	3	1	55

Let us take a look at rows two and three of Table 2.4; these represent those images that were supposed to be disgust and fear. These were the two expressions that had the worst recognition rate when we had trained the ANN on JAFFE database and tested it on the MMI database. We see for disgust that a large portion of the images that were misclassified were marked as anger (*) (Figure 2.27), and similarly for fear it is surprise (*) (Figure 2.28). These are expression pairs that look very alike and we could argue that the descriptor is not capable of differentiating between these pairs of expressions. But in that case we should have an equal misclassification of ‘anger’ as ‘disgust’ and ‘surprise’ as ‘fear’, this we see is not the case.

Figure 2.27 Comparing Anger and Disgust Expressions

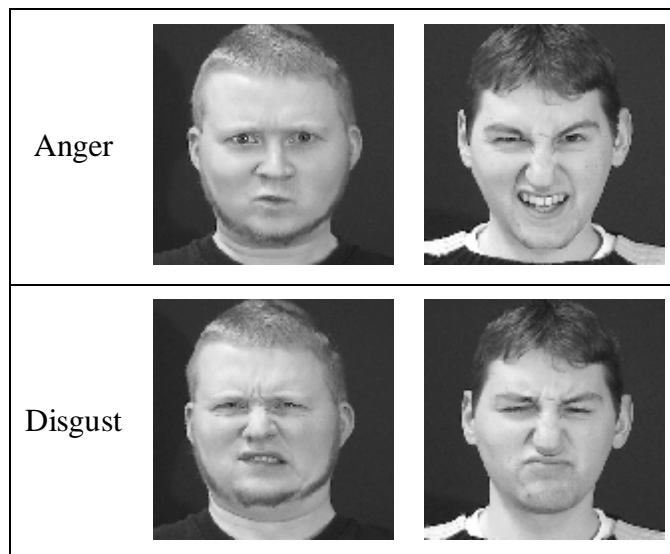


Figure 2.28 Comparing Surprise and Fear Expressions



Now we shift our focus to Table 2.5 which represents a system that is trained on MMI and tested on JAFFE. We see a similar pattern with misclassification but in the reverse order, i.e. a significant number of angry and surprise images are classified disgust and fear respectively. Even with this configuration we see that misclassification in the reverse order is not as significant. There is some misclassification of ‘disgust’ as ‘angry’ but a greater number is misclassified as ‘sad’. Similarly there is misclassification of ‘fear’ as ‘anger’, ‘disgust’ and ‘sad’ but not ‘surprise’. This leads us to the partial conclusion that the ANN is able to see a pattern even for disgust and fear, and suggests that it is not a problem with the descriptor or the training of the ANN, but rather a more fundamental difference between the two databases. The basic difference between the two databases is the cultural and ethnicity of the subjects. This leads us to at least a partial conclusion that there is indeed some difference in how people from different backgrounds express themselves. The confusion between the ‘fear’ and ‘surprise’ among groups from different cultural groups was also shown by Ekman in his experiment with an isolated tribe in Papua New Guinea, and later by Heider when he repeated a similar experiment in Indonesia [21]. This is not only corroborates with previous results [70] [44] and consistent with human experience, but perhaps for the first time, this has been revealed by digital means. So although some facial expressions, such as ‘happy’ and ‘surprise’, are understood all over the world (and across cultures and ethnicities), some others may depend on cultural and ethnic factors.

CHAPTER 3 - EMOTION RECOGNITION IN SPEECH

Speech is the primary form of communication used by humans. Along with the linguistic content of speech, the way a word is said is equally important. The tone of the speech contains cues to the emotional state of the person speaking, and we as humans naturally recognize these emotions. When translating using speech recognition systems, emotions are only noise that degrades the performance of the speech recognition systems. This prevents us from using computers to transcribe emotional speech in conversation [57]. Hence if the emotions in speech could be recognized and subtracted from the speech it could improve speech recognition systems. Usually emotions are consciously expressed to complement and emphasize the linguistic content of speech. At other times the emotions contradict the meaning of the words used, usually when trying to conceal ones feelings [47].

Each emotion has particular way in which it affects speech and it is these changing properties that we use as cues to emotion present in the speech [3].

- Anger: Is characterized by higher pitch, pitch range, mean energy and increased rate of articulation.
- Fear: Is similar to angry, with higher pitch, pitch range and high frequency energy levels, and also quicker speech.
- Sad: Is characterized by lower pitch, pitch range and mean energy and lower articulation rate.
- Happy: has higher mean pitch, mean energy and pitch variability.

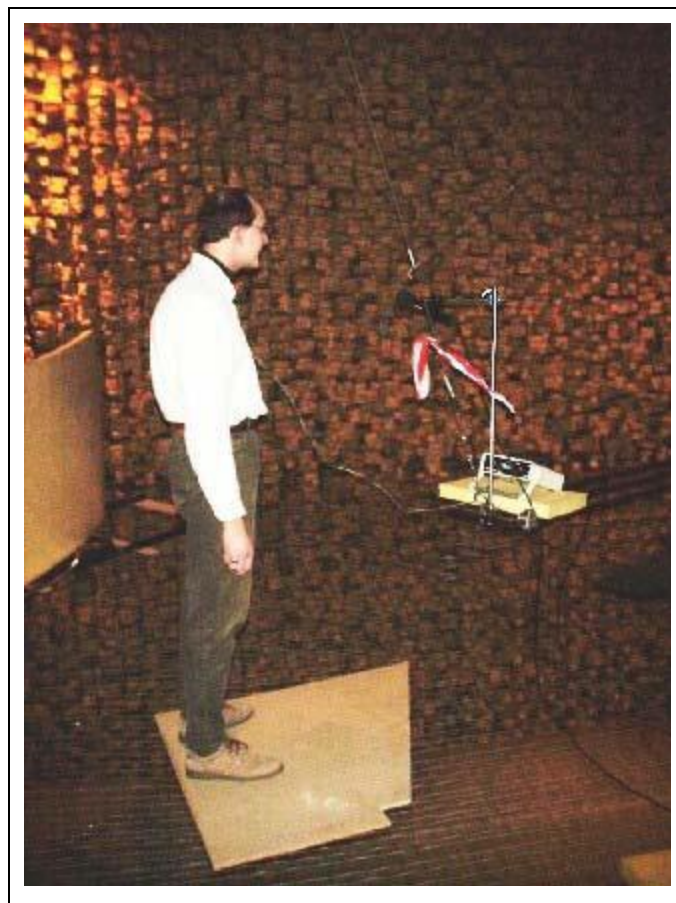
Speech parameters used commonly for classification of emotions are pitch, formants, mel frequency cepstral coefficients and energy levels.

SPEECH DATABASE

The database we used to develop our system for emotion recognition in speech is in German language, it was developed by Burkhardt. F., et al, at the Technical University at Berlin [7]. The database contains sentences portraying emotions of neutral, anger, fear, joy, sadness, disgust and boredom. These sentences are spoken by 10 subjects, 5 male and 5 female, selected

from a larger group of 40 people. Each one of the 40 members was asked to record one sentence per emotion. These recordings were then evaluated by a group of experts for naturalness and reconcilability of the emotion, and the final group of 10 was selected to record all the possible combinations of sentences and emotions. The recordings took place in an anechoic chamber at the Technical University at Berlin. The samples were originally recorded at 48 kHz. The final database available contains samples at 16 kHz, these were obtained by re-sampling the 48kHz signal.

Figure 3.1 Photograph during recording in anechoic chamber at TU-Berlin. [7]



The database consists of 10 sentences in all. These sentences were chosen such that they could be used in every-day conversation and also contained as many vowels as possible. During the recording sessions the actors were asked to visualize past situations in which they had felt such emotions. The sentences used are in the Table 3.1 below. These recordings were then presented to 20 subjects who had to recognize the emotional state of the speaker and the

naturalness in of the emotion. The final database then consisted of the only those sentences whose recognition rate was better than 80% and naturalness better than 60%. This finally produced a database of 500 sentences out of the 800 sentences that were recorded.

Figure 3.2 Recognition rate for various emotions in database [7]

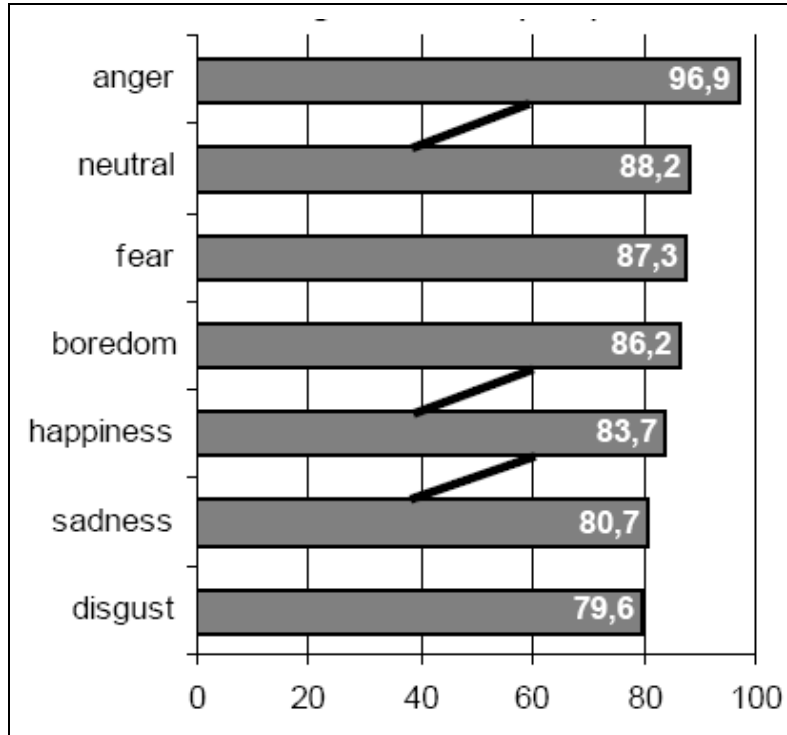


Table 3.1 Sentences in speech database [7]

Code	Text (German)	English Translation
a01	Der Lappen liegt auf dem Eisschrank.	The tablecloth is lying on the fridge.
a02	Das will sie am Mittwoch abgeben.	She will hand it in on Wednesday.
a04	Heute abend könnte ich es ihm sagen.	Tonight I could tell him.
a05	Das schwarze Stück Papier befindet sich da oben neben dem Holzstück.	The black sheet of paper is located up there besides the piece of timber.
a07	In sieben Stunden wird es soweit sein.	In seven hours it will be.
b01	Was sind denn das für Tüten, die da unter dem Tisch stehen?	What about the bags standing there under the table?

b02	Sie haben es gerade hochgetragen und jetzt gehen sie wieder runter.	They just carried it upstairs and now they are going down again.
b03	An den Wochenenden bin ich jetzt immer nach Hause gefahren und habe Agnes besucht.	Currently at the weekends I always went home and saw Agnes.
b09	Ich will das eben wegbringen und dann mit Karl was trinken gehen.	I will just discard this and then go for a drink with Karl.
b10	Die wird auf dem Platz sein, wo wir sie immer hinlegen.	It will be in the place where we always store it.

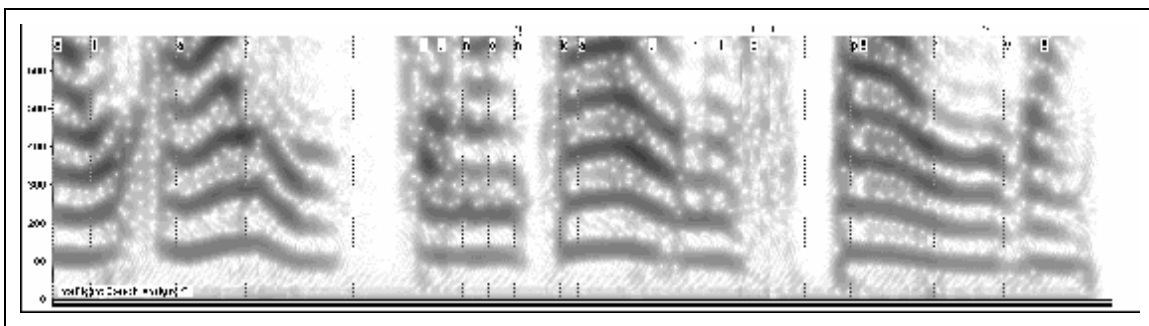
SPEECH PROCESSING

Significant amount of energy in a speech signal is contained in the region 0 to 5kHz [59]. Hence should also contain the most significant amount of emotional information, this is why we concentrate on this range of frequencies, and also reduce the amount of computational power required for processing the same. The various methods that we use to extract the emotional information from the speech signal are spectrogram, mel-frequency spectrum and linear predictive coding, which we will discuss in detail in following sections.

Spectrogram

The magnitude of a short term Fourier transform, when plotted is called a spectrogram. It is a plot with time on the X-axis and frequency on the Y-axis with the darkness representing the magnitude of the frequency band at that time. This is an important tool in speech processing.

Figure 3.3 A Spectrogram



The first step in plotting a spectrogram is calculating the short term Fourier transform of the speech signal. Computing the Fourier transform over short intervals of time enables us to capture the dynamic changes in the speech signal. Typically these parameters are estimated every 10ms, so as to obtain a smooth tracking of parameters. These short pieces of signal is cut out from the complete signal by multiplying the speech signal $s(k)$ with a windowing function, $w(k)$, to get a segmented speech signal $v_m(k)$. We use the Hamming window, which is the most commonly used windowing function.

$$w(k) = \begin{cases} 0.54 - 0.46 \cos\left(\frac{2\pi k}{n-1}\right) & k = 0, \dots, n-1 \\ 0 & k \neq 0, \dots, n-1 \end{cases}$$

where, n is the length of the time window in number of samples. The window usually represents 16ms to 25ms. A wider window would have better frequency resolution but bad time localization, while a narrow window is better for time localization but has bad frequency resolution.

Segment of the speech signal obtained by multiplication of the original signal with the Hamming window, is given by

$$v_m(k) = \begin{cases} s(k) * w(k-m) & k = m, m+1, \dots, m+n-1 \\ 0 & k \neq m, m+1, \dots, m+n-1 \end{cases}$$

where, m is the beginning sample of the time signal and $m+n-1$ the ending sample. The value of m is incremented such that there is a 10ms shift in the time signal.

Window length is important to audio processing. Longer vowels can be processed using windows up to 100ms wide, while some short burst of sounds need windows 5 to 10ms long. Since it is not possible to know before hand what kind of sounds are going to be generated, window length of 16ms to 25ms is generally used as a compromise. The time window is shifted 10ms for each set of parameters, it is possible to analyze the whole length of the signal [18]. The shift being smaller than the window length causes an overlap in the signals of consecutive sets of parameters. The overlapping samples assist in smoother temporal tracking of parameters.

Assuming the speech signal was a continuous time signal $s(t)$, its Fourier transform would be given by,

$$S(\omega) = \int_{-\infty}^{\infty} s(t)e^{-j\omega t} dt$$

The discrete time Fourier transform of a signal of length N would then be given by,

$$S^{dt}(q) = \sum_{k=0}^{N-1} s(k)e^{-jqk} \quad q \in \mathfrak{R}$$

Uniform sampling of frequency axis, θ over $[0, 2\pi]$ gives us the sampling points of

$$q(i) = \frac{2\pi i}{N} \quad 0 \leq i \leq N-1$$

Moving from a continuous frequency to discrete frequency bands we get the discrete Fourier Transform, different from discrete time Fourier transform.

$$S^d(i) = \sum_{k=0}^{N-1} s(k)e^{-\frac{j2\pi ik}{N}} \quad 0 \leq i \leq N-1$$

Now, if we computed the discrete time Fourier transform for each window of time, which is computed as described above, the short term Fourier transform would be given by

$$S(m, q) = \sum_{k=-\infty}^{\infty} v_m(k)e^{-jqk} \quad q \in \mathfrak{R}, \quad m \in \mathbb{N}$$

Substituting θ with $\theta(i)$ from above,

$$S(m, i) = \sum_{k=0}^{N-1} v_m(k)e^{-\frac{j2\pi ik}{N}} \quad 0 \leq i, \quad 0 \leq k \leq N-1$$

Once we have computed the short term Fourier transform across the whole signal using a moving window, we get an array of complex numbers. The image produced by the magnitude of the complex values as the darkness, m indicating the frame number in time, and i representing the center frequencies of the different frequency bands [18].

Word Separation

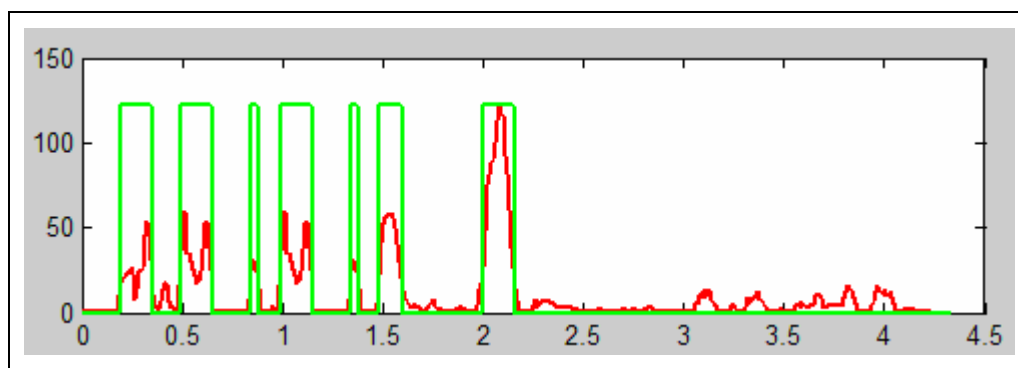
Now that we know how to plot the spectrogram, we can go ahead and compute the discrete power spectrum from the complex spectrum we computed previously. This power spectrum is the sum of the energies across all frequencies at a given time. This is done by adding up the absolute values, representing energy levels at each frequency, of the spectrogram at a given time.

$$P(m) = \sum_i |S(m, i)|$$

We then use this power threshold method similar to that used in [31] and [48]. In this method we compare the power to a predetermined threshold. Increase in speech power above the threshold mark the beginning of an utterance/word, and when the speech power drops below the

threshold it is the end of the utterance. This method has a problem however, since fluctuations in the power of the signal could cause the method to classify them as words. We thus require that there be a minimum number of frames below the threshold before and after, we call this a pause, to classify a part of a signal as a word. The results of this method are shown in Figure 3.4. There is another advantage of this, sometimes the word is uttered making the segment very short and unusable for emotion classification, so by putting two words uttered in quick succession we could successfully classify emotion in the segment.

Figure 3.4 Power plot with word separation

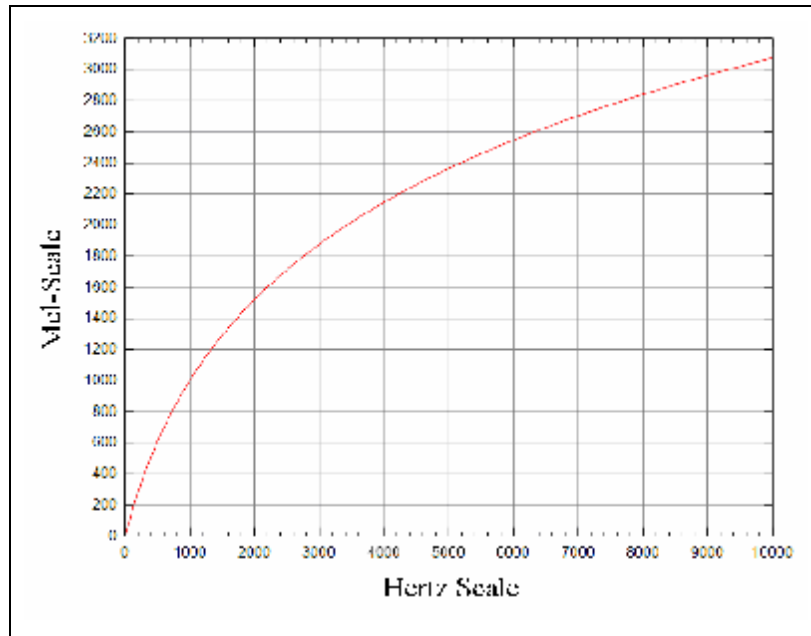


Mel Frequency Spectrum

The frequency response for the human ear is non-linear, with higher resolution at lower frequencies and lower resolution as the frequency increases. Perception experiments have shown the human ear does not recognize the frequencies as continuous but rather divides it into several groups, with the center frequencies of the band following the mel-scale. Mel-scale was proposed by Stevens, Volkman and Newmann in 1937, it is the scale of pitches judged by listeners to be equal in distance from one pitch to another. Its reference point is defined by equating a 1000 Hz tone, 40dB above the listener's threshold to with a pitch of 1000 mels [66].

$$f_{mel}(f) = 2595 \log\left(1 + \frac{f}{700\text{Hz}}\right)$$

Figure 3.5 Mel frequency Scale



The bandwidth at each of these frequencies increases with frequency, but the frequency distribution within the band is assumed to be linearly distributed. This is done by using a triangle shaped window function, with the peak at the center frequency of the given band and the upper and lower boundaries at the center frequencies of the upper and lower frequency bands respectively as shown in Figure 3.6 [53]. Figure 3.7 is a representation of the mel frequency filter matrix as a plot, where the Y axis represents the channel of the filter, X axis the fast Fourier transform index of the frequencies in Hz, and the darkness of the multiplication factor of the filter window for the particular frequency.

Figure 3.6 Mel frequency filterbank [58]

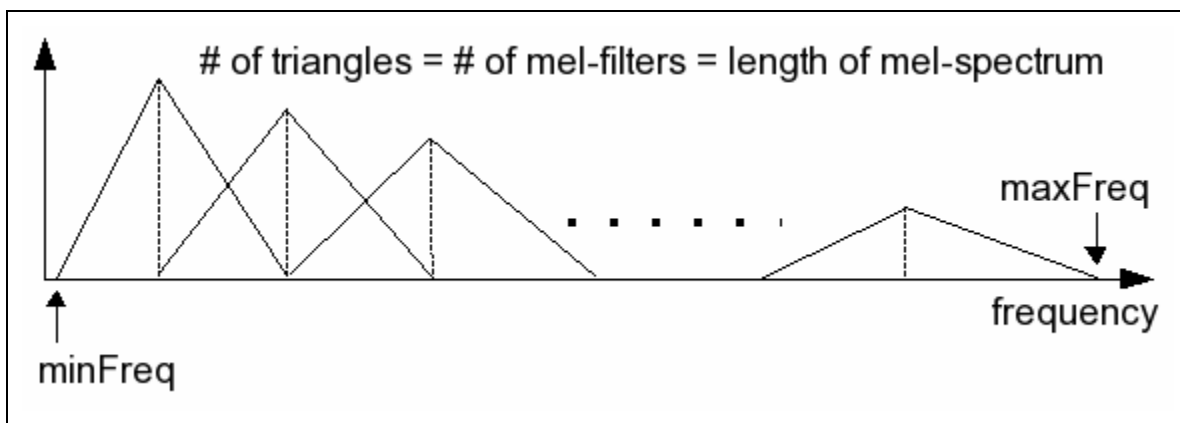
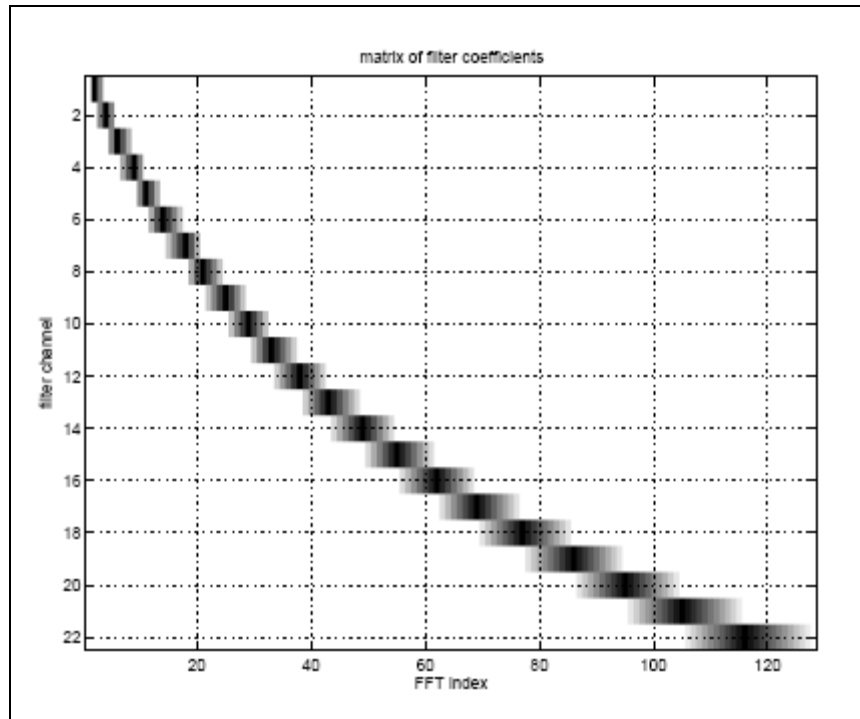


Figure 3.7 Mel frequency filter bank as a matrix



Mel Frequency spectrum is similar to a regular spectrogram, just that the frequency bands on the Y-axis are defined by the mel-scale explained above. It can be computed by multiplying the filter matrix with the spectrum of the lower half frequencies of the regular spectrum. These are the frequencies usually until 4000Hz that contain the most energy and information that the human ear perceives. We will use the energies in the mel-spectrum later while forming input vectors for the ANN, used for classification.

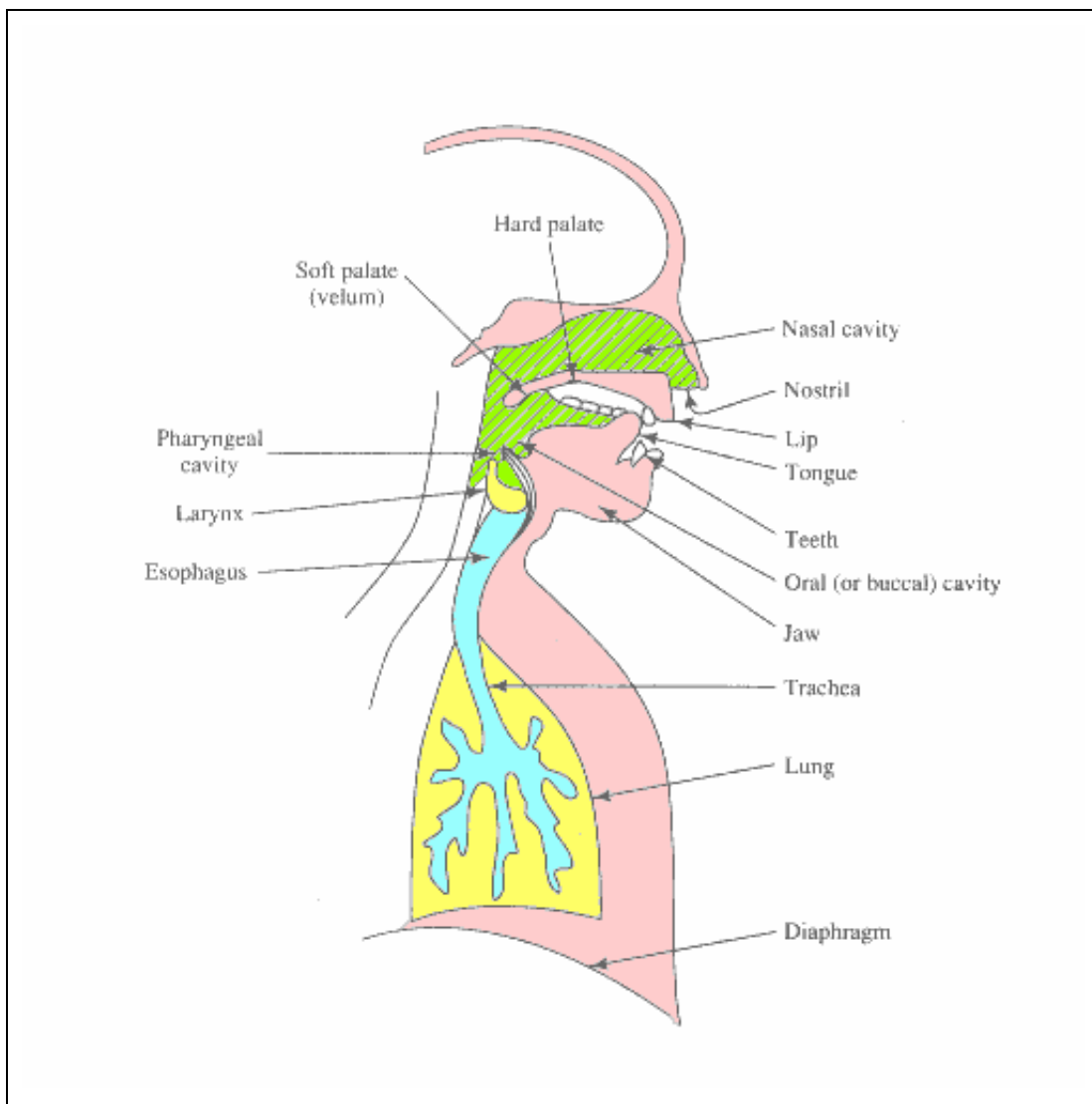
Formant Extraction using LPC

Speech Generation in Humans

Humans produce speech as they push the air out of the lungs through the articulators. ‘Articulators’ is a term used to describe all moving organs that assist in speech production. The major articulators are the tongue and the lips, while the larynx and the velum are considered secondary articulators. Refer Figure 3.8 for to aid in understanding of the structure of the human speech production system.

Speech generated by the humans can be classified into two main categories, namely voiced and unvoiced. Voiced speech is periodic and produced in the larynx when the vocal cords obstruct the flow of air from the lungs. The vocal cords modulate the airflow to create the sound variations. However it is the vocal tract that is the most important. The vocal tract is a term used to group the pharyngeal and oral cavities as one. It begins at the output of the vocal cords and ends at the lips. It produces different sounds by modifying the temporal and spectral distribution of power in the waves initiated at the glottis.

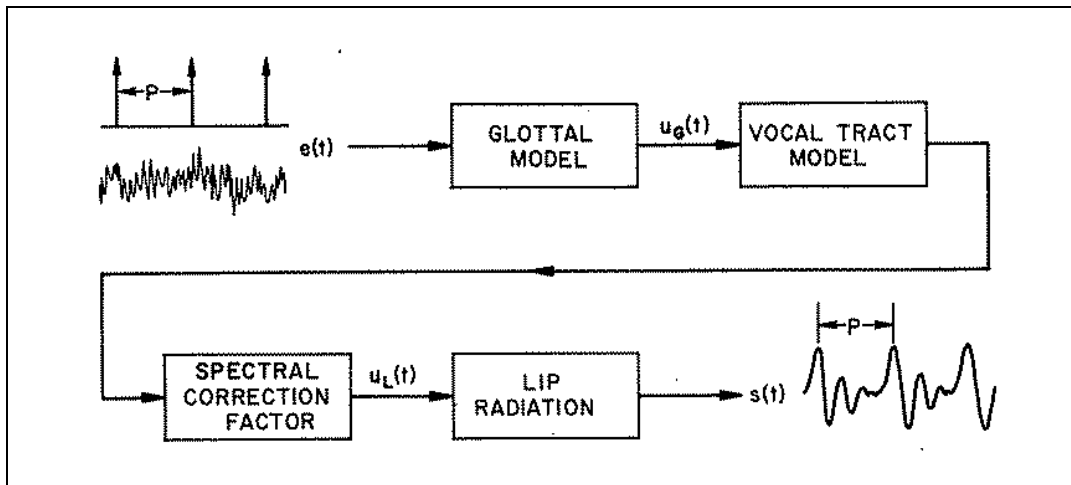
Figure 3.8 Schematic of the Human Speech Production System



Human perception of speech is based on those frequencies that are strong, i.e. have more power. Hence the vocal tract is often described in terms of its resonant frequencies, also known as formants. These resonances are because of the poles of the vocal tract transfer function. These formants are written as F_i ; where i stands for the formant number e.g. F_1, F_2, \dots, F_n . Usually there are four formants in the 0 to 4000Hz range of human speech. Movement in the articulators changes the shape of the vocal tract and hence the frequency response [18].

The nasal cavity also known as nasal tract, is another part of the speech production system. It begins at the velum and ends with the nose, and is associated with generation of nasal sounds. The velum is a trap door like mechanism at the back of the mouth and separates the nasal and the vocal tracts. During normal speech it is usually pushed up and prevents air from entering the nasal cavity. This can be lowered to acoustically couple the vocal and the nasal tracts to produce nasal sounds [62].

Figure 3.9 Linear Speech Production Model



Now that we have seen how humans produce speech, we need to represent this as a source-filter model to draw analogies to digital filters. We can then use the much studied and proved techniques in digital filtering to for formant extraction. In this model we would have two sources; first an impulse train with period P for voiced sound and second would be random noise having a flat spectrum for unvoiced sounds. These sources together are represented as $e(t)$, this signal is then modulated through a series of models to output the final acoustic pressure

waveform. The source $e(t)$ is passed through the glottal model, vocal tract model and the lip radiation model sequentially to produce the final output waveform. A schematic of the model is shown in Figure 3.9. The Glottal model converts the input signal $e(t)$ to glottal volume velocity waveform $u_g(t)$ which is then passed to the vocal tract model, this acts as a resonator amplifying certain frequencies known as formants. The volume velocity waveform at the lips $u_l(t)$ is converted to acoustic pressure waves by the lip radiation model.

The model in Figure 3.9 can be described by the following equation in Z-transform

$$S(z) = E(z)G(z)V(z)L(z)$$

where, $E(z)$ is the Z-transformed form of $e(n)$, a train of impulses spaced by pitch period $P=iT$ where i is a positive integer and T is usually set to 1.

$$E(z) = s \sum_{n=0}^{\infty} z^{-in} = \frac{s}{1 - z^{-i}}$$

for $|z| > 1$ the glottal model $G(z)$ is of the form

$$G(z) = \frac{1}{(1 - e^{cT} z^{-1})^2}$$

the lip radiation model $L(z)$ is given by

$$L(z) = 1 - z^{-1}$$

and an all pole vocal tract model $V(z)$ with K formants is given by

$$V(z) = \frac{1}{\prod_{i=1}^K [1 - 2e^{-c_i T} \cos(b_i T) z^{-1} + e^{-2c_i T} z^{-2}]}$$

The i^{th} formant frequency is given by

$$F_i = \frac{b_i}{2p}$$

and its bandwidth by

$$B_i = \frac{c_i}{2p}$$

The three models can be put together as

$$G(z)V(z)L(z) = \frac{(1 - z^{-1})}{(1 - e^{-cT} z^{-1})^2 \left\{ \prod_{i=1}^K [1 - 2e^{-c_i T} \cos(b_i T) z^{-1} + e^{-2c_i T} z^{-2}] \right\}}$$

This model simplifies to one similar to that of the vocal tract only, since cT is usually very small, much less than unity and hence as e^{-cT} approaches unity the glottal model $[G(z)]$ and the lip radiation model $[L(z)]$ cancel each other out. This can be further simplified into an all pole synthesis model given by [42]

$$S(z) = E(z) \frac{1}{A(z)}$$

Where $A(z)$ is defined by

$$A(z) = \sum_{i=0}^M a_i z^{-i} \quad a_0 = 1$$

$$A(z) = 1 + \sum_{i=1}^M a_i z^{-i}$$

$$A(z) \approx \frac{1}{G(z)V(z)L(z)}$$

The only condition being $M \geq 2K+1$.

Filter Designing using Linear Predictive Coding (LPC)

A mathematical theory for calculation of best filters and predictors was developed in the later 1940s by Norman Weiner. This was used by Weiner during World War II so as to aim at moving targets, such as aircrafts. Around the year 1967 Atal B. S., then a PhD student with the Polytechnic Institute of Brooklyn, developed the Linear predictive coding (originally called Adaptive Predictive Coding) for speech compression [2].

Linear prediction defined in simple terms is predicting the current time domain sample $\hat{s}(n)$ by using a linear combination of past time domain samples, $s(n-1)$, $s(n-2)$, ..., $s(n-m)$.

$$s[n] \approx \hat{s}[n] = -\sum_{i=1}^M a_i s[n-i]$$

The LPC coefficients can then be used to represent the sequence of the signal $s[n]$.

$$\text{Let error, } e[n] = s[n] - \hat{s}[n] = s[n] + \sum_{i=1}^M a_i s[n-i] = \sum_{i=0}^M a_i s[n-i]$$

where, $a_0 = 1$.

Taking z -transform of the above equation, we get

$$E(z) = S(z) + \sum_{i=1}^M a_i S(z) z^{-i} = S(z) \left[1 + \sum_{i=1}^M a_i z^{-i} \right] = S(z) A(z)$$

where,
$$A(z) = 1 - \sum_{i=1}^M a_i z^{-i} = \sum_{i=0}^M a_i z^{-i}$$

which can also be written as the synthesis model as

$$S(z) = E(z) \frac{1}{A(z)}$$

and the analysis model as

$$E(z) = S(z) A(z)$$

The LPC coefficients, a_1, a_2, \dots, a_m , which are parameters of the filter $A(z)$ are determined from the speech signal $s[n]$ by the methods of least squares [18].

The filter $A(z)$ is called an inverse filter since it is an all zero filter, where as $1/A(z)$ is an all pole filter and also known as the synthesis model [42].

Inverse filtering is a technique used for de-convolution of the signal, and to obtain the original signal from the final output. In our case since we start with white noise to synthesis speech, an inverse filter would convert the input signal into white noise or a constant. An inverse filter with infinite poles ($M \rightarrow \infty$) would theoretically predict the exact inverse of the input spectrum. But in practice an infinite number of poles are unrealistic. A filter with a finite number of poles cannot span the whole input signal and thus can only be designed to approximate the inverse of the signal characteristics. Markel J. D. in 1972 showed that with a properly chosen M it is possible to predict the inverse of the gross spectral structure i.e. the formant behavior.

The input samples to the inverse filter are first windowed using a Hamming window. This reduces the effect of the leading and trailing zeros of a rectangular window function. These zeros can often completely disguise the 2nd and/or 3rd formants. Since it is the formants that we are interested in, we have to segment streams of input signal.

The roots of the polynomial which are possible candidates for formant parameters are of narrow enough bandwidth that the useful information can be obtained from the frequency domain representation of the inverse filter. Peak picking of the inverse filter spectra, in general gives correct results 90% of the time [39][43].

Formant Extraction

Formants are computed as the roots of the polynomial defined by the LPC coefficients. We could also plot the polynomial and pick the peaks in the plot, these are frequencies at which the gain of the filter is maximum. Kishore S. P. et al. in their paper have shown that the difference between the spectrum of two linear prediction filters of the same signal emphasize formants, or pole frequencies of the filter. In fact, the closer the orders of two linear prediction filters are the emphasis is more uniform across the spectrum. But this method also has a drawback, if the lower ordered filter is already an overfit, i.e. when the characteristic formants have been modeled and any additional poles are being used to model the noise, then the difference will also boost the noisy peaks [54]. This is taken care of by selecting an optimal order for the filter which is a function of sampling frequency, and is given by

$$M = F_s / g \quad \text{where } \gamma = 4 \text{ or } 5 \text{ and } F_s \text{ is in kHz}$$

All the above computation is only meant for a small segment of a signal. But since the speech signal is much longer and contains continuously changing formants, we would have to divide the longer speech signal into smaller segments; that is, segments that can be satisfactorily be modeled using linear prediction. Nominal length of the segment 20 to 35 ms [43].

CLASSIFICATION OF EXPRESSION

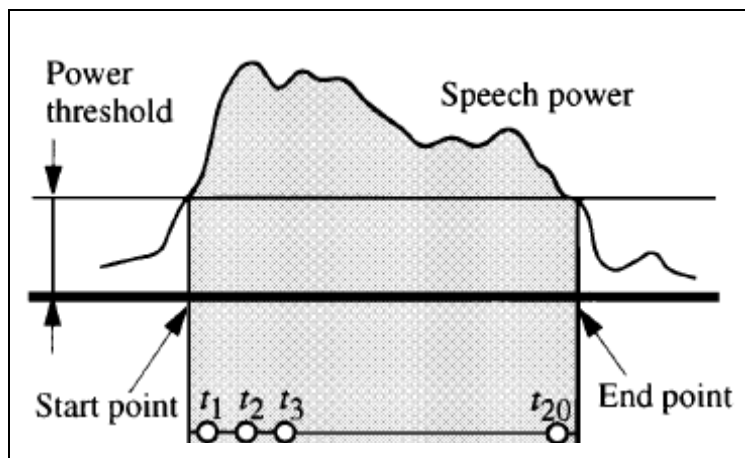
Unlike in facial expression recognition where each frame is classified, here a word is classified as a whole, similar to a phase instance. Again unlike the face where we can compute all the parameters at each frame, we cannot compute all the parameters in the unvoiced parts of the signal and hence classification is only possible for the voiced parts of the signal.

Input Vector used with ANN

The input vector to the ANN represents a word as a whole and not each frame. It consists of mean and median of formants, mel-energy levels, rate of change of formants and mel-energy; word speed, ratio of voiced and unvoiced speech, time length of word and average energy per word. The mel-energy for each of the formant is computed as the total energy of the band whose center frequency is closest to the frequency of the respective formant. These parameters are first computed along the whole length of the word, after which the signal is divided into five segments, like in Figure 3.10, and then mean and the median are computed for each segment, the

rest of the parameters such as ratio of voiced to unvoiced segment in the signal, number of words spoken per second, time length of the word and average energy per word are constant across the whole word. Features selected were the most commonly used parameters in literature [37] [3] [14] [48], and then some trials were conducted to arrive at the final feature set.

Figure 3.10 Speech Feature Extraction [48]



Training, Validation and testing sets

The speech database consisted of recordings from 10 subjects in all, 5 male and 5 female. We divided these recordings into three sets for training, testing and validation. The training set consists of 6 subjects while the validation and testing set consists of 2 subjects each. Each of these sets has an equal ratio of male and female speakers.

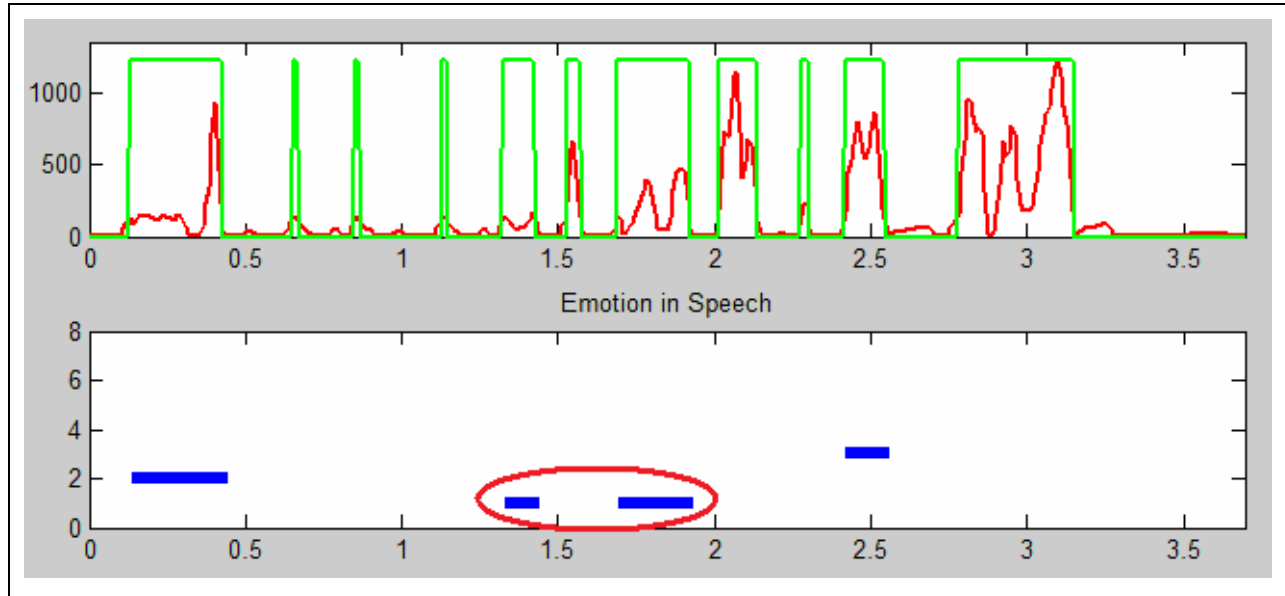
Results and Discussion

The classification of expression in voice is done on a word to word basis, and not continuously over time like in facial expression recognition.

The Figure below is a result from emotion recognition for a sentence that is supposed to be classified as an angry sentence. From the results above we see the two words in the middle of the sentence are classified correctly as angry, while the first and last word are classified as boredom and disgust. This is what we would expect when someone speaks naturally, not all words carry the same emotion in tone and meaning, and there are always some words that would emphasize a particular emotion in the sentence. But this works as a drawback while trying to put

numbers to the accuracy of recognition, since all words in the sentence are expected to be classified as angry.

Figure 3.11 Emotion Recognition in Speech



The results for classification of emotion in voice are presented below. Overall accuracy of this mode of emotion classification is 43% over all seven expressions. Emotion classification in voice seems to work best for the more subtle expressions such as anger, sad, neutral, boredom and disgust. Happy and fear being the worst expression at 0% and 3% accuracy respectively.

Table 3.2 Confusion Matrix for Emotion recognition in Voice

True\Classified	An	Bo	Di	Fe	Ha	Sa	Ne
An	48	1	4	0	1	2	3
Bo	10	19	10	0	1	7	5
Di	12	2	9	1	0	3	4
Fe	10	7	2	1	0	4	4
Ha	14	0	3	0	0	1	0
Sa	4	7	2	2	0	23	1
Ne	2	13	2	1	0	6	18

Table 3.3 Recognition Rate for Emotions in Voice

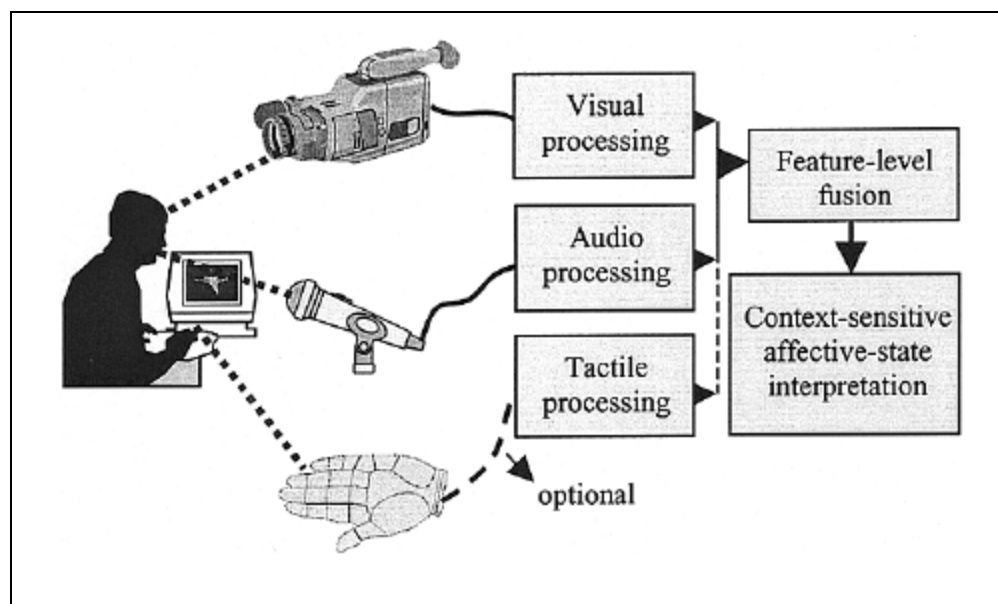
TP	An	Bo	Di	Fe	Ha	Sa	Ne
43.87	81.40	36.54	29.03	3.57	0.00	58.97	42.86

CHAPTER 4 - MULTI-MODAL EXPRESSION RECOGNITION

Emotion recognition from facial and voiced expressions processed separately is a step towards facilitating human computer interaction (HCI). Integrating these two modalities is a further step towards a more human like functional interaction. Bimodal systems have already proved themselves in the area of speech recognition, e.g. cues from lip movement have been used to improve the accuracy of speech recognition in noisy environments [56].

An ideal emotion recognition system would be one that takes in to account signal from various modalities such as facial expressions, vocal expressions, body gestures and physiological reactions. These modalities provide signals that complement each other. Usually humans tend to neglect the physiological signals since they are not easy to sense at all times. For example, a person's heart rate can only be detected by being in physical contact with the subject; which in HCI terms would mean the subject will have to be "wired" [51]. Therefore an expert system that can recognize emotions from face, voice and body gestures can be used in a wider variety of environments; that is, the biometric is captured non-invasively.

Figure 4.1 Ideal Emotion Recognition System



The way humans integrate information from the various modalities is one of the biggest challenges in developing a human computer intelligent interaction. Human interpretation of the information available from these modalities depends on the context of the situation and (intelligent) decision-making and depends on which modality to trust at a given point in time. For example, while some one is talking research has shown that vocal cues are more reliable when the facial expression appears just before or after the sentence [10].

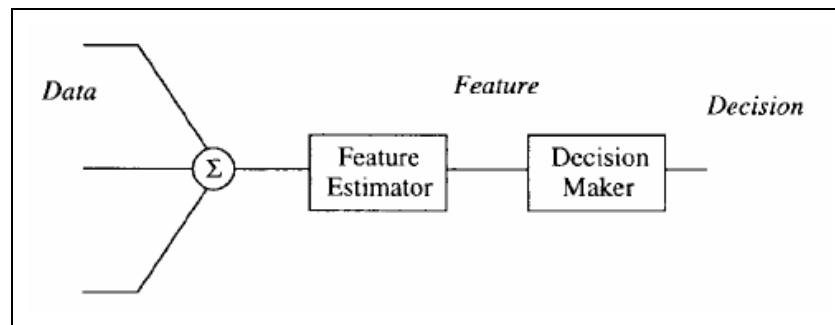
MULTIMODAL FUSION

Computer fusion of information from the various modalities can be performed at one of the three levels as follows:

Signal level

In this method the signals are fused before extracting features required by the decision maker. This can only be used on signals of the same type, for example on two or more voice signals. This method is not a feasible fusion of signals from different modalities because of the very difference in nature of signals. Instead this method can improve estimates using multiple sensors for a single modality, like an array of microphones [49]. Also in a noisy environment we can record the noise separately and subtract this signal from the original signal to obtain clean sounds.

Figure 4.2 Multimodal fusion at Signal level

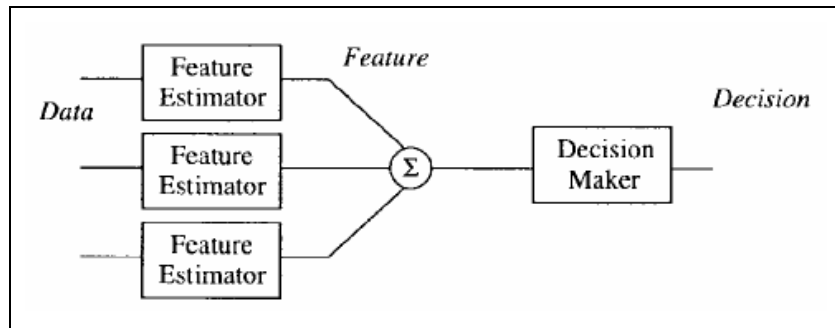


Feature level

Here features extracted from each of the modalities are fused before being passed to the decision maker. This is the fusion method used by humans while combining information from various modalities [51]. The input signals to the decision maker have to be (time) synchronized.

This is a challenge in itself since in multimodal expression recognition not all signals are present at the same point in time, but could be spaced out in time. For example we might see a smile on the face just before or after we hear a happy tone, but while talking this smile would not be discernable due to the movement of the jaw and lips. This kind of fusion is usually achieved using hidden Markov models or time biased neural networks.

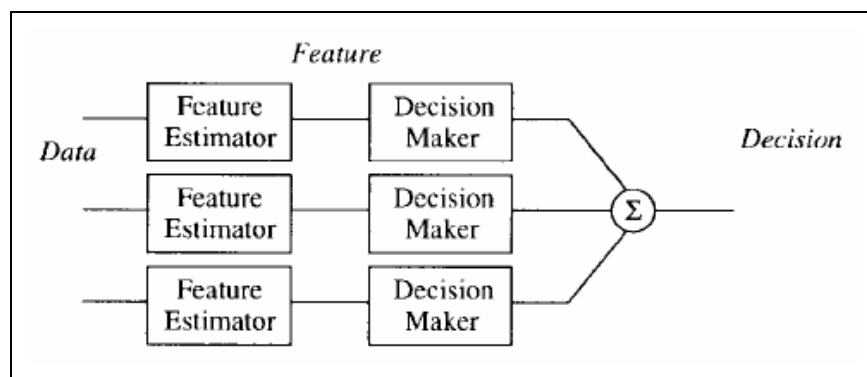
Figure 4.3 Multimodal fusion at Feature level



Decision or Conceptual level

Here, different modalities are independently analyzed for the emotions present in them. The decision from these modalities is then combined using either apriori rules or machine learning techniques. In this technique, it is possible to develop each of the modality separately before finally putting together the various decisions.

Figure 4.4 Multimodal fusion at Decision level



As mentioned earlier, processing multimodal information would require time synchronized data, i.e. video footage containing both audio and video. Absence of a readily

available audio/video database is one of the main hindrances in the current effort. The few studies [11] [16] [8] [10] [15] that have been made in this direction of multimodal emotion recognition are based on small databases collected by the research teams themselves. Even fewer of these use information fusion at the feature level. Decision level fusion of multimodal information is preferred by most researchers. With a larger amount of research on uni-modal emotion recognition and large databases available for each of these modalities, it is easier to develop the systems independently before focusing on time synchronization. If an audio/video database is available, it can be used as a testbed for each of the modalities and also as training and testing database for the multimodal system.

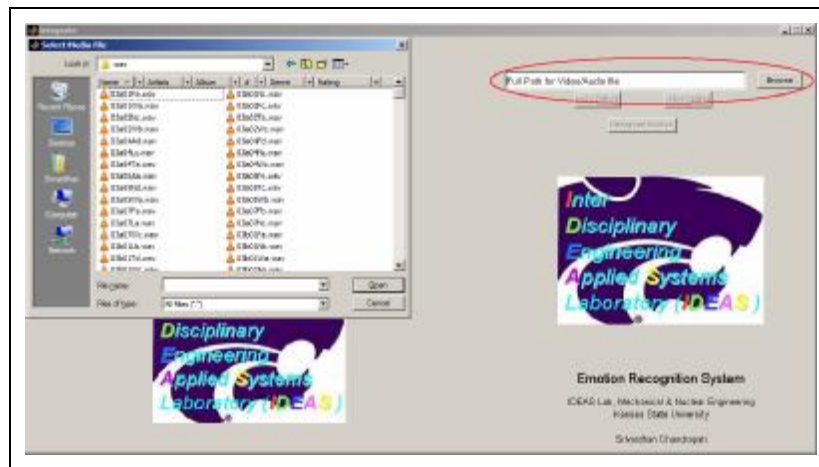
EMOTION RECOGNITION TOOLBOX

We developed a tool to integrate the emotion recognition systems based on facial and voiced expressions, explained in Chapter 2 and Chapter 3. This tool can recognize emotions in audio, video or combined audio/video tracks. In the audio/video mode, the audio and video signatures are independently presented on a same timescale. A lack of a free and easily available audio/video emotion contained database has prevented us from automating the multimodal decision making process.

Instruction Manual

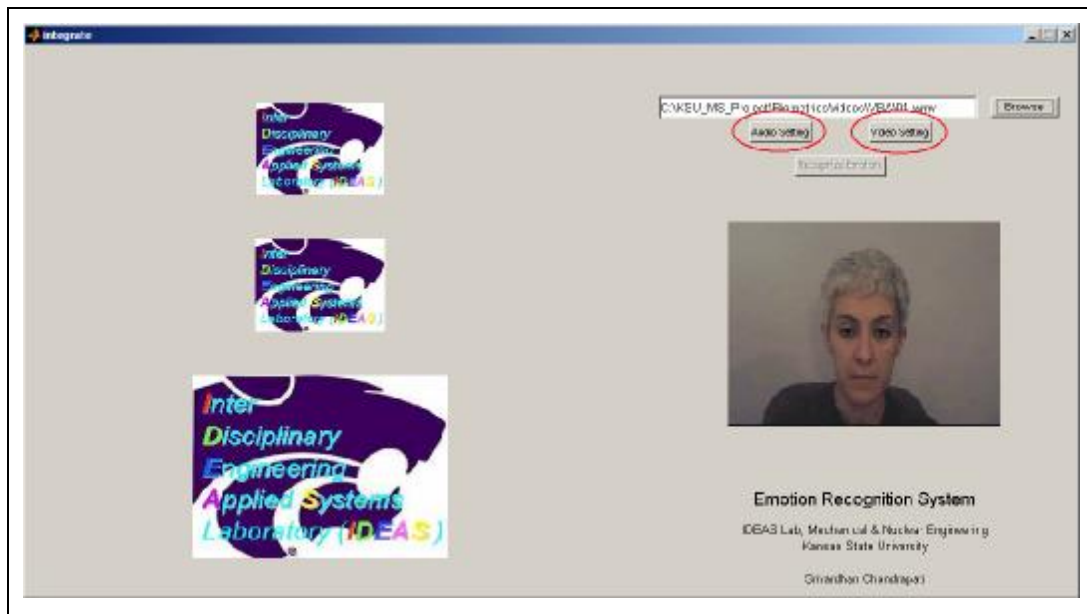
With respect to navigating through the ‘toolbox’, we first select the audio and/or video file by either entering the whole path for the file or selecting the file after clicking on the ‘Browse’ button.

Figure 4.5 Selecting a Media File



The tool reads the file using the ‘mmread’ function written by Richert [64]. The mmread function imports the data from the media file into the workspace as individual structures both audio and video. Along with the data from the file these structures also contain details about the data, such as total time of the file, sampling frequency for audio, number of channels and height and width of the video frame. Depending on the contents of the ‘Audio Setting’ and ‘Video Setting’ buttons are enabled, which are used to activate the audio and video toolboxes.

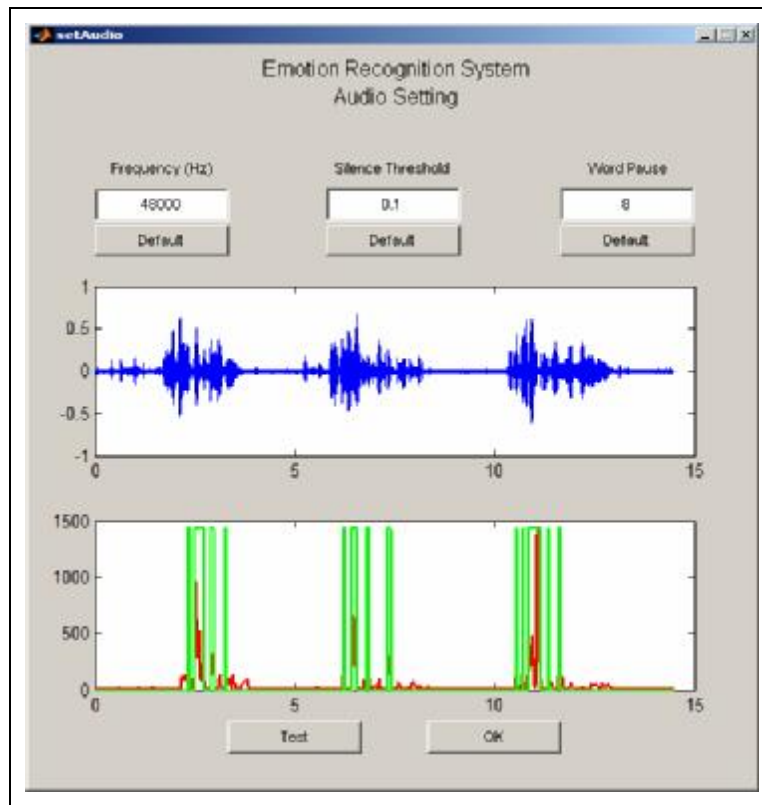
Figure 4.6 Reading a media file



Features from the audio and video are then extracted using the respective toolboxes. These toolboxes open in new windows upon clicking on the audio or video setting buttons. The audio toolbox performs word separation using energy levels and then computes a feature vector compatible with the ANN used for emotion recognition in speech, as explained in Chapter 3. Since people talk with varying speeds, the toolbox contains parameters to customize the feature extraction. This toolbox can be used to set the threshold levels as percentage of maximum energy in the speech signal. The energy level below this threshold is considered to be background noise or unspelled (unspoken) segments of the signal. The user can also change the length of the pause between two words. A ‘pause’ is the minimum number of samples for which the energy

has to be below the threshold so as to consider the segment of signal as a separate word/utterance.

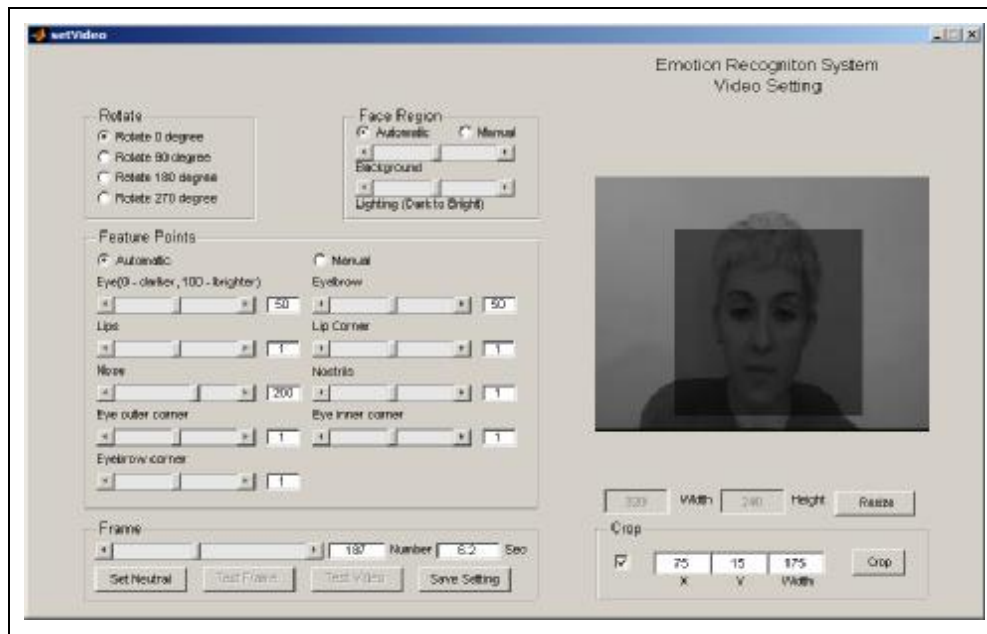
Figure 4.7 Audio Toolbox



The video toolbox is used to locate feature points for each frame in the video sequence. There are some sliders to adjust threshold for various parameters and features such as background, lighting, eyes, eye corners, eyebrows, eyebrow corners, lip, lip corners, nose and nostrils. The background and lighting thresholds are used for the automatic face region extraction algorithm, while the rest of the thresholds are used during extraction of the respective feature points. The user also has an option to switch to manual mode for either face region extraction and/or feature extraction. The 'Set Neutral' button selects the present frame as the neutral image, this is later used as a reference while creating the feature vector. By default the first frame is taken as the neutral frame. There also are some additional features to rotate images, crop the frames such that the face of the subject is in the center and resized to 256 x 256 pixels. The shaded window over the image of the face (Figure 4.8) represents the final cropped image. The

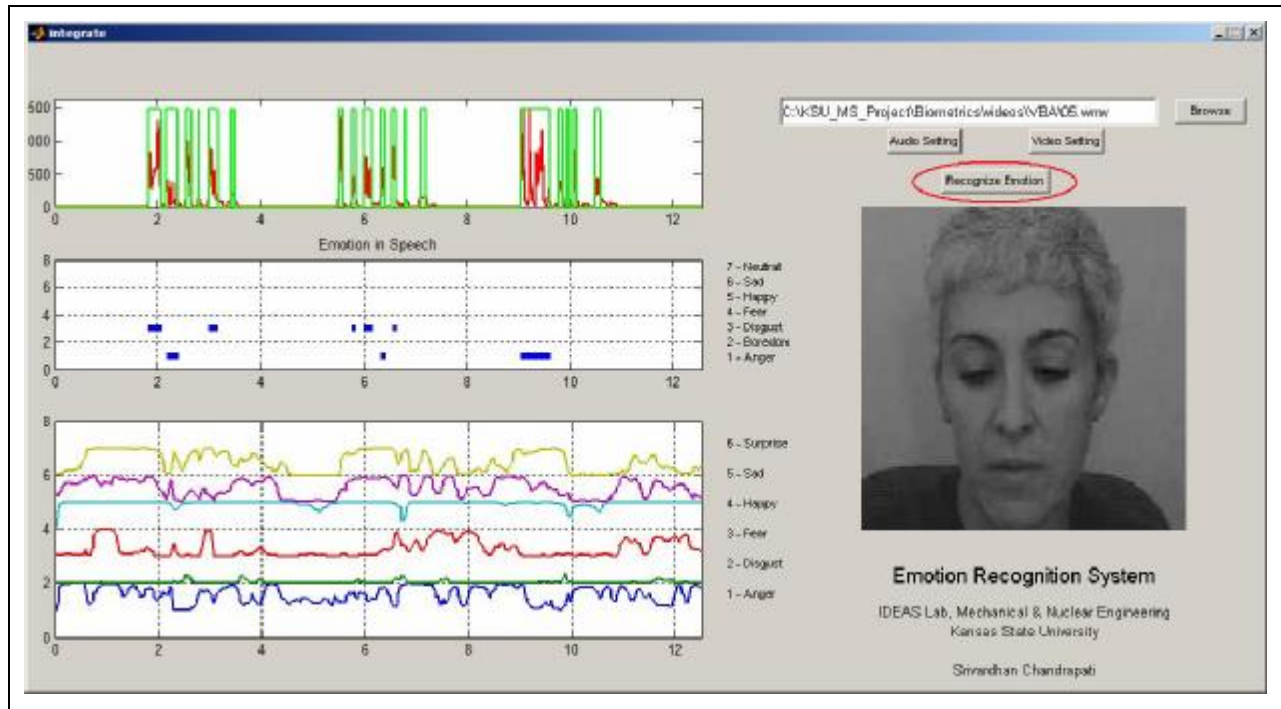
user can use the slider in the frame panel to scroll through all the frames and make sure that the selection is correct. After the video is cropped and resized to the correct size, the ‘Test Frame’ button is enabled. This gives the user a chance to either test the threshold setting selected, or to manually select feature points for the any one frame and use automatic extraction for subsequent frames. Once the user is satisfied with the feature points extracted in the test frame, the test video button then locates features of the whole video sequence. The ‘Save Setting’ button then passes the information for the extracted feature points to the main window where the emotions are recognized.

Figure 4.8 Video Toolbox



Once the information from all the available modalities is available, the ‘Recognize Emotion’ button is enabled (Figure 4.9). This button loads the neural networks for face and voice emotion recognition, and analyzes the available features for emotions. In the voice domain emotions are recognized at word or utterance level, while the expression in faces is recognized for each frame.

Figure 4.9 Emotion Recognition Toolbox



TESTING and ANALYSIS

Testing of the multi-modal system was unfortunately limited because of lack of any freely available multi-modal databases. For purposes of consistency with the audio database testing of the system should be in German. With the help of a German speaking volunteer (to whom we are very thankful) we were able to record the same phrase with facial and voiced expressions. From this we have been able to perform some preliminary tests. For reasons to be determined all of the recorded data from this user was classified as either 'anger', 'boredom' or 'disgust' by our system. Since boredom was not a part of the training database of the facial expression recognition system and could be recognized only in voiced expression, we were not able to completely test the system for it. However the classification of anger and disgust helped test the system to see if the addition of emotion recognition in speech enhanced the recognition of expressions displayed by the face. If one recalls, one of the problems encountered with the facial expression recognition system was that it misclassified disgust as anger and fear as surprise.

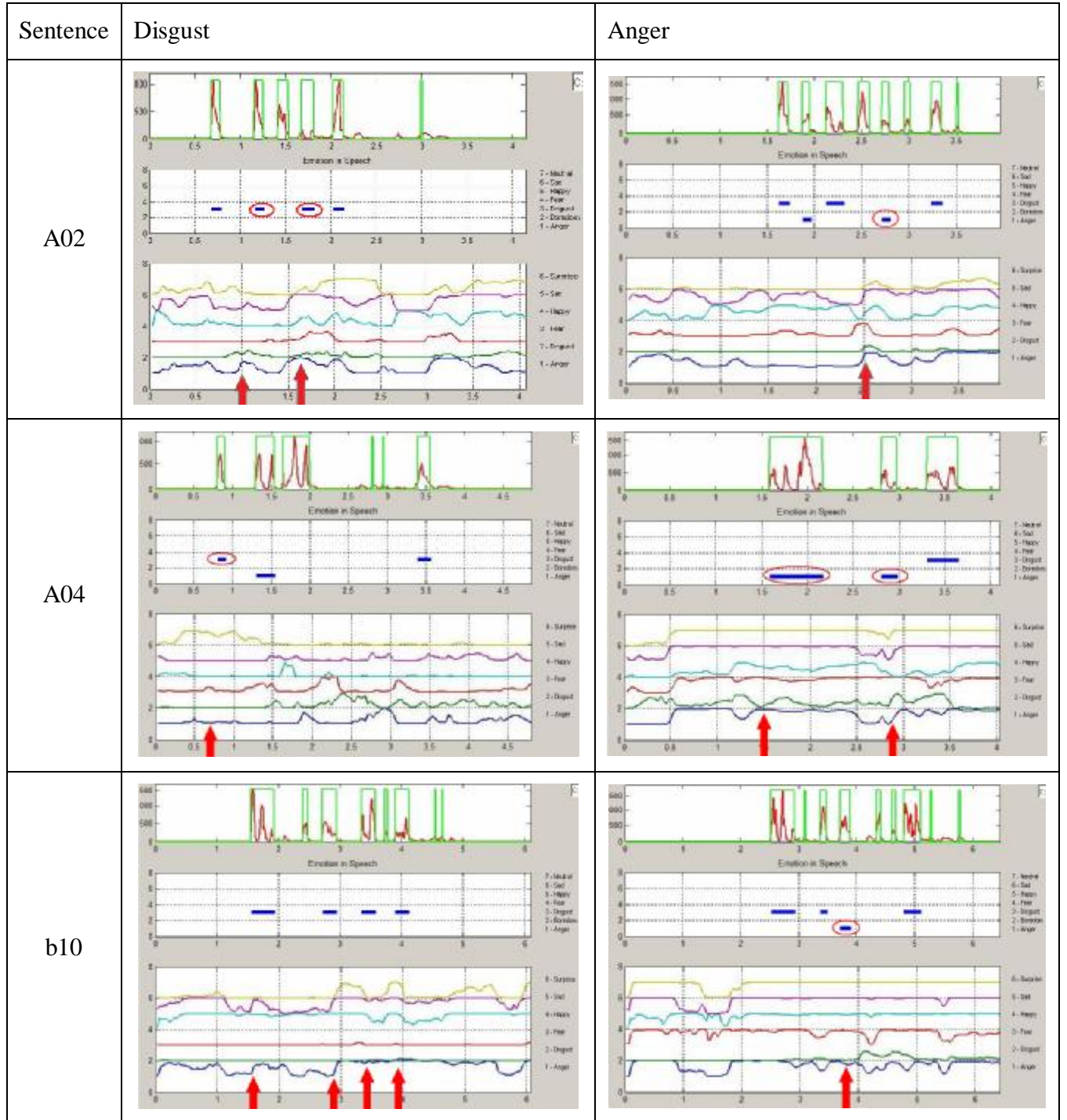
While interpreting results from the multi-modal toolbox, we rely more on the results of the facial expression recognition system and use the voiced emotion recognition system to supplement the information. It is thus not the primary decision maker or modality for emotion recognition. This is because of lack of extensive testing of the voiced emotion recognition system and better recognition rates achieved by the facial expression recognition system. Results from voiced emotion recognition can facilitate the process when the user has to make decisions pertaining to the confused emotion (anger-disgust pair and surprise-fear pair). At present we are only able use these results to make a distinction between anger-disgust and not between surprise-fear. Here again the German speech database used for training the emotion recognition system did not contain expressions of surprise and even though it contained fear, fear was one of the least recognized emotion in voice.

Figure 4.10 below compares results from the multimodal toolbox for disgust and anger expressions. Looking at the results we observe that the best time to observe an emotion is just before an utterance. Since an utterance will change the shape of the mouth and hence move the mouth's feature points, it is not reliable to read the expression from the facial expression recognition system when the word is actually being pronounced. In the result for disgust expressed emotion of sentence 'a02' we observe that the facial expression recognition system shows a slight trace of disgust just before the second word. In fact the word is classified as disgust by the voiced emotion recognition system; thus reinforcing the conclusion that the expression is disgust. Also the consistent classification of all the words as disgust points towards the fact. We can observe a similar scenario at the third word in the identification of disgust for sentence 'a04'. In an unbiased opinion for the sentence a04 of disgust, I would classify the first word as surprise based on the facial expression recognition, and the second as anger based on the voiced emotion recognition since, the facial expression recognition is inconclusive for the second word. While with sentence b10 I would classify any individual word as angry based on results from facial expression recognition, but asked for an opinion based on the whole segment, I would classify it as disgust because of the consistency of expression in voice, better recognition rate of disgust in voice over face and knowledge about the anger-disgust confusion.

Further for angry expressions, we see that anger is very prominent in the facial expression recognition system especially before the utterance of the word and when corroborated by recognition of anger in speech, leads us to conclude that the utterance is an angry one. It

seems possible to at least improve the classification of the expression recognition process while simultaneously reduce the 'anger-disgust' confusion when emotion recognition is limited to facial expressions.

Figure 4.10 Comparing Multi-Modal Results for Disgust and Anger



CHAPTER 5 - Conclusion

This research is directed towards developing a multi-modal emotion recognition system. This system was developed because of the general interest of the researchers in the area of robotics, specifically social robots. In the future when robots will begin co-existing with humans in the same environment they will have to learn to be socially interactive, and emotion recognition is the key skill since humans communicate only 7% through spoken words, while the remaining 93% communication is through emotional expression in face and voice. Another interesting area where such a system will be handy is in law enforcement for detection of intent to deceive. This relies on the fact that emotions expressions are not completely voluntary and can be used to gain an insight into the person's actual feelings.

In order to achieve our goals of building an emotion recognition system we firstly build systems to recognize emotions in face and voice separately. Emotion recognition in face was in three stages, namely face region estimation using seeded region growing algorithm, feature point extraction using particle swarm optimization techniques and lastly facial expression classification using feed forward artificial neural networks. While feature extraction in voice was done using techniques such as Fourier transforms and linear predictive coding, they were classified using feed forward artificial neural networks. These two systems were then put together into a user friendly multi-modal expression recognition toolbox, by synchronizing the systems based on the time scale. All the programming for the purpose was in Matlab.

Some conclusions we can draw from the research above are

- Expression recognition using both facial and voice expressions helps solve the issue of confusion and frequent misclassification among some expression pairs, particularly anger and disgust.
- Surprise and happy are the most accurately recognized facial expressions, while disgust and fear are the least accurately recognized.
- There are clues that people from Asian cultures express some emotions differently when compared those from European cultures. Hence the emotions pairs of anger-disgust and surprise-fear are the misclassified often when a neural network is trained on one database and tested on another.

- Emotions of “happy” and “fear” are difficult to recognize in voice.
- Since time dependent machine learning techniques allow for extraction of temporal information they are better suited for emotion recognition in voice, as against the feed forwards networks that have been used.
- Use of time dependent systems also allow for complete multi-modal fusion of emotional information from voice and face.

The facial expression recognition system developed was trained on the JAFFE database, consisting of images of Japanese females. The testing was performed on MMI database, consisting of a mix of males and females of European background. This system recognizes emotions in faces with an accuracy of about 55% over 6 emotions (anger, disgust, fear, happy, sad and surprise), across the noted cultures. The expression “surprise” was the easiest to classify with about 92% accuracy even on a database on which the system had not been trained. This was followed by “happy” at 90% accuracy, “anger” at 57%, “sad” at 50% and, “fear” and “disgust” at 17% and 13% respectively. Upon closer inspection of the results of expression recognition we realize that 70% of fear was misclassified as surprise and 87% of disgust misclassified as anger. We suspected these to be because of cultural differences. While performing some trial recordings of emotional expressions in the lab, we realized that perhaps those of Asian origin were less expressive when compared to those from the USA or Europe. To test this hypothesis we trained another neural network using the same methodology used the first time, except this time it was trained using the MMI and tested using the JAFFE. As we suspected this time 74% of anger that was misclassified was classified as disgust and 77% of misclassified fear was classified as surprise. This is the exact opposite of previous scenario. The misclassification of fear as surprise was 3.57% this time as opposed to 87% previously and disgust as anger was 38% down from 70%. Hence leading us to believe that there are indeed some cultural differences. However with the small amount of data available and the facial expression system designed to optimize computational load by extracting only 17 feature points, this cannot be said with absolute conviction. Further research aimed at studying cultural differences in facial expressions, would be needed to either prove or disprove this finding. Such a research would not be limited by computing power and hence could be studied using more sophisticated image processing techniques which can extract subtle details from the image.

We then developed an initial system to recognize emotions in speech. However we were not able to test it as extensively as we could test the face, predominantly because of the lack of a freely available emotionally labeled speech database. The one database that we had used to develop the system was in German. It consisted of recordings from 10 subjects of which we used 6 subjects for training, 2 subjects for validation and 2 subjects for testing. Our system could recognize emotions in voice with 40% accuracy across 7 emotions (anger, boredom, disgust, fear, happy, sad and neutral). Happy and Fear were the worst recognized at 0% and 3% respectively. This corroborates previous findings where happy and fear have been difficult to recognize [6]. The more subtle expressions such as anger, sad and disgust were recognized with higher accuracy when compared to facial expression recognition.

We then developed the multi-modal toolbox, which can take either an audio or video file as an input, classify emotions present in each of the modalities and display them on a synchronized time-scale. Having the information on a synchronism time scale helps the user recognize and compare emotions in face and voice, expressed at the same time. Complete testing of this was not possible, because of lack of databases, except for a small set of video provided by a volunteer. The volunteer voiced the phrases in German with the noted emotion and corresponding expressions. However as English is Germanic language and spoken under culturally similar conditions, we see this database as valid. Unfortunately most of the speech data from the volunteer was either classified as anger, boredom or disgust, preventing any further extensive multimodal testing. We took this opportunity to check if the addition of the voice data would help with the anger-disgust confusion present while testing facial expression database. Some tests were performed only on the anger and disgust expressions recordings, and the result of combining both the modalities appears to be encouraging. We were able to recognize trends in the combined data pointing towards either one of the two disgust or angry expressions. Any indication of disgust in the facial expression need not be very prominent, just before a word along with the emotion being classified as disgust in the speech can be used to classify the segment of data as disgust. Disgust can also be recognized by its prominent presence in speech across a number of consecutive words. Anger is characterized by its strong presence in facial expressions, especially before a word and is present in short bursts in speech. These results hold promise of better recognition accuracy with a multimodal system.

Some of these results above might be biased by my personal opinion, and knowledge of results expected. A complete unbiased estimation of recognition rates would only be possible when complete multi-modal fusion is achieved. As long as the system is open ended, leaving final judgment to the user, a fair way to perform such an analysis would be to first get a group of people label the multi-modal information for emotion. Then process this information through a recognition system such as ours and get a second group of people (those who have not seen the videos previously) to interpret the results after receiving some basic training on interpreting the results.

The main limitation with achieving complete multi-modal fusion is the lack of a freely available and emotionally labeled standard multi-modal database. Such a database is to be built by recording on video, a group of subjects expressing emotions while a second group of blinded subjects classify these recordings. Any clip that is consistently classified correctly will then be included in the database. So as to use the database for future research with out of plane rotation, the video should be recorded simultaneously with multiple cameras. This database should preferably be in English so as to making it easier to find test subjects in the predominantly English speaking research communities. Such a database would also provide a level ground for measuring performances of systems developed by various researchers.

Another technical aspect that prevents complete fusion in our case is the difference in the classification methods used for the two modalities. Facial expression recognition was classified using frame instance, where complete temporal data is available for each emotion. Emotions in voice were classified using phase instance, each word was classified as a particular emotion. Feature level fusion is possible if emotion recognition in voice is also performed on frame to frame basis so as to obtain temporal information. Another advantage of a frame-by-frame technique in speech is that information is not lost due to averaging, unlike our present approach where the word is divided into a fixed number of segments and information averaged over each segment. The information had to be averaged over each segment because a feed forward neural network cannot accept input vectors of varying length or use a stream of vectors. Time dependent machine learning techniques such as time delayed neural network or hidden Markov models are capable of accepting a stream of input vectors, and hence reduce the effects of averaging. Feed-forward neural networks were selected for the purpose of the research since they were suitable to the first task at hand, which was emotion recognition on snapshots. This method

could be extended to a video since the basic concepts were the same for both snapshots and videos. In case of videos all that had to be done was split it into individual frames and treat each one of them as separate snapshot. We initially continued using feed forward networks for voice too because of the previously gained expertise in the area. At the time we did not foresee this problem with getting temporal information in speech and obtaining frame by frame information of emotions in speech. This was realized only towards the end of this research after the emotion recognition system in speech was developed and while building the multimodal toolbox.

On a parting note some suggestions for future work in this direction would be to take up development of a multi-modal database with utmost importance. Further use of time-dependent machine learning techniques for emotion extraction in voice, because these can extract temporal information of emotions in voice and assist in multi-modal fusion. Matlab is an excellent tool for prototyping with its easy handling of matrices and inbuilt toolboxes for image processing, signal processing and neural networks, but not the most efficient in terms of processing speed. For being able to run this system in real time it will have to be reprogrammed in a more efficient language such a C/C++.

References And Bibliography

1. Adams, R., Bischof, L., 1994, "Seeded Region Growing", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 16, No. 6, June 1994
2. Atal, B.S., "The history of linear prediction", Signal Processing Magazine, IEEE , vol.23, no.2, pp.154-161, March 2006
3. Banse, R. and Scherer, K. R., 1996, "Acoustic Profiles in Vocal Emotion Expression", Journal of Personality and Social Psychology, vol. 70, no. 3, pp. 614-636
4. Bartlett, M.S., Littlewort, G.C., Frank, M.G., Lainscsek, C., Fasel, I. and Movellan, J.R., 2006, "Automatic Recognition of Facial Actions in Spontaneous Expressions", Journal of Multimedia, vol. 1, no. 6, September 2006
5. Bassili, N. J., 1978, "Facial Motion in perception of faces and of Emotional Expressions", Journal of Experimental Psychology: Human Perception and Performance, Vol. 4, No. 3, pp. 373-379.
6. Bo, X., Ling, C., Gen-Cai, C., Chun, C., 2005, "EmoEars: An Emotion Recognition System for Mandarin Speech", Computational Intelligence and Security, Lecture Notes in Computer Science, Springer Berlin, vol. 3801/2005, pp. 941-948
7. Burkhardt, F., Paeschke, A., Rolfes, M., Sendlmeier, W. and Weiss, B., "A Database of German Emotional Speech", Proc. of Interspeech 2005, Lissabon, Portugal
8. Busso, C., Deng, Z., Yildirim, S., Bulut, M., Lee, C. M., Kazemzadeh, A., Lee, S., Neumann, U., and Narayanan, S., "Analysis of emotion recognition using facial expressions, speech and multimodal information", In Proceedings of the 6th international Conference on Multimodal interfaces (State College, PA, USA, October 13 - 15, 2004). ICMI '04. ACM, New York, NY, 205-211
9. Canny, J., 1986, "A Computational Approach to Edge Detection", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 6, pp. 679-698
10. Chen, L.S. and Huang, T.S., "Emotional expressions in audiovisual human computer interaction", Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference on , vol.1, no., pp.423-426 vol.1, 2000

11. Chen, L.S., Huang, T.S., Miyasato, T. and Nakatsu, R., "Multimodal human emotion/expression recognition," Automatic Face and Gesture Recognition, 1998. Proc. Third IEEE International Conference on , vol., no., pp.366-371, 14-16 Apr 1998
12. Chennamsetty, N. K, "Development of automatic facial expression recognition system using Gabor wavelets and learning vector quantization networks", M.S. Mechanical Engineering Thesis, University of Missouri, Rolla, 2005
13. Chibelushi, C.C. and Bourel, F., "Facial Expression Recognition: A Brief Tutorial Overview", In CVonline: On-Line Compendium of Computer Vision. R. Fisher (Ed.), January 2003. URL: <http://www.dai.ed.ac.uk/cgi-bin/rbf/CVONLINE>
14. Cowie, R., Douglas-Cowie, E., Tsapatsoulis, N., Votsis, G., Kollias, S., Fellenz, W. and Taylor, J.G., "Emotion recognition in human-computer interaction", Signal Processing Magazine, IEEE , vol.18, no.1, pp.32-80, Jan 2001
15. Datcu, D. and Rothkrantz, L., "Multimodal Web based system for human emotion recognition", Proceedings ISC2007, pp. 91-98, EUROSIS-ETI, June 2007.
16. De Silva, L.C. and Pei Chi Ng, "Bimodal emotion recognition," Automatic Face and Gesture Recognition, 2000. Proc. Fourth IEEE International Conference on , vol., no., pp.332-335, 2000
17. de Sousa, R., "Emotion", The Stanford Encyclopedia of Philosophy (Summer 2007 Edition), Edward N. Zalta (ed.). URL: <http://plato.stanford.edu/entries/emotion>
18. Deng, L. and O'Shaughnessy, D., "Speech Processing: A Dynamic and Optimization Oriented Approach", Marcel Dekker Inc, 2003
19. Ekman, P. and Friesen, W. V., "Facial Action Coding System", Palo Alto: Consulting Psychologist Press, 1978.
20. Ekman, P. and Friesen, W. V., "Unmasking the Face. A guide to recognizing emotions from facial clues", Palo Alto: Consulting Psychologists Press, 1975.
21. Ekman, P., "Emotions Revealed", Times Books, 2003
22. Ekman, P., Friesen, W. V. and Hager, J. C., "Facial Action Coding System", Salt Lake City: A Human Face, 2002.
23. Ekman, P., O'Sullivan, M., Friesen, W.V., and Scherer, K.R., 1991, "Face, voice and body in detecting deception", Journal of Nonverbal Behavior, vol. 15, pp. 125-135
24. Ekman, P., "Telling Lies", W. W. Norton and Company Inc., New York, 2001

25. Fasel, B. and Luetttin, J., 2003, "Automatic Facial Expression Analysis: A Survey", Pattern Recognition, Vol. 36, pp. 259-275.
26. Gao, Y., Leung, M.K.H., Hui, S. C. and Tananda, M.W., 2003, "Facial expression recognition from line-based caricatures", IEEE Transactions on Systems, Man and Cybernetics, Part A, vol. 33, no. 3, pp. 407-412, May 2003
27. Gladwell, M., "The Naked Face: Can you read people's face just by looking at them", The New Yorker Magazine, vol. August, No. 5, pp 38-49, 2002
28. Greengrass. M., 2002, "Emotion and cognition work together in the brain", Monitor on Psychology, vol. 33, no. 6, June 2002. URL: <http://www.apa.org/monitor/jun02/emotion.html> (05/12/2008)
29. Haisong, G., Qiang, J., "An automated face reader for fatigue detection", Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on , pp. 111-116, 17-19 May 2004
30. Haisong, G., Qiang, J., "An automated face reader for fatigue detection", Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on , pp. 111-116, 17-19 May 2004
31. Hammal, Z., Bozkurt, B., Couvreur, L., Unay, D., Caplier, A. and Dutoit, T., 2005, "Passive versus active: vocal classification system", Proc. EUSIPCO, European Signal Processing Conference, Antalya (Turkey)
32. Kaplan, F., 2005, "Everyday robotics: robots as everyday objects", in Proc. of the 2005 Joint Conference on Smart Objects and Ambient intelligence: innovative Context-Aware Services: Usages and Technologies (Grenoble, France, October 12 - 14, 2005). sOc-EUSAI '05, vol. 121. ACM, New York, NY, 59-64
33. Kennedy, J. and Eberhart, R., "Particle Swarm Optimization", Neural Networks, 1995. Proc., IEEE Intl. Conf. on , vol. 4, no., pp.1942-1948, Nov/Dec 1995
34. Kennedy, J., Eberhart, R. and Shi, Y., "Swarm Intelligence", Morgan Kaufmann Publishers, San Francisco, CA, 2001
35. Krose, B., and van der Smagt, P., "An Introduction to neural networks", Univ. of Amsterdam, 1996
36. Kulkarni, D. A., Computer Vision and Fuzzy-Neural Systems, Prentice Hall, 2001

37. Lee, M. C. and Narayanan, S.S., 2005, "Toward detecting emotions in spoken dialogs", IEEE Transactions on Speech and Audio Processing, vol.13, no.2, pp. 293-303, March 2005
38. Lin, C. S., Chen, H. T., Lin, C. H., Yeh, M. S., Lin, S. L., 2005, "Polar Coordinate Mapping Method for an Improved Infrard Eye Tracking System", Biomedical Engineering – Application, Basis & Communication, vol. 17, no. 3, pp. 141-146, June 2005
39. Littlewort, G., Bartlett, M., Fasel, I., Susskind, J. and Movellan, J., 2006, "Dynamics of facial expression extracted automatically from video", Image and Vision Computing, Vol. 24, No. 6, pp. 615-625.
40. Littlewort, G., Stewart, M., Fasel, I., Chenu, J. and Movellan, J., 2003, "Analysis of Machine Learning Methods for Real-Time Recognition of Facial Expression from Video", Technical Report, Machine Perception Laboratory, Institute for Neural Computation, University of California, San Diego.
41. Lyons, M. J., Akamatsu, S., Kamachi, M. and Gyoba, J., 1998, "Coding Facial Expressions with Gabor Wavelets", Proceedings, Third IEEE International Conference on Automatic Face and Gesture Recognition, Nara Japan, IEEE Computer Society, pp. 200-205.
42. Markel, J. D. and Gray, A. H. Jr., "Linear Prediction of Speech", Springer-Verlag, Berlin Heidelberg, 1976
43. Markel, J., "Digital inverse filtering-a new tool for formant trajectory estimation," Audio and Electroacoustics, IEEE Transactions on , vol.20, no.2, pp. 129-137, Jun 1972
44. Marsh, A. A., Effenbein, H. A., and Ambady, N., 2003 "Nonverbal 'accents': Cultural differences in facial expressions of emotion", Psychological Science, vol. 14, pp. 373-376
45. Mehrabian, A., 1968, "Communication without Words", Psychology Today, Vol. 2, No. 4, pp. 53-56.
46. Mitlell, T. M., "Machine Learning", McGraw Hill, 1997
47. Nicholson, J., Takahashi, K. and Nakatsu, R., "Emotion Recognition in Speech", Neural computing and applications, vol 9, pp. 290-296, Springer-Verlag London, 2000

48. Nicholson. J., Takahashi. K., Nakatsu. R., "Emotion recognition in speech using neural networks", Neural Computing and Applications, vol. 9, issue 4, pp. 290-296 Springer-Verlag, London, 2000
49. Paleari, M. and Lisetti, C. L., "Toward multimodal fusion of affective cues", in Proc. of the 1st ACM international Workshop on Human-Centered Multimedia (Santa Barbara, California, USA, October 27 - 27, 2006). HCM '06. ACM, New York, NY, 99-108.
50. Pantic, M. and Rothkrantz, L. J. M., 2003, "Automatic Analysis of Facial Expressions: The State of the Art", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 22, No. 12, pp. 1424-1455.
51. Pantic, M. and Rothkrantz, L.J.M., "Toward an affect-sensitive multimodal human-computer interaction", Proceedings of the IEEE , vol.91, no.9, pp. 1370-1390, Sept. 2003
52. Pantic, M., Valstar, M. F., Rademaker, R. and Maat, L., 2005, "Web-based Database for Facial Expression Analysis", Proceedings IEEE International Conference on Multimedia and Expo, Amsterdam, The Netherlands.
53. Plannerer, B., "Introduction to Speech Recognition", 28 March 2005. URL: <http://www.speech-recognition.de/pdf/introSR.pdf>
54. Prahallad, K., Varanasi, S., Veluru, R., Krishna, M. B. and Roy, D. S., "Significance of Formants from Difference Spectrum for Speaker Identification", in Proceedings of Interspeech, Pittsburgh,2006
55. Rojas, R., "Neural Networks: A Systematic Introduction", New York:Springer-Verlag, 1996
56. Sebe, N., Cohen, I., Gevers, Th. and Huang, T. S., "Multimodal approaches for emotion recognition: a survey (Invited Paper)", SPIE, Internet Imaging, San Jose, 2005
57. Theologos, A., Stelios, B. and Ioannis. D, 2006, "Automatic Recognition of Emotionally Coloured Speech", Proceeding of world academy of science, engineering and technology, vol. 12, March 2006
58. URL: <http://cmusphinx.sourceforge.net/sphinx4/javadoc/edu/cmu/sphinx/frontend/frequencywrap/MelFrequencyFilterBank.html> (05/12/2008)
59. URL: <http://cnx.org/content/m0049/latest/#spectrogram> (05/12/2008)
60. URL: <http://en.wikipedia.org/wiki/Polygraph> (05/12/2008)

61. URL: <http://gundam.cs.yale.edu/TheLab.htm> (05/12/2008)
62. URL: <http://ispl.korea.ac.kr/~wikim/research/speech.html> (05/12/2008)
63. URL: <http://scien.stanford.edu/class/ee368/projects2000/project2/img2.gif> (05/12/2008)
64. URL:
<http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=8028&objectType=FILE> (05/12/2008)
65. URL: <http://www.oatsoft.org/Software/whistling-user-interface> (05/12/2008)
66. URL: <http://www.sfu.ca/sonic-studio/handbook/Mel.html> (05/12/2008)
67. Vrij, A., Edward, K., Roberts, K. P., and Bull, R., 2000, "Detecting deceit via analysis of verbal and nonverbal behavior", *Journal of Nonverbal Behavior*, vol. 24, no. 4, pp. 239-263
68. Weisstein, E. W., "Method of Steepest Descent", From MathWorld--A Wolfram Web Resource. URL: <http://mathworld.wolfram.com/MethodofSteepestDescent.html> (05/12/2008)
69. Werbos, P. J., "Roots of Backpropagation: From ordered Derivatives to neural networks and political forecasting", John-Wiley & Sons Inc., 1994
70. Yrizarry N., Matsumoto D. and Wilson-Cohn C., 1998, "American-Japanese Differences in Multiscalar Intensity Rating of Universal Facial Expression of Emotion", *Motivation and Emotion*, vol. 22, no. 4, Springer Netherlands, December 1998.
71. Zhan, Y. Z., Ye, J. F., Niu, D. J. and Cao, P., "Facial expression recognition based on Gabor wavelet transformation and elastic templates matching," *Image and Graphics*, 2004. Proceedings. Third International Conference on , pp. 254-257, 18-20 Dec. 2004

Appendix A - MATLAB Code

Filename : integrate.m

This is the code that controls the main window in the graphical user interface.

```
function varargout = integrate(varargin)
% INTEGRATE M-file for integrate.fig
%   INTEGRATE, by itself, creates a new INTEGRATE or raises the existing
%   singleton*.
%
%   H = INTEGRATE returns the handle to a new INTEGRATE or the handle to
%   the existing singleton*.
%
%   INTEGRATE('CALLBACK',hObject,eventData,handles,...) calls the local
%   function named CALLBACK in INTEGRATE.M with the given input arguments.
%
%   INTEGRATE('Property','Value',...) creates a new INTEGRATE or raises the
%   existing singleton*. Starting from the left, property value pairs are
%   applied to the GUI before integrate_OpeningFunction gets called. An
%   unrecognized property name or invalid value makes property application
%   stop. All inputs are passed to integrate_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
%   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help integrate

% Last Modified by GUIDE v2.5 31-Mar-2008 15:58:41

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @integrate_OpeningFcn, ...
                  'gui_OutputFcn',  @integrate_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
```

```

        'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if narginout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before integrate is made visible.
function integrate_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to integrate (see VARARGIN)

% Choose default command line output for integrate
handles.output = hObject;

handles.audioSet = [];
handles.videoSet = [];
handles.audio = [];
handles.video = [];
% Update handles structure
guidata(hObject, handles);

set(handles.recognizeEmo, 'Enable', 'off');

% UIWAIT makes integrate wait for user response (see UIRESUME)
% uiwait(handles.figure1);
axes(handles.audioSignal);
imshow('logo.jpg');
axes(handles.speechEmo);
imshow('logo.jpg');
axes(handles.faceEmo);
imshow('logo.jpg');
axes(handles.face);

```

```

imshow('logo.jpg');

% --- Outputs from this function are returned to the command line.
function varargout = integrate_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

function fileName_Callback(hObject, eventdata, handles)
% hObject handle to fileName (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of fileName as text
% str2double(get(hObject,'String')) returns contents of fileName as a
double
% global audio;
% global video;
% global audioSet;
fileName = get(hObject, 'String');
[handles.video handles.audio] = mmread(fileName);
% % sets audio setting button on if media file contains audio
set(handles.recognizeEmo, 'Enable', 'off');
set(handles.setAudio, 'Enable', 'off');
if isempty(handles.audio) == 0
    set(handles.setAudio, 'Enable', 'on');
    handles.audioSet = [];
% set(handles.recognizeEmo, 'Enable', 'on');
% handles.audioSet.freq = handles.audio.rate;
% handles.audioSet.pause = 8;
% handles.audioSet.silence = 0.1;
end
% % sets video setting button on if media file contains video
set(handles.setVideo, 'Enable', 'off');
if isempty(handles.video) == 0
    set(handles.setVideo, 'Enable', 'on');

```

```

        handles.videoSet = [];
        axes(handles.face);
        imshow(handles.video.frames(1).cdata)
    end
    guidata(hObject, handles);
% --- Executes during object creation, after setting all properties.
function fileName_CreateFcn(hObject, eventdata, handles)
% hObject    handle to fileName (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in setAudio.
function setAudio_Callback(hObject, eventdata, handles)
% hObject    handle to setAudio (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% global audio;
% global audioSet;
setAudio(handles.audio);
handles.audioSet = evalin('base', 'audioSet');
guidata(hObject, handles);
if ( isempty(handles.video) || isempty(handles.videoSet) == 0)
    set(handles.recognizeEmo, 'Enable', 'on');
end

% --- Executes on button press in setVideo.
function setVideo_Callback(hObject, eventdata, handles)
% hObject    handle to setVideo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
setVideo(handles.video);
handles.videoset = evalin('base', 'videoSet');
guidata(hObject, handles);
set(handles.recognizeEmo, 'Enable', 'on');

```

```

if ( isempty(handles.audio) || isempty(handles.audioSet) == 0)
    set(handles.recognizeEmo, 'Enable', 'on');
end

% --- Executes on button press in browse.
function browse_Callback(hObject, eventdata, handles)
% hObject    handle to browse (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[FN PN] = uigetfile('*..*', 'Select Media File');
set(handles.fileName, 'String', strcat(PN,FN));
fileName_Callback(handles.fileName, [], handles);

% --- Executes on button press in recognizeEmo.
function recognizeEmo_Callback(hObject, eventdata, handles)
% hObject    handle to recognizeEmo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.audioSignal);
imshow('logo.jpg');
axes(handles.speechEmo);
imshow('logo.jpg');
set(handles.audioLeg, 'Visible', 'off');
axes(handles.faceEmo);
imshow('logo.jpg');
set(handles.videoLeg, 'Visible', 'off');
axes(handles.face);
imshow('logo.jpg');

if strcmp(get(handles.setAudio, 'Enable'), 'on')
try
    handles.audioSet = evalin('base', 'audioSet');
catch
%     Just to eliminate a crash in case of an error.
end

% Plotting word separation
axes(handles.audioSignal);
plot((0:length(handles.audioSet.features.fullSpec)-
1)*handles.audioSet.totTime/(length(handles.audioSet.features.fullSpec)-1),
handles.audioSet.features.fullSpec, 'r', 'LineWidth', 2);

```

```

        hold on;
        plot(handles.audioSignal, (0:length(handles.audioSet.features.word)-
1)*handles.audioSet.totTime/(length(handles.audioSet.features.word)-1),
handles.audioSet.features.word, 'g', 'LineWidth', 2);
        axis([0 handles.audioSet.totTime -Inf
max(handles.audioSet.features.fullSpec)*1.1]);
        hold off;

if(size(handles.audioSet.features.F,1) < 166)
    errordlg('Words are too short for classification', 'Feature Length
Error')
    axes(handles.speechEmo), title('Words are too short to be classified');
elseif (size(handles.audioSet.features.F,1) > 166)
    handles.audioSet.features.F = handles.audioSet.features.F(1:166,:);
end

if(size(handles.audioSet.features.F,1) == 166)
    % Doing Emotion recogniton in speech
    load('VEnet.mat');
    ANNvoice = network2;
    netOP = sim(ANNvoice, handles.audioSet.features.F);

    SpeechEmo.Emo = [];
    M = max(netOP);
    for i = 1:size(netOP,2)
        SpeechEmo.Emo = [SpeechEmo.Emo, find(netOP(:,i) == M(1,i), 1)];
    end

    SpeechEmo.time = handles.audioSet.features.time;

    A = [SpeechEmo.Emo; SpeechEmo.Emo];

    axes(handles.speechEmo)
    for i = 1:length(SpeechEmo.Emo)
        plot(SpeechEmo.time(:,i), A(:,i), 'LineWidth', 5);
        axis([0 length(handles.audio.data)/handles.audioSet.freq 0 8]);
        title('Emotion in Speech');
        hold on;
    end
    grid on;
    hold off;
    set(handles.audioLeg, 'Visible', 'on');

```

```

        end
    end

    if strcmp(get(handles.setVideo, 'Enable'), 'on')
    % % Video emotion recognition
        try
            handles.videoSet = evalin('base', 'videoSet');
        catch
            % just to eliminate a crash
        end

        if (isempty(handles.videoSet) == 0 &&
            isempty(handles.videoSet.featurePoints) == 0)
            load('FEnet.mat');
            FEnet = network1;
            axes(handles.face);
            imshow(handles.videoSet.neutralImage);
            dist_Vec_Ne =
            distanceDirVector1(handles.videoSet.featurePoints{handles.videoSet.neutral});
            window = 5;
            A = [];
            op = [];
            for i = 1:handles.video.nframesTotal
                dist_Vec = distanceDirVector1(handles.videoSet.featurePoints{i});
                dist_Change_Vec = distanceDiffVector(dist_Vec_Ne, dist_Vec);
                A = [A dist_Change_Vec];
                if (i > window)
                    A(:,1) = [];
                end
                ip = mean(A,2);
                op = [op sim(FEnet, ip)];
            end

            timeScale = (1:length(op))*handles.video.times(end)/length(op);
            axes(handles.faceEmo);
            for i = 1:6
                plot(timeScale, op(i,:)+i, 'LineWidth', 1.5);
                drawnow;
                axis([0 handles.video.times(end) 0 8]);
                hold all;
            end
        end
    end

```



```
        grid on;
        hold off;
        set(handles.videoLeg, 'Visible', 'on');
    end
end
guidata(hObject, handles);
```

Filename : mmread.m

This is used to read the media file. It reads the file and stores data from the audio and video channel separately into data structures in the Matlab workspace. This is written by Micah Richert.

URL: <http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=8028>

```
function [video, audio] = mmread(filename, frames, time, disableVideo,
disableAudio, matlabCommand, trySeeking)
% function [video, audio] = mmread(filename, frames, time, disableVideo,
disableAudio, matlabCommand)
% mmread reads virtually any media file.  If Windows Media Play can play
% it, so should mmread, this includes URLs.  It uses the Window's DirectX
% infrastructure to render the media, so other OSs are out of luck.
%
% INPUT
% filename      input file to read (mpg, avi, wmv, asf, wav, mp3, gif, jpg,
...)
% frames        specifies which video frames to capture, default [] for all or
%               to specify time
% time          [startTime stopTime], default [] for all
% disableVideo  disables ALL video capturing, to save memory or time
% disableAudio  disables ALL audio capturing, to save memory or time
% matlabCommand Do not return the video structure, but call the function
%               specified by matlabCommand.  The function definition must
%               match that of processFrame.m.  See processFrame.m for more
%               information.
% trySeeking    [true] set to false to disable this if when using time
%               ranges or frames, and between subsequent reads the data
%               doesn't match.
%
% OUTPUT
% video is a struct with the following fields:
%   width        width of the video frames
%   height       height of the video frames
%   rate         the frame rate of the video, if it can't be determined
%               it will be 1.
%   nrFramesTotal the total number of frames in the movie regardless of
%               how many were captured.  Unfortunately, this can not
%               always be determined.  If it is negative then it
```

```

%           is an estimate based upon the duration and rate
%           (normally accurate to within .1%).  It can be 0,
%           in which case it could not be determined at all.  If it
%           is a possitive number then it should always be accurate.
%   totalDuration  the total length of the video in seconds.
%   frames         a struct array with the following fields:
%       cdata      [height X width X 3] uint8 matricies
%       colormap   always empty
%   times         the corresponding time stamps for the frames (in
milliseconds)
%
% audio is a struct with the following fields:
%   nrChannels     the number of channels in the audio stream (1 or 2)
%   rate          sampling rate of the audio, ex. 44100.  If it can't be
%               determined then it will be 1.
%   bits          bit depth of the samples (8 or 16)
%   data          the real data of the whole audio stream.  This can be
%               played using wavplay.  If time ranges are specified,
%               the length of the data may not correspond to the total
%               time.  This normally happens with movies.  The issue is
%               that the start of the audio stream is generally counted
%               at the END of the first frame.  So, time is shifted by
%               1/framerate.
%   nrFramesTotal Audio comes in packets or frames when captured, the
%               division of the audio into frames may or may not make
%               sense.
%   totalDuration the total length of the audio in seconds.
%   frames        cell array of uint8s.  Probably not of great use.
%   times        the corresponding time stamps for the frames (in
milliseconds)
%
% If there is no video or audio stream the corresponding structure will be
% empty.
%
% Specifying frames does not effect audio capturing.  If you want only a
% subsection of the audio use the 3rd parameter "time".  Specifying time
% effects both audio and video.  Time is specified in seconds (subsecond
% resolution is supported with fractional numbers ex. 1.125), starting at 0.
% Time is defined as startTime (inclusive) to stopTime (exclusive), or
% using set notation [startTime stopTime).
%
% If there are multiple video or audio streams, then the structure will be

```

```

% of length > 1. For example: audio(1).data and audio(2).data.
%
% Images work, however the frames must be specified. For some reason
% DirectShow doesn't ever stop when "playing" an image. So to deal with
% this, I added support so that the processing stops once the last
% specified frame is captured instead of waiting until the media completes.
%
% EXAMPLES
% [video, audio] = mmread('chimes.wav'); % read whole wav file
% wavplay(audio.data,audio.rate);
%
% video = mmread('mymovie.mpg'); % read whole movie
% movie(video.frames);
%
% video = mmread('mymovie.mpg',1:10); %get only the first 10 frames
%
% video = mmread('mymovie.mpg',[],[0 3.5]); %read the first 3.5 seconds of the
video
%
% [video, audio] = mmread('chimes.wav',[],[0 0.25]); %read the first 0.25
seconds of the wav
% [video, audio] = mmread('chimes.wav',[],[0.25 0.5]); %read 0.25 to 0.5
seconds of the wav, there is no overlap with the previous example.
%
% video = mmread('mymovie.mpg',[],[],false,true); %read all frames, disable
audio
%
% % read a movie directly from a URL
% video = mmread('http://www.nature.com/neuro/journal/v9/n4/extref/nn1660-
S8.avi');
%
% mmread('mymovie.mpg',[],[],false,false,'processFrame'); %Use inline
processing for all frames in a movie using the function processFrame.m
%
% Written by Micah Richert
%
http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=8028&objectTy
pe=FILE

if nargin < 7
    trySeeking = true;
    if nargin < 6

```

```

matlabCommand = '';
if nargin < 5
    disableAudio = false;
    if nargin < 4
        disableVideo = false;
        if nargin < 3
            time = [];
            if nargin < 2
                frames = [];
            end
        end
    end
end
end
end

try
mexDDGrab('buildGraph',filename);
if (isempty(time))
    mexDDGrab('setFrames',frames);
else
    if (numel(time) ~= 2)
        error('time must be a vector of length 2: [startTime stopTime]');
    end
    mexDDGrab('setTime',time(1),time(2));
end
if (disableVideo)
    mexDDGrab('disableVideo');
end;
if (disableAudio | nargout < 2)
    mexDDGrab('disableAudio');
end;
mexDDGrab('setMatlabCommand',matlabCommand);

mexDDGrab('setTrySeeking',double(trySeeking));

try
    mexDDGrab('doCapture');
catch
    err = lasterror;
    if (~strcmp(err.identifier,'processFrame:STOP'))
        rethrow(err);
    end
end

```

```

        end
    end

    [nrVideoStreams, nrAudioStreams] = mexDDGrab('getCaptureInfo');

    video = struct('width',{},'height',{},'nrFramesTotal',{},'frames',{});
    audio =
    struct('nrChannels',{},'rate',{},'bits',{},'nrFramesTotal',{},'data',{},'frames',{});

    warned = false;

    % we can only get the video frames if we don't process a matlabCommand
    if strcmp(matlabCommand, '')
        % loop through getting all of the video data from each stream
        for i=1:nrVideoStreams
            [width, height, rate, nrFramesCaptured, nrFramesTotal,
totalDuration] = mexDDGrab('getVideoInfo',i-1);
            video(i).width = width;
            video(i).height = height;
            video(i).rate = rate;
            video(i).nrFramesTotal = nrFramesTotal;
            video(i).totalDuration = totalDuration;
            video(i).frames =
    struct('cdata',cell(1,nrFramesCaptured),'colormap',cell(1,nrFramesCaptured));

            if (nrFramesTotal > 0 && any(frames > nrFramesTotal))
                warning('mmread:general',['Frame(s) '
num2str(frames(frames>nrFramesTotal)) ' exceed the number of frames in the movie.']);
            end

            scanline = ceil(width*3/4)*4; % the scanline size must be a
multiple of 4.

            for f=1:nrFramesCaptured
                [data, time] = mexDDGrab('getVideoFrame',i-1,f-1);

                if (numel(data) ~= scanline*height)
                    if (numel(data) > 3*width*height)
                        if (~warned)
                            warning('mmread:general','dimensions do not match
data size. Guessing badly...');
                            warned = true;

```

```

        end
        scanline = width*3;
        data = data(1:3*width*height);
    else
        if (f == 1)
            error('dimensions do not match data size. Too
little data.');
```

```

        else
            warning(['dimensions do not match data size. Too
little data for ' num2str(f) 'th frame.']);
            continue;
        end
    end
end
end

% if there is any extra scanline data, remove it
data = reshape(data,scanline,height);
data = data(1:3*width,:);

% the data ordering is wrong for matlab images, so permute it
tmp = permute(reshape(data, 3, width, height),[3 2 1]);
% the images are also upside down and colors were backwards.
video(i).frames(f).cdata = tmp(end:-1:1, :, 3:-1:1);
video(i).times(f) = time;
end

% if frames are specified then make sure that the order is the same
if (~isempty(frames) && nrFramesCaptured > 0)
    [uniqueFrames, dummy, frameOrder] = unique(frames);
    if (length(uniqueFrames) > nrFramesCaptured)
        warning('mmread:general','Not all frames specified were
captured. Returning what was captured, but order may be different than specified.');
```

```

        remainingFrames =
frames(frames<=uniqueFrames(nrFramesCaptured));
        [dummy, dummy, frameOrder] = unique(remainingFrames);
    end
end

video(i).frames = video(i).frames(frameOrder);
video(i).times = video(i).times(frameOrder);
end
end
end
```

```

% loop through getting all of the audio data from each stream
for i=1:nrAudioStreams
    [nrChannels, rate, bits, nrFramesCaptured, nrFramesTotal, subtype,
totalDuration] = mexDDGrab('getAudioInfo',i-1);
    audio(i).nrChannels = nrChannels;
    audio(i).rate = rate;
    audio(i).bits = bits;
    audio(i).nrFramesTotal = nrFramesTotal;
    audio(i).totalDuration = totalDuration;
    audio(i).frames = cell(1,nrFramesCaptured);
    for f=1:nrFramesCaptured
        [data, time] = mexDDGrab('getAudioFrame',i-1,f-1);
        audio(i).frames{f} = data;
        audio(i).times(f) = time;
    end
    % combine the data across frames
    d = double(cat(1,audio(i).frames{:}));

    % rescale the data so that it is between -1.0 and 1.0
    if (subtype==0)
        %PCM formatted data...
        switch (bits)
            case {4, 8}
                d = (d-2^(bits-1))/2^(bits-1);
            case {16, 24, 32}
                d = d/2^(bits-1);
        end
    elseif (subtype==1)
        if (bits == 32)
            %IEEE FLOAT formatted data...
            if (max(d) > 1 | min(d) < -1)
                % there are two float formats one that is already -1 to 1
                % and the there is between -2^15 to 2^15
                d = d / 2^15;
            end
        else
            warning('Audio data format not recognized/supported, it
probably is going to be useless.');
```



```

        warning('Audio data format not recognized/supported, it probably is
going to be useless. ');
    end

    % reshape the data so that it is nrChannels x Samples. This should be
the same output as wavread.
    audio(i).data = reshape(d,nrChannels,length(d)/nrChannels)';
end

mexDDGrab('cleanUp');
catch
    err = lasterror;
    mexDDGrab('cleanUp');
    if strfind(err.message,'combination')
        disp('The ''No combination of intermediate filters could be found to
make the connection'' error');
        disp('means that no appropriate codec could be found. Mpg2 files seem
to be the worst. ');
        disp('Installing ffdshow (www.free-codecs.com/FFDShow\_download.htm)
often fixes this problem. ');
    end
    rethrow(err);
end
end

```

Filename : distanceDiffVector.m

It computes the vector difference between its inputs, Vec1 and Vec2.

```
function dist_change_vec = distanceDiffVector(Vec1, Vec2)

dist_change_vec = Vec2 - Vec1;
```

Filename : distanceDirVector1.m

This function builds the 44 dimensional feature mask from the coordinated of the seventeen feature points.

```
function [dist_vec] = distanceDirVector1(Vec)

% get all the feature points
right_eye = Vec(1,:);
right_eye_in_corner = Vec(2,:);
right_eye_out_corner = Vec(3,:);
right_eyebrow = Vec(4,:);
right_eyebrow_corner = Vec(5,:);
left_eye = Vec(6,:);
left_eye_in_corner = Vec(7,:);
left_eye_out_corner = Vec(8,:);
left_eyebrow = Vec(9,:);
left_eyebrow_corner = Vec(10,:);
mouth_corner_left = Vec(11,:);
mouth_corner_right = Vec(12,:);
mouth_lip_upper = Vec(13,:);
mouth_lip_lower = Vec(14,:);
nose = Vec(15,:);
nose_left(1:2) = Vec(16,:);
nose_right(1:2) = Vec(17,:);

%normalizing value - distance between left and right eye outer corners
normalize = distance(left_eye_out_corner, right_eye_out_corner);

% without normalizing
V_mouth_width = distance(mouth_corner_left, mouth_corner_right);
```

```

V_mouth_height = distance(mouth_lip_upper, mouth_lip_lower);

V_mouth_nose_left = distance(mouth_corner_left, nose);
V_mouth_nose_right = distance(mouth_corner_right, nose);

V_lip_nose = distance(mouth_lip_upper, nose);

V_eyebrow_eye_inner_left = distance(left_eye_in_corner, left_eyebrow_corner);
V_eyebrow_eye_inner_right = distance(right_eye_in_corner,
right_eyebrow_corner);
V_eyebrow_eye_left = distance(left_eye, left_eyebrow);
V_eyebrow_eye_right = distance(right_eye, right_eyebrow);

V_eye_angle_left = angle(left_eye_out_corner, left_eye, left_eye_in_corner);
V_eye_angle_right = angle(right_eye_out_corner, right_eye,
right_eye_in_corner);

V_eyebrow_corners = distance(left_eyebrow_corner, right_eyebrow_corner);
V_eye_corners = distance(left_eye_in_corner, right_eye_in_corner);

V_eyebrow_angle_left = angle2points(left_eyebrow, left_eyebrow_corner, 1);
V_eyebrow_angle_right = angle2points(right_eyebrow, right_eyebrow_corner, 2);

V_mouth_eyes_left = distance(mouth_corner_left, left_eye);
V_mouth_eyes_right = distance(mouth_corner_right, right_eye);

V_mouth_lip_lower_left = distance(mouth_corner_left, mouth_lip_lower);
V_mouth_lip_lower_right = distance(mouth_corner_right, mouth_lip_lower);
V_mouth_lip_upper_left = distance(mouth_corner_left, mouth_lip_upper);
V_mouth_lip_upper_right = distance(mouth_corner_right, mouth_lip_upper);

V_nose_left = distance(nose_left, nose);
V_nose_right = distance(nose_right, nose);

V_nose_eye_left = distance(nose, left_eye);
V_nose_eye_right = distance(nose, right_eye);

V_mouth_eye_corner_left = distance(mouth_corner_left, left_eye_out_corner);
V_mouth_eye_corner_right = distance(mouth_corner_right, right_eye_out_corner);

V_eye_out_eyebrow_left = distance(left_eye_out_corner, left_eyebrow);
V_eye_out_eyebrow_right = distance(right_eye_out_corner, right_eyebrow);

```

```

    V_mouth_angle_lower = angle(mouth_corner_left, mouth_lip_lower,
mouth_corner_right);
    V_mouth_angle_upper = angle(mouth_corner_right, mouth_lip_upper,
mouth_corner_left);

    V_nose_angle = angle(mouth_corner_left, nose, mouth_corner_right);

    V_mouth_left_nose_vec = (nose - mouth_corner_left)';
    V_mouth_right_nose_vec = (nose - mouth_corner_right)';

    V_eyebrow_left_vec = (left_eyebrow_corner - left_eyebrow)';
    V_eyebrow_right_vec = (right_eyebrow_corner - right_eyebrow)';

    V_mouth_eyes_left_vec = (left_eye - mouth_corner_left)';
    V_mouth_eyes_right_vec = (right_eye - mouth_corner_right)';

    % vector containing all the vectors descibing the face
    dist_vec = [V_mouth_width; V_mouth_height; V_mouth_nose_left;
V_mouth_nose_right; V_lip_nose; V_eyebrow_eye_inner_left; V_eyebrow_eye_inner_right;
V_eyebrow_eye_left; V_eyebrow_eye_right; V_eye_angle_left; V_eye_angle_right;
V_eyebrow_corners; V_eye_corners; V_eyebrow_angle_left; V_eyebrow_angle_right;
V_mouth_eyes_left; V_mouth_eyes_right; V_mouth_lip_lower_left;
V_mouth_lip_lower_right; V_mouth_lip_upper_left; V_mouth_lip_upper_right; V_nose_left;
V_nose_right; V_nose_eye_left; V_nose_eye_right; V_mouth_eye_corner_left;
V_mouth_eye_corner_right; V_eye_out_eyebrow_left; V_eye_out_eyebrow_right;
V_mouth_angle_lower; V_mouth_angle_upper; V_nose_angle; V_mouth_left_nose_vec;
V_mouth_right_nose_vec; V_eyebrow_left_vec; V_eyebrow_right_vec;
V_mouth_eyes_left_vec; V_mouth_eyes_right_vec];
    %////////////////////////////////////

    function a = distance(point_a, point_b)
    % calculates the distance between two points in 2D

    a = sqrt(power((point_a(1,1) - point_b(1,1)),2) + power((point_a(1,2) -
point_b(1,2)),2));
    %////////////////////////////////////

    function a = angle(point_a, point_b, point_c)
    % finds the angle created by 3 points

    Pi = 4*atan(1);

```

```

vec1 = [(point_a(1,1) - point_b(1,1)) (point_a(1,2) - point_b(1,2))];
vec2 = [(point_c(1,1) - point_b(1,1)) (point_c(1,2) - point_b(1,2))];
scalar = [vec1(1) * vec2(1) + vec1(2) * vec2(2)];
length_vec1 = sqrt(power(vec1(1),2) + power(vec1(2),2));
length_vec2 = sqrt(power(vec2(1),2) + power(vec2(2),2));

a = acos(scalar/(length_vec1 * length_vec2));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function a = angle2points(point_a, point_b, side)

Pi = 4*atan(1);

vec1 = [(point_a(1,1) - point_b(1,1)) (point_a(1,2) - point_b(1,2))];
if side == 1
    vec2 = [0 (-10)];
else
    vec2 = [0 10];
end
scalar = [vec1(1) * vec2(1) + vec1(2) * vec2(2)];
length_vec1 = sqrt(power(vec1(1),2) + power(vec1(2),2));
length_vec2 = sqrt(power(vec2(1),2) + power(vec2(2),2));

a = acos(scalar/(length_vec1 * length_vec2));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

Filename : setAudio.m

This controls the audio toolbox that pops up upon clicking on the “Audio Setting” button. It separates words and extracts features based on the setting feed by the user. It then passes this data back to *integrate.m*.

```
function varargout = setAudio(varargin)
% SETAUDIO M-file for setAudio.fig
%   SETAUDIO, by itself, creates a new SETAUDIO or raises the existing
%   singleton*.
%
%   H = SETAUDIO returns the handle to a new SETAUDIO or the handle to
%   the existing singleton*.
%
%   SETAUDIO('CALLBACK',hObject,eventData,handles,...) calls the local
%   function named CALLBACK in SETAUDIO.M with the given input arguments.
%
%   SETAUDIO('Property','Value',...) creates a new SETAUDIO or raises the
%   existing singleton*. Starting from the left, property value pairs are
%   applied to the GUI before setAudio_OpeningFunction gets called. An
%   unrecognized property name or invalid value makes property application
%   stop. All inputs are passed to setAudio_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
%   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help setAudio

% Last Modified by GUIDE v2.5 16-Jan-2008 12:47:51

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @setAudio_OpeningFcn, ...
                  'gui_OutputFcn',  @setAudio_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',    []);
```

```

if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargin
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before setAudio is made visible.
function setAudio_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to setAudio (see VARARGIN)

% Choose default command line output for setAudio
    handles.output = hObject;

% UIWAIT makes setAudio wait for user response (see UIRESUME)
% uiwait(handles.figure1);

    handles.okclicked = false;
    handles.data = mean(varargin{1}.data,2);
    handles.totTime = varargin{1}.totalDuration;

    handles.dFreq = varargin{1}.rate;
    handles.freq = handles.dFreq;
    dFreqButton_Callback(handles.dFreqButton, [], handles)

    handles.dSilence = 0.1;
    handles.silence = handles.dSilence;
    dSilenceButton_Callback(handles.dSilenceButton, [], handles)

    handles.dPause = 8;
    handles.pause = handles.dPause;
    dPauseButton_Callback(handles.dPauseButton, [], handles)

```

```

% Update handles structure
    guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
function varargout = setAudio_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

    op.freq = handles.dFreq;
    op.silence = handles.dSilence;
    op.pause = handles.dPause;
    op.features = [];
    op.totTime = handles.totTime;

    assignin('base', 'audioSet', op);

% Get default command line output from handles structure
    varargout{1} = handles.output;

function eFreq_Callback(hObject, eventdata, handles)
% hObject    handle to eFreq (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of eFreq as text
%        str2double(get(hObject,'String')) returns contents of eFreq as a
double

    handles.freq = str2double(get(hObject,'String'));
    guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function eFreq_CreateFcn(hObject, eventdata, handles)
% hObject    handle to eFreq (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.

```



```

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function eSilence_Callback(hObject, eventdata, handles)
% hObject    handle to eSilence (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of eSilence as text
%       str2double(get(hObject,'String')) returns contents of eSilence as a
double

    handles.silence = str2double(get(hObject,'String'));
    guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function eSilence_CreateFcn(hObject, eventdata, handles)
% hObject    handle to eSilence (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function ePause_Callback(hObject, eventdata, handles)
% hObject    handle to ePause (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ePause as text

```

```

%         str2double(get(hObject,'String')) returns contents of ePause as a
double

        handles.pause = str2double(get(hObject,'String'));
        guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function ePause_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ePause (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in dFreqButton.
function dFreqButton_Callback(hObject, eventdata, handles)
% hObject    handle to dFreqButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

    set(handles.eFreq, 'String', num2str(handles.dFreq));
    eFreq_Callback(handles.eFreq, [], handles);

% --- Executes on button press in dSilenceButton.
function dSilenceButton_Callback(hObject, eventdata, handles)
% hObject    handle to dSilenceButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

    set(handles.eSilence, 'String', num2str(handles.dSilence));
    eSilence_Callback(handles.eSilence, [], handles);

% --- Executes on button press in dPauseButton.
function dPauseButton_Callback(hObject, eventdata, handles)
% hObject    handle to dPauseButton (see GCBO)

```

```

% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

    set(handles.ePause, 'String', num2str(handles.dPause));
    ePause_Callback(handles.ePause, [], handles);

% --- Executes on button press in testButton.
function testButton_Callback(hObject, eventdata, handles)
% hObject handle to testButton (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
    handles.features = featureExtraction(handles.data, handles.freq,
handles.silence, handles.pause);
    axes(handles.audioSignal);
    plot((0:length(handles.data)-1)*handles.totTime/(length(handles.data)-1),
mean(handles.data,2), 'b');
    axes(handles.audioWord);
    plot((0:length(handles.features.fullSpec)-
1)*handles.totTime/(length(handles.features.fullSpec)-1), handles.features.fullSpec,
'r', 'LineWidth', 2);
    hold on;
    plot((0:length(handles.features.word)-
1)*handles.totTime/(length(handles.features.word)-1), handles.features.word, 'g',
'LineWidth', 2);
    hold off;
    guidata(hObject,handles);

% --- Executes on button press in okButton.
function okButton_Callback(hObject, eventdata, handles)
% hObject handle to okButton (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

    op.freq = handles.freq;
    op.silence = handles.silence;
    op.pause = handles.pause;
    op.features = handles.features;
    op.totTime = handles.totTime;
    assignin('base', 'audioSet', op);
    close(handles.figure1);

```

Filename : featureExtraction.m

This function extracts the features from a segment of word. It takes the ra signal from a word segment as its input and extracts formants, mel-energy, their rate of change, and their mean and median.

```
% Parameter extraction for voice emotion recogniton
% Srivardhan C, Kansas State University, Sept 2007.
% Input audio data and frequency. To be used while emotion recogniton
function [features] = featureExtraction(y, Fs, silence, pause)

% Resampling at 8000Hz
y = resample(y, 8000, Fs);
Fs = 8000;
tic

% Computing spectrum for energy levels.
fullSpec = zeros(1, size(y));
overlapEnd = floor((Fs-256)/(Fs/100));
nFFT = 256;
for i = 1 : Fs/2: size(y)-Fs
    spectrum = computeSpectrum(nFFT, Fs/100, y(i:i+Fs));
    if(i==1)
        fullSpec = spectrum.e;
    end
    fullSpec = [fullSpec spectrum.e(overlapEnd-49:overlapEnd)];
end

% Word separation if energy level is less than a preset limit for more than
0.08 sec
silenceLevel = silence*max(fullSpec); %Setting silence at 5% of Peak
wordPause = pause; % 30 milliseconds is the pause that seperates 2 words

[wordIndex, word] = wordSeperation(fullSpec, wordPause, silenceLevel);

% Index for words on original Signal
wordIndexOriginal = round(wordIndex .* length(y)/length(fullSpec));
features.word = word;
features.fullSpec = fullSpec;
```

```

% Linear Predictive Coding vocal tract filter for each word
AvgEnergyWord = zeros(1,length(wordIndex)/2);
PeakEnergyWord = zeros(1,length(wordIndex)/2);
j = 0;
voiced = 0;
totPeaks = 0;
[melMat fCenters] = melFilterMatrix(Fs,256,22);
for i = 1:length(wordIndexOriginal)/2
    sample = y(wordIndexOriginal(2*i-1):wordIndexOriginal(2*i));
    voiced = voiced + (wordIndexOriginal(2*i) - wordIndexOriginal(2*i-1));
    totPeaks = totPeaks + length(Peak(fullSpec(wordIndex(2*i-
1):wordIndex(2*i))));
    if(length(sample) > 600)
        j = j+1;
        Formant(j) = computeFormant(sample, Fs, 4, melMat, fCenters);
        AvgEnergyWord(j) = mean(fullSpec(wordIndex(2*i-1):wordIndex(2*i)));
        PeakEnergyWord(j) = max(fullSpec(wordIndex(2*i-1):wordIndex(2*i)));
        timeStamp(:,j) = [wordIndexOriginal(2*i-1) wordIndexOriginal(2*i)]' /
Fs;
    end
end

%% %% %% %% %% %% %% %% %% %% %% %% %% %% %% %%
%% UTTERANCE LEVEL PARAMETERS %%
%% %% %% %% %% %% %% %% %% %% %% %% %% %% %% %%

% Calculating Words Spoken Per Sec
WordPerSec = (length(wordIndex)/2)/(length(y)/Fs); % Words Per Sec

% Calculating ratio of voiced to unvoiced speech
unvoiced = (length(y) - voiced)/length(y);
voiced = voiced/length(y);

% Average Energy Over Utterance
AvgEnergyUtterance = mean(AvgEnergyWord);

% Building Feature Vector
features.F = []; features.time = [];
for i = 1:j
    try
        if(sum(isnan(Formant(i).features)) == 0)

```

```
        features.F = [features.F [Formant(i).features; unvoiced; voiced;
AvgEnergyUtterance; WordPerSec]];
        features.time = [features.time timeStamp(:,i)];
    end
catch
end
end
toc
```

Filename : computeSpectrum.m

This function computes the spectrum using fast Fourier transform, and also computes the energy levels as a sum of energies at various frequency bands.

URL : <http://www.speech-recognition.de/matlab-examples.html> (05/12/2008)

```
function SPEC = computeSpectrum (fftLength,winShift,s)
% -----
% compute spectrum from time signal
%
% Returns power spectrum (|X(f)|^2) in
% matrix SPEC.X(coefficientIndex,frameIndex) .
% No energy normalization is performed.
% The signal energy (sum of power spectrum coefficients)
% is returned in vector SPEC.e(frameIndex)
%
% parameters:
% fftLength: length of FFT
% winShift: window shift [number of samples]
% s: vector of time samples
%
% last update 18.1.04
% http://www.speech-recognition.de/matlab-examples.html
% modified Srivardhan, Kansas State University
% -----

% compute local variables
nofSamples = size(s);
maxFFTIIdx = fftLength/2;

% compute time window
win = hamming(fftLength);

% compute matrix X(fftIndex,timeFrameIndex) short term spectra

k = 1;
for m = 1:winShift:nofSamples-fftLength

    spec = fft( (win.*s(m:m+fftLength-1)) ,fftLength);
    %use only lower half of fft coefficients
```

```
    SPEC.X(:,k) = ( abs( spec(1:maxFFTIIdx) ) ).^2;
    %compute energy
    SPEC.e(k) = sum(SPEC.X(:,k));

    k = k+1;
end
```


Filename : wordSeperation.m

This function separates words in the signal using the energy levels calculated by *computeSpectrum.m* and the word pause and threshold setting from the audio toolbox.

```
function [wordIndex, word] = wordSeperation(Spec, Pause, Silence)

% Srivardhan C, Kansas State University
% Inputs are
%     Spec = energy spectrum
%     Pause = Time in milliSec that seperates 2 words
%     Silence = The energy level below which it is considered unvoiced
% Outputs are
%     wordIndex = index of begning and ending of each segment of voiced
%                 data in the spectrum vector. [begin1 end1 begin2 ...]
%     word = vector the same size as spectrum, with 0s for unvoiced
%            segment and 1s for voiced segment

SpecNorm = zeros(1, size(Spec,2));
SpecNorm(Spec > Silence) = 1;

word = zeros(1,size(SpecNorm,2));
i = find(SpecNorm, 1);
wordIndex = i;
while i <= length(SpecNorm)-Pause
    if sum(SpecNorm(i : i + Pause)) < 0.075*Pause
        wordIndex = [wordIndex , i + 1];
        i = find(SpecNorm(i+1 : end), 1) + i;
        wordIndex = [wordIndex , i];
    else
        word(i) = 1;
        i = i + 1;
    end
end

if wordIndex(end) ~= (find(SpecNorm,1,'last')+1)
    wordIndex = [wordIndex, find(SpecNorm,1,'last')];
    word(wordIndex(end-1):wordIndex(end)) = 1;
end

word= word * max(Spec);
```

Filename : melfiltermatrix.m

This function computes the mel frequency filter matrix that is used to convert the spectrogram into a mel spectrum. The center frequencies of the channels follow the mel scale, while the frequency distribution is linear. The bandwidth increases with frequencies.

URL : <http://www.speech-recognition.de/matlab-examples.html> (05/12/2008)

```
function [W, fcenters] = melFilterMatrix(fs, N, nofChannels)
% -----
% melFilterMatrix(fs, N, nofChannels):
% compute mel filter coefficients
%
% returns: Matrix (channelIndex, FFTIndex)
% of mel filter coefficients.
%
% parameters:
% fs: Sampling rate [Hz], eg., 8000
% N: FFT length, eg., 256
% nofChannels: Number of mel channels, eg., 22
%
% last update: 13.1.04
% http://www.speech-recognition.de/matlab-examples.html
% -----

%for test, use these parameters
%parameters
%fs = 8000;
%N = 256;
%nofChannels = 22;

%compute resolution etc.
df = fs/N; %frequency resolution
Nmax = N/2; %Nyquist frequency index
fmax = fs/2; %Nyquist frequency
melmax = freq2mel(fmax); %maximum mel frequency

%mel frequency increment generating 'nofChannels' filters
melinc = melmax / (nofChannels + 1);

%vector of center frequencies on mel scale
melcenters = (1:nofChannels) .* melinc;
```

```

%vector of center frequencies [Hz]
fcenters = mel2freq(melcenters);

%compute bandwidths
%startfreq = [0 , fcenters(1:(nofChannels-1))];
%endfreq = [fcenters(2:nofChannels) , fmax];
%bandwidth = endfreq - startfreq ;

%quantize into FFT indices
indexcenter = round(fcenters ./df);

%compute resulting frequencies
%fftfreq = indexcenter.*df;

%compute resulting error
%diff = fcenters - fftfreq;

%compute startfrequency, stopfrequency and bandwidth in indices
indexstart = [1 , indexcenter(1:nofChannels-1)];
indexstop = [indexcenter(2:nofChannels),Nmax];
%idxbw = (indexstop - indexstart)+1;
%FFTbandwidth = idxbw.*df;

%compute matrix of triangle-shaped filter coefficients
W = zeros(nofChannels,Nmax);
for c = 1:nofChannels
    %left ramp
    increment = 1.0/(indexcenter(c) - indexstart(c));
    for i = indexstart(c):indexcenter(c)
        W(c,i) = (i - indexstart(c))*increment;
    end %i
    %right ramp
    decrement = 1.0/(indexstop(c) - indexcenter(c));
    for i = indexcenter(c):indexstop(c)
        W(c,i) = 1.0 - ((i - indexcenter(c))*decrement);
    end %i
end %c

%normalize melfilter matrix
for j = 1:nofChannels

```

```
W(j,:) = W(j,:)/ sum(W(j,:)) ;  
end
```

Filename : freq2mel.m

This converts the from the hertz frequency scale to mel scale

URL: <http://www.speech-recognition.de/matlab-examples.html> (05/12/2008)

```
function m = freq2mel (f)
% compute mel value from frequency f
% http://www.speech-recognition.de/matlab-examples.html

m = 2595 * log10(1 + f./700);
```

Filename : mel2freq.m

This converts from mel scale to hertz frequency scale.

URL: <http://www.speech-recognition.de/matlab-examples.html> (05/12/2008)

```
function f = mel2freq (m)
% compute frequency from mel value
% http://www.speech-recognition.de/matlab-examples.html

f = 700*((10.^(m ./2595)) -1);
```

Filename : Peak.m

This function picks peaks by comparing the sample with its previous and next sample. If the previous sample is less than or equal to the sample and the next sample is less then it is recognized as a peak.

```
function [Peak] = peak(ip)
% Srivardhan C, Kansas State University, 2007
% this function returns indices of local maxima in vector
% ip has to be a vector and not a matrix

Peak = [];
for k = 2:length(ip)-1
    if( ip(k-1) <= ip(k) && ip(k+1) < ip(k) )
        Peak = [Peak, k];
    end
end
end
```

Filename : computeFormant.m

This compute the formants and mel energy along the given segment of signal. Formant extraction is done using the linear predictive coding.

```
function [Formant] = computeFormant(y, Fs, num, melMat, fCenters)
% Srivardhan C, Kansas State University, 2007
% Computes Formant frequencies, median value for formant freq
% Inputs - 1. y = Voiced portion of Speech
%          2. Fs = Sampling Frequency
%          3. num = number of formants desired
%          4. melMat = Mel Matrix for computing mel Spectrum

% Sampling window length
N = Fs*32/1000;

if length(y) < N
    N = length(y)-1;
end

% Hamming Window
ham = hamming(N+1);

Step = Fs * 5/1000;

% Number of poles of filter
M = 5 + Fs/1000;

% Number of windows in given sample
noWin = (length(y)-N);

Formant.freq = [];
Formant.Spec = [];
Formant.pitch = [];
for i = 1:Step:noWin
    try
%         Pitch Extraction using AutoCorrelation
%         http://www.phon.ucl.ac.uk/courses/spsci/matlab/lect10.html
        CoeffP = xcorr(y(i:i+N), 'coeff');
        pitchP = peak(CoeffP);
        Pitch = 8000/(pitchP(find(pitchP>257,1,'first'))-257);
```

```

Formant.pitch = [Formant.pitch, Pitch];

%      Method described in 'FORMANT EXTRACTION USING DIFFERENCE SPECTRUM',
%      S.P.Kishore et.al. ,
Coeff1 = lpc(y(i:i+N).*ham, M);
Coeff2 = lpc(y(i:i+N).*ham, M+1);

[h1,f]=freqz(1,Coeff1,512,Fs);
[h2,f]=freqz(1,Coeff2,512,Fs);

Gain1 = 20*log10(abs(h1)+eps);
Gain2 = 20*log10(abs(h2)+eps);
GainDiff = Gain2-Gain1;
P = peak(GainDiff);
formant = zeros(M,1);
formant(1:length(P),1) = P' .* (Fs/(2*length(GainDiff))); % Convert
from index to Hz
Formant.freq = [Formant.freq , formant(1:num)]; % convert to Hz and
sort

[Spec,Freq,Time] = spectrogram(y(i:i+N),N,N-Step,formant(1:num),Fs);
Formant.Spec = [Formant.Spec, abs(Spec)];
catch
display('Error Computing Formant');
end
end

% Compute mel Spectrum
MEL = computeMelSpectrum(melMat,Step,y);

%normalize energy of mel spectra
%take log value
epsilon = 10e-5;
for k = 1:size(MEL.M,2);
for c = 1:size(MEL.M,1)
%normalize energy
MEL.M(c,k) = MEL.M(c,k)/MEL.e(k);
%take log energy
MEL.M(c,k) = loglimit(MEL.M(c,k),epsilon);
end %for c
end %for k

```



```

% Number of peaks in the segment
Formant.noPeaks = length(P);

% time of word segment
Formant.time = length(y)/Fs;

% Picking Mel band energies for formants
for i = 1:size(Formant.freq, 2)
    for j = 1:num
        % Absolute Diff between Center frequency and Formant
        absDiff = abs(fCenters - Formant.freq(j,i));
        % Index of min difference
        MelInd = find (absDiff == min(absDiff));
        % Energy from mel band for formant freq
        Formant.MEL.M(j,i) = MEL.M(MelInd, i);
    end
end

Formant.diffFreq = [Formant.freq(:,2:end) - Formant.freq(:,1:end-1)
zeros(num,1) ];
Formant.diffMel = [Formant.MEL.M(:,2:end) - Formant.MEL.M(:,1:end-1)
zeros(num,1) ];

% Dividing a word into 5 parts
s = floor((size(Formant.freq,2))/5);
Formant.features = [];
for l = 1:s:(size(Formant.freq,2)-s)
    A = [mean(Formant.freq(:,l:l+s),2); ...
        median(Formant.freq(:,l:l+s),2); ...
        mean(Formant.MEL.M(:,l:l+s),2); ...
        median(Formant.MEL.M(:,l:l+s),2); ...
        mean(Formant.diffFreq(:,l:l+s),2); ...
        median(Formant.diffFreq(:,l:l+s),2); ...
        mean(Formant.diffMel(:,l:l+s),2); ...
        median(Formant.diffMel(:,l:l+s),2)];
    Formant.features = [Formant.features; A];
end
Formant.features = [Formant.features; Formant.time; Formant.noPeaks];

```

Filename : loglimit.m

It computes the log if the signal is above a threshold, or else returns the log of the lower limit.

URL: <http://www.speech-recognition.de/matlab-examples.html> (05/12/2008)

```
function y = loglimit(x,limit)
% return log(x) or log(limit) if x < limit
% http://www.speech-recognition.de/matlab-examples.html

if (x < limit)
    y = log(limit);
else
    y = log(x);
end;
```

Filename : computeMelSpectrum.m

This computes the mel spectrum by multiplying the spectrum obtained using *computeSpectrum.m* with the mel frequency filter matrix obtained from *melFilterMatrix.m*.

URL: <http://www.speech-recognition.de/matlab-examples.html> (05/12/2008)

```
function MEL = computeMelSpectrum (W,winShift,s)
% -----
% computeMelSpectrum (W,winShift,s)
% compute mel spectrum from time signal
%
% Returns mel spectral coefficients in
% matrix MEL.M(coefficientIndex,frameIndex).
% No energy normalization is performed.
% Signal energy
% is copied from SPEC.e
% ('computeSpectrum') to vector MEL.e(frameIndex).
%
% parameters:
% W: matrix(channelIndex,FFTIndex) of mel filter coefficients
% winShift: window shift [number of samples]
% s: vector of time samples
%
% last update 18.1.04
% http://www.speech-recognition.de/matlab-examples.html
% -----

% compute local variables
[nofChannels,maxFFTIdx] = size(W);
fftLength = maxFFTIdx * 2;

% compute matrix X(fftIndex,timeFrameIndex) short term spectra
SPEC = computeSpectrum(fftLength,winShift,s);

% apply mel filter to spectra

MEL.M = W * SPEC.X;

%copy energy vector
MEL.e = SPEC.e;
```

Filename : setVideo.m

Controls the video setting toolbox, and all its functions.

```
function varargout = setVideo(varargin)
% SETVIDEO M-file for setVideo.fig
%     SETVIDEO, by itself, creates a new SETVIDEO or raises the existing
%     singleton*.
%
%     H = SETVIDEO returns the handle to a new SETVIDEO or the handle to
%     the existing singleton*.
%
%     SETVIDEO('CALLBACK',hObject,eventData,handles,...) calls the local
%     function named CALLBACK in SETVIDEO.M with the given input arguments.
%
%     SETVIDEO('Property','Value',...) creates a new SETVIDEO or raises the
%     existing singleton*. Starting from the left, property value pairs are
%     applied to the GUI before setVideo_OpeningFunction gets called. An
%     unrecognized property name or invalid value makes property application
%     stop. All inputs are passed to setVideo_OpeningFcn via varargin.
%
%     *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
%     instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help setVideo

% Last Modified by GUIDE v2.5 23-Jan-2008 17:49:30

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @setVideo_OpeningFcn, ...
                  'gui_OutputFcn',  @setVideo_OutputFcn, ...
                  'gui_LayoutFcn',   [] , ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
```

```

if nargin
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before setVideo is made visible.
function setVideo_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to setVideo (see VARARGIN)

% Choose default command line output for setVideo
handles.output = hObject;

handles.neutral = 1;
handles.frameNo = 1;
handles.lastFrame = varargin{1}.nrFramesTotal;
for i = 1 :handles.lastFrame
    handles.frames{i} = rgb2gray(varargin{1}.frames(i).cdata);
    handles.originalFrames{i} = rgb2gray(varargin{1}.frames(i).cdata);
end
handles.time = varargin{1}.times;
handles.neutralImage = handles.frames{handles.neutral};

% updating thresholds
handles.thresh.background = get(handles.sBackground, 'Value');
handles.thresh.lighting = get(handles.sLighting, 'Value');
handles.thresh.eye = get(handles.sThreshEye, 'Value');
handles.thresh.eyebrow = get(handles.sThreshEyebrow, 'Value');
handles.thresh.lips = get(handles.sThreshLips, 'Value');
handles.thresh.lipcorner = get(handles.sThreshLipcorner, 'Value');
handles.thresh.nose = get(handles.sThreshNose, 'Value');
handles.thresh.nosetril = get(handles.sThreshNosetril, 'Value');
handles.thresh.eyeOut = get(handles.sThreshEyeOut, 'Value');
handles.thresh.eyeIn = get(handles.sThreshEyeIn, 'Value');
handles.thresh.ebCorner = get(handles.sThreshEBcorner, 'Value');

```

```

handles.pointsManual = 1;
handles.manualFace = 1;

% Update handles structure
guidata(hObject, handles);

% Setting Frame Panel
set(handles.sFrame, 'Max', handles.lastFrame);
set(handles.sFrame, 'Value', 1);
set(handles.sFrame, 'SliderStep', [0.99/handles.lastFrame, ...
    9.9/handles.lastFrame]);
set(handles.iframeNo, 'String', num2str(handles.frameNo));
set(handles.etime, 'String', num2str(handles.time(handles.frameNo)));
% Setting Crop Panel
resetCrop(handles)
% Showing the image
showFace(handles)
% Setting Height and Width values.
set(handles.eWidth, 'String',
num2str(size(handles.frames{handles.frameNo},2)));
set(handles.eHeight, 'String',
num2str(size(handles.frames{handles.frameNo},1)));
% Setting points selection to manual
rPointsManual_Callback(handles.rPointsManual, [], handles);
% Setting Face region selection to manual
rFaceManual_Callback(handles.rFaceManual, [], handles);
% Setting Rotate Panel
rotate0_Callback(handles.rotate0, [], handles);

% UIWAIT makes setVideo wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = setVideo_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure

```

```

varargout{1} = handles.output;

op.thresh = handles.thresh;
op.featurePoints = [];
op.neutral = handles.neutral;
op.neutralImage = handles.neutralImage;
assignin('base', 'videoSet', op);

% --- Executes on slider movement.
function sFrame_Callback(hObject, eventdata, handles)
% hObject    handle to sFrame (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
handles.frameNo = ceil(get(hObject,'Value'));
set(handles.iframeNo, 'String', num2str(handles.frameNo));
set(handles.etime, 'String', num2str(handles.time(handles.frameNo)));
% Showing the image
showFace(handles)
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function sFrame_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sFrame (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on button press in testFrameButton.
function testFrameButton_Callback(hObject, eventdata, handles)
% hObject    handle to testFrameButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.face);

```

```

handles.manualFace
if handles.manualFace
    imshow(handles.frames{handles.frameNo});
    tl = round(ginput(1));
    tr = round(ginput(1));
    bl = round(ginput(1));
    Rmin = tl(2);
    Rmax = bl(2);
    Cmin = tl(1);
    Cmax = tr(1);
    handles.faceCoord = [Rmin, Rmax, Cmin, Cmax];

else
    J = getFaceRegion(handles.frames{handles.frameNo},
handles.thresh.lighting, handles.thresh.background);
    L = bwlabel(J);
    stats=regionprops(L, 'BoundingBox');
    k=[stats.BoundingBox];
    stats=regionprops(L, 'Area');
    a = [stats.Area];
    display(a);
    %Separating was unsuccessfull if no region was found or if area of found
    %region is less than 1000 pixels
    if (isempty(a) || a < 1000)
        display('Unable to locate face region.');
```

```

        imshow(J);
    else
        Rmin=round(k(2));
        Rmax=round(k(2)+k(4));
        Cmin=round(k(1));
        Cmax=round(k(1)+k(3));
        handles.faceCoord = [Rmin, Rmax, Cmin, Cmax];
    end
end

end

%visualisation of the selected face region in the face Region subimage
ImagePoints = handles.frames{handles.frameNo};
ImagePoints(Rmin, Cmin:Cmax) = 256;
ImagePoints(Rmax, Cmin:Cmax) = 256;
ImagePoints(Rmin:Rmax, Cmin) = 256;
ImagePoints(Rmin:Rmax, Cmax) = 256;
imshow(ImagePoints);

```



```

if(handles.pointsManual)
    % manual input to points
    imshow(ImagePoints);

    % right eye
    in = round(ginput(1));
    feature_points(1,1) = in(2);
    feature_points(1,2) = in(1);
    ImagePoints(feature_points(1,1) - 5 : feature_points(1,1) + 5 ,
feature_points(1,2)) = 256;
    ImagePoints(feature_points(1,1), feature_points(1,2) - 5 :
feature_points(1,2) + 5) = 256;
    imshow(ImagePoints);

    %right eye inner corner
    in = round(ginput(1));
    feature_points(2,1) = in(2);
    feature_points(2,2) = in(1);
    ImagePoints(feature_points(2,1) - 5 : feature_points(2,1) + 5 ,
feature_points(2,2)) = 256;
    ImagePoints(feature_points(2,1), feature_points(2,2) - 5 :
feature_points(2,2) + 5) = 256;
    imshow(ImagePoints);

    %right eye outer corner
    in = round(ginput(1));
    feature_points(3,1) = in(2);
    feature_points(3,2) = in(1);
    ImagePoints(feature_points(3,1) - 5 : feature_points(3,1) + 5 ,
feature_points(3,2)) = 256;
    ImagePoints(feature_points(3,1), feature_points(3,2) - 5 :
feature_points(3,2) + 5) = 256;
    imshow(ImagePoints);

    %right eyebrow
    in = round(ginput(1));
    feature_points(4,1) = in(2);
    feature_points(4,2) = in(1);
    ImagePoints(feature_points(4,1) - 5 : feature_points(4,1) + 5 ,
feature_points(4,2)) = 256;

```

```

        ImagePoints(feature_points(4,1), feature_points(4,2) - 5 :
feature_points(4,2) + 5) = 256;
        imshow(ImagePoints);

        %right eyebrow corner
        in = round(ginput(1));
        feature_points(5,1) = in(2);
        feature_points(5,2) = in(1);
        ImagePoints(feature_points(5,1) - 5 : feature_points(5,1) + 5 ,
feature_points(5,2)) = 256;
        ImagePoints(feature_points(5,1), feature_points(5,2) - 5 :
feature_points(5,2) + 5) = 256;
        imshow(ImagePoints);

        %left eye
        in = round(ginput(1));
        feature_points(6,1) = in(2);
        feature_points(6,2) = in(1);
        ImagePoints(feature_points(6,1) - 5 : feature_points(6,1) + 5 ,
feature_points(6,2)) = 256;
        ImagePoints(feature_points(6,1), feature_points(6,2) - 5 :
feature_points(6,2) + 5) = 256;
        imshow(ImagePoints);

        %left eye inner corner
        in = round(ginput(1));
        feature_points(7,1) = in(2);
        feature_points(7,2) = in(1);
        ImagePoints(feature_points(7,1) - 5 : feature_points(7,1) + 5 ,
feature_points(7,2)) = 256;
        ImagePoints(feature_points(7,1), feature_points(7,2) - 5 :
feature_points(7,2) + 5) = 256;
        imshow(ImagePoints);

        %left eye outer corner
        in = round(ginput(1));
        feature_points(8,1) = in(2);
        feature_points(8,2) = in(1);
        ImagePoints(feature_points(8,1) - 5 : feature_points(8,1) + 5 ,
feature_points(8,2)) = 256;
        ImagePoints(feature_points(8,1), feature_points(8,2) - 5 :
feature_points(8,2) + 5) = 256;

```

```

imshow(ImagePoints);

%left eyebrow
in = round(ginput(1));
feature_points(9,1) = in(2);
feature_points(9,2) = in(1);
ImagePoints(feature_points(9,1) - 5 : feature_points(9,1) + 5 ,
feature_points(9,2)) = 256;
ImagePoints(feature_points(9,1), feature_points(9,2) - 5 :
feature_points(9,2) + 5) = 256;
imshow(ImagePoints);

%left eyebrow corner
in = round(ginput(1));
feature_points(10,1) = in(2);
feature_points(10,2) = in(1);
ImagePoints(feature_points(10,1) - 5 : feature_points(10,1) + 5 ,
feature_points(10,2)) = 256;
ImagePoints(feature_points(10,1), feature_points(10,2) - 5 :
feature_points(10,2) + 5) = 256;
imshow(ImagePoints);

%left mouth corner
in = round(ginput(1));
feature_points(11,1) = in(2);
feature_points(11,2) = in(1);
ImagePoints(feature_points(11,1) - 5 : feature_points(11,1) + 5 ,
feature_points(11,2)) = 256;
ImagePoints(feature_points(11,1), feature_points(11,2) - 5 :
feature_points(11,2) + 5) = 256;
imshow(ImagePoints);

%rightmouth corner
in = round(ginput(1));
feature_points(12,1) = in(2);
feature_points(12,2) = in(1);
ImagePoints(feature_points(12,1) - 5 : feature_points(12,1) + 5 ,
feature_points(12,2)) = 256;
ImagePoints(feature_points(12,1), feature_points(12,2) - 5 :
feature_points(12,2) + 5) = 256;
imshow(ImagePoints);

```

```

%upper lip
in = round(ginput(1));
feature_points(13,1) = in(2);
feature_points(13,2) = in(1);
ImagePoints(feature_points(13,1) - 5 : feature_points(13,1) + 5 ,
feature_points(13,2)) = 256;
ImagePoints(feature_points(13,1), feature_points(13,2) - 5 :
feature_points(13,2) + 5) = 256;
imshow(ImagePoints);

%lower lip
in = round(ginput(1));
feature_points(14,1) = in(2);
feature_points(14,2) = in(1);
ImagePoints(feature_points(14,1) - 5 : feature_points(14,1) + 5 ,
feature_points(14,2)) = 256;
ImagePoints(feature_points(14,1), feature_points(14,2) - 5 :
feature_points(14,2) + 5) = 256;
imshow(ImagePoints);

% nose
in = round(ginput(1));
feature_points(15,1) = in(2);
feature_points(15,2) = in(1);
ImagePoints(feature_points(15,1) - 5 : feature_points(15,1) + 5 ,
feature_points(15,2)) = 256;
ImagePoints(feature_points(15,1), feature_points(15,2) - 5 :
feature_points(15,2) + 5) = 256;
imshow(ImagePoints);

%nose corner left
in = round(ginput(1));
feature_points(16,1) = in(2);
feature_points(16,2) = in(1);
ImagePoints(feature_points(16,1) - 5 : feature_points(16,1) + 5 ,
feature_points(16,2)) = 256;
ImagePoints(feature_points(16,1), feature_points(16,2) - 5 :
feature_points(16,2) + 5) = 256;
imshow(ImagePoints);

%nose corner right
in = round(ginput(1));

```

```

feature_points(17,1) = in(2);
feature_points(17,2) = in(1);
ImagePoints(feature_points(17,1) - 5 : feature_points(17,1) + 5 ,
feature_points(17,2)) = 256;
ImagePoints(feature_points(17,1), feature_points(17,2) - 5 :
feature_points(17,2) + 5) = 256;
imshow(ImagePoints);

%visualize the selected points on the image
ImagePoints = handles.frames(handles.frameNo);

handles.points(handles.frameNo) = feature_points;

if(handles.frameNo == handles.neutral)
    handles.neutralImage = ImagePoints;
end
else
    [ImagePoints, handles.featurePoints(handles.frameNo)] =
testFrame(handles.frames(handles.frameNo), handles.faceCoord, handles.thresh);

    if(handles.frameNo == handles.neutral)
        handles.neutralImage = ImagePoints;
    end
%    handles.frames(handles.frameNo) = ImagePoints;
end
axes(handles.face), imshow(ImagePoints), drawnow;
set(handles.testVideoButton, 'Enable', 'on');
guidata(hObject,handles);

% --- Executes on button press in setNeutralButton.
function setNeutralButton_Callback(hObject, eventdata, handles)
% hObject    handle to setNeutralButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.neutral = ceil(get(handles.sFrame, 'Value'));
guidata(hObject, handles);

function eframeNo_Callback(hObject, eventdata, handles)
% hObject    handle to eframeNo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

% Hints: get(hObject,'String') returns contents of eframeNo as text
%         str2double(get(hObject,'String')) returns contents of eframeNo as a
double
handles.frameNo = ceil(str2double(get(hObject,'String')));
if (handles.frameNo < 1)
    handles.frameNo = 1;
    set(handles.iframeNo, 'String', num2str(handles.frameNo));
elseif (handles.frameNo > handles.lastFrame)
    handles.frameNo = handles.lastFrame;
    set(handles.iframeNo, 'String', num2str(handles.frameNo));
end
set(handles.sFrame, 'Value', handles.frameNo);
set(handles.etime, 'String', num2str(handles.time(handles.frameNo)));
% Showing the image
showFace(handles)
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function eframeNo_CreateFcn(hObject, eventdata, handles)
% hObject    handle to eframeNo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function etime_Callback(hObject, eventdata, handles)
% hObject    handle to etime (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of etime as text
%         str2double(get(hObject,'String')) returns contents of etime as a
double

```

```

% --- Executes during object creation, after setting all properties.
function etime_CreateFcn(hObject, eventdata, handles)
% hObject    handle to etime (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in cCrop.
function cCrop_Callback(hObject, eventdata, handles)
% hObject    handle to cCrop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of cCrop
if get(hObject,'Value')
    set(handles.eCropX, 'Enable', 'on');
    set(handles.eCropY, 'Enable', 'on');
    set(handles.eCropW, 'Enable', 'on');
    set(handles.cropButton, 'Enable', 'on');
else
    set(handles.eCropX, 'Enable', 'off');
    set(handles.eCropY, 'Enable', 'off');
    set(handles.eCropW, 'Enable', 'off');
    set(handles.cropButton, 'Enable', 'off');
end

function eCropX_Callback(hObject, eventdata, handles)
% hObject    handle to eCropX (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of eCropX as text

```

```

        %         str2double(get(hObject,'String')) returns contents of eCropX as a
double
        if(str2double(get(hObject,'String')) < 1)
            set(hObject, 'String', num2str(1));
        elseif
(str2double(get(hObject,'String'))+str2double(get(handles.eCropW,'String')) >
str2double(get(handles.eWidth,'String')))
            set(hObject, 'String', num2str(str2double(get(handles.eWidth,'String')) -
str2double(get(handles.eCropW,'String'))));
        end
        showFace(handles);

% --- Executes during object creation, after setting all properties.
function eCropX_CreateFcn(hObject, eventdata, handles)
% hObject    handle to eCropX (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function eCropY_Callback(hObject, eventdata, handles)
% hObject    handle to eCropY (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of eCropY as text
%         str2double(get(hObject,'String')) returns contents of eCropY as a
double
        if(str2double(get(hObject,'String')) < 1)
            set(hObject, 'String', num2str(1));
        elseif
(str2double(get(hObject,'String'))+str2double(get(handles.eCropW,'String')) >
str2double(get(handles.eHeight,'String')))
            set(hObject, 'String', num2str(str2double(get(handles.eHeight,'String')) -
str2double(get(handles.eCropW,'String'))));

```



```

end
showFace(handles);

% --- Executes during object creation, after setting all properties.
function eCropY_CreateFcn(hObject, eventdata, handles)
% hObject    handle to eCropY (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function eCropW_Callback(hObject, eventdata, handles)
% hObject    handle to eCropW (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of eCropW as text
%        str2double(get(hObject,'String')) returns contents of eCropW as a
double
    if(str2double(get(hObject,'String')) >=
(str2double(get(handles.eHeight,'String'))- str2double(get(handles.eCropY,'String')))
|| str2double(get(hObject,'String')) >= (str2double(get(handles.eWidth,'String'))-
str2double(get(handles.eCropX,'String'))))
        set(hObject,'String',
num2str(min((str2double(get(handles.eHeight,'String'))-
str2double(get(handles.eCropY,'String'))), (str2double(get(handles.eWidth,'String'))-
str2double(get(handles.eCropX,'String')))))));
    end
showFace(handles);

% --- Executes during object creation, after setting all properties.
function eCropW_CreateFcn(hObject, eventdata, handles)
% hObject    handle to eCropW (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

```

```

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in cropButton.
function cropButton_Callback(hObject, eventdata, handles)
% hObject    handle to cropButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
for i = 1:handles.lastFrame
    xCrop = str2double(get(handles.eCropX, 'String'));
    yCrop = str2double(get(handles.eCropY, 'String'));
    wCrop = str2double(get(handles.eCropW, 'String'));
    oHeight = str2double(get(handles.eHeight, 'String'));
    handles.frames{i} = handles.frames{i}((oHeight+1-yCrop-wCrop):(oHeight-
yCrop), xCrop:(xCrop+wCrop));
end
resetCrop(handles);
showFace(handles);
guidata(hObject, handles);

% --- Executes on button press in rotate0.
function rotate0_Callback(hObject, eventdata, handles)
% hObject    handle to rotate0 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of rotate0
if get(hObject,'Value')
    handles.frames = handles.originalFrames;
end
resetCrop(handles);
showFace(handles);
guidata(hObject, handles);

% --- Executes on button press in rotate90.
function rotate90_Callback(hObject, eventdata, handles)
% hObject    handle to rotate90 (see GCBO)

```

```

% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of rotate90
if get(hObject,'Value')
    for i = 1:handles.lastFrame
        handles.frames{i} = imrotate(handles.originalFrames{i}, 90);
    end
end
resetCrop(handles);
showFace(handles)
guidata(hObject, handles);

% --- Executes on button press in rotate180.
function rotate180_Callback(hObject, eventdata, handles)
% hObject handle to rotate180 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of rotate180
if get(hObject,'Value')
    for i = 1:handles.lastFrame
        handles.frames{i} = imrotate(handles.originalFrames{i}, 180);
    end
end
resetCrop(handles);
showFace(handles);
guidata(hObject, handles);

% --- Executes on button press in rotate270.
function rotate270_Callback(hObject, eventdata, handles)
% hObject handle to rotate270 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of rotate270
if get(hObject,'Value')
    for i = 1:handles.lastFrame
        handles.frames{i} = imrotate(handles.originalFrames{i}, 270);
    end
end
resetCrop(handles);

```

```

showFace(handles);
guidata(hObject, handles);

function eWidth_Callback(hObject, eventdata, handles)
% hObject    handle to eWidth (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of eWidth as text
%        str2double(get(hObject,'String')) returns contents of eWidth as a
double

% --- Executes during object creation, after setting all properties.
function eWidth_CreateFcn(hObject, eventdata, handles)
% hObject    handle to eWidth (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function eHeight_Callback(hObject, eventdata, handles)
% hObject    handle to eHeight (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of eHeight as text
%        str2double(get(hObject,'String')) returns contents of eHeight as a
double

% --- Executes during object creation, after setting all properties.
function eHeight_CreateFcn(hObject, eventdata, handles)

```

```

% hObject    handle to eHeight (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in resizeMode.
function resizeMode_Callback(hObject, eventdata, handles)
% hObject    handle to resizeMode (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
dlg = 0;
if (str2double(get(handles.eHeight, 'String')) ~=
str2double(get(handles.eWidth, 'String')))
    dlg = questdlg('Are you sure you want to RESIZE? Height and Width are not
equal. Image might get distorted', 'Size Mismatch', 'Yes', 'No', 'No');
end

if strcmp(dlg,'Yes')
    for i = 1:handles.lastFrame
        handles.frames{i} = imresize(handles.frames{i}, [256,256]);
    end
    resetCrop(handles);
    set(handles.testFrameButton, 'Enable', 'on');
end
guidata(hObject, handles);
showFace(handles);

% --- Executes on button press in rFaceManual.
function rFaceManual_Callback(hObject, eventdata, handles)
% hObject    handle to rFaceManual (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

% Hint: get(hObject,'Value') returns toggle state of rFaceManual
handles.manualFace = 1;
guidata(hObject, handles);
set(handles.sBackground, 'Enable', 'off');
set(handles.sLighting, 'Enable', 'off');

% --- Executes on button press in rFaceAuto.
function rFaceAuto_Callback(hObject, eventdata, handles)
% hObject    handle to rFaceAuto (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of rFaceAuto
handles.manualFace = 0;
set(handles.sBackground, 'Enable', 'on');
set(handles.sLighting, 'Enable', 'on');
guidata(hObject, handles);

% --- Executes on button press in rPointsManual.
function rPointsManual_Callback(hObject, eventdata, handles)
% hObject    handle to rPointsManual (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of rPointsManual
% Turning off threshold setting
set(handles.sThreshEye, 'Enable', 'off');
set(handles.eThreshEye, 'Enable', 'off');
set(handles.sThreshEyebrow, 'Enable', 'off');
set(handles.eThreshEyebrow, 'Enable', 'off');
set(handles.sThreshLips, 'Enable', 'off');
set(handles.eThreshLips, 'Enable', 'off');
set(handles.sThreshLipcorner, 'Enable', 'off');
set(handles.eThreshLipcorner, 'Enable', 'off');
set(handles.sThreshNose, 'Enable', 'off');
set(handles.eThreshNose, 'Enable', 'off');
set(handles.sThreshNosetril, 'Enable', 'off');
set(handles.eThreshNosetril, 'Enable', 'off');
set(handles.sThreshEyeOut, 'Enable', 'off');
set(handles.eThreshEyeOut, 'Enable', 'off');

```

```

set(handles.sThreshEyeIn, 'Enable', 'off');
set(handles.eThreshEyeIn, 'Enable', 'off');
set(handles.sThreshEBcorner, 'Enable', 'off');
set(handles.eThreshEBcorner, 'Enable', 'off');

handles.pointsManual = 1;
guidata(hObject, handles);

% --- Executes on button press in rPointsAuto.
function rPointsAuto_Callback(hObject, eventdata, handles)
% hObject    handle to rPointsAuto (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of rPointsAuto
set(handles.sThreshEye, 'Enable', 'on');
set(handles.eThreshEye, 'Enable', 'inactive');
set(handles.sThreshEyebrow, 'Enable', 'on');
set(handles.eThreshEyebrow, 'Enable', 'inactive');
set(handles.sThreshLips, 'Enable', 'on');
set(handles.eThreshLips, 'Enable', 'inactive');
set(handles.sThreshLipcorner, 'Enable', 'on');
set(handles.eThreshLipcorner, 'Enable', 'inactive');
set(handles.sThreshNose, 'Enable', 'on');
set(handles.eThreshNose, 'Enable', 'inactive');
set(handles.sThreshNosetril, 'Enable', 'on');
set(handles.eThreshNosetril, 'Enable', 'inactive');
set(handles.sThreshEyeOut, 'Enable', 'on');
set(handles.eThreshEyeOut, 'Enable', 'inactive');
set(handles.sThreshEyeIn, 'Enable', 'on');
set(handles.eThreshEyeIn, 'Enable', 'inactive');
set(handles.sThreshEBcorner, 'Enable', 'on');
set(handles.eThreshEBcorner, 'Enable', 'inactive');

handles.pointsManual = 0;
guidata(hObject, handles);

function showFace(handles)
% Cropping Image
if get(handles.cCrop, 'Value')

```

```

        xCrop = str2double(get(handles.eCropX, 'String'));
        yCrop = str2double(get(handles.eCropY, 'String'));
        wCrop = str2double(get(handles.eCropW, 'String'));
        oHeight = str2double(get(handles.eHeight, 'String'));
        handles.frames{handles.frameNo}((oHeight+1-yCrop-wCrop):(oHeight-yCrop),
xCrop:(xCrop+wCrop)) = ...
            handles.frames{handles.frameNo}((oHeight+1-yCrop-wCrop):(oHeight-
yCrop), xCrop:(xCrop+wCrop))/2;
    end
    % Showing the image
    axes(handles.face);
    imshow(handles.frames{handles.frameNo});
    % Setting Height and Width values.
    set(handles.eWidth, 'String',
num2str(size(handles.frames{handles.frameNo},2)));
    set(handles.eHeight, 'String',
num2str(size(handles.frames{handles.frameNo},1)));

function resetCrop(handles)
set(handles.testFrameButton, 'Enable', 'off');
set(handles.testVideoButton, 'Enable', 'off');
% Setting Crop Panel
set(handles.eCropX, 'String', num2str(1));
set(handles.eCropY, 'String', num2str(1));
set(handles.eCropW, 'String', num2str(0));
set(handles.cCrop, 'Value', 0);
cCrop_Callback(handles.cCrop, [], handles);

% --- Executes on button press in testVideoButton.
function testVideoButton_Callback(hObject, eventdata, handles)
% hObject    handle to testVideoButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
offset.EBL = handles.points{handles.frameNo}(6,2) -
handles.points{handles.frameNo}(9,2);
    offset.EBR = handles.points{handles.frameNo}(1,2) -
handles.points{handles.frameNo}(4,2);
    axes(handles.face);
    if handles.pointsManual
        for i = 1:handles.lastFrame
            %visualize the selected points on the image

```



```

ImagePoints = handles.frames{i};

imshow(ImagePoints);
% right eye
in = round(ginput(1));
feature_points(1,1) = in(2);
feature_points(1,2) = in(1);
ImagePoints(feature_points(1,1) - 5 : feature_points(1,1) + 5 ,
feature_points(1,2)) = 256;
ImagePoints(feature_points(1,1), feature_points(1,2) - 5 :
feature_points(1,2) + 5) = 256;
imshow(ImagePoints);

%right eye inner corner
in = round(ginput(1));
feature_points(2,1) = in(2);
feature_points(2,2) = in(1);
ImagePoints(feature_points(2,1) - 5 : feature_points(2,1) + 5 ,
feature_points(2,2)) = 256;
ImagePoints(feature_points(2,1), feature_points(2,2) - 5 :
feature_points(2,2) + 5) = 256;
imshow(ImagePoints);

%right eye outer corner
in = round(ginput(1));
feature_points(3,1) = in(2);
feature_points(3,2) = in(1);
ImagePoints(feature_points(3,1) - 5 : feature_points(3,1) + 5 ,
feature_points(3,2)) = 256;
ImagePoints(feature_points(3,1), feature_points(3,2) - 5 :
feature_points(3,2) + 5) = 256;
imshow(ImagePoints);

%right eyebrow
in = round(ginput(1));
feature_points(4,1) = in(2);
feature_points(4,2) = in(1);
ImagePoints(feature_points(4,1) - 5 : feature_points(4,1) + 5 ,
feature_points(4,2)) = 256;
ImagePoints(feature_points(4,1), feature_points(4,2) - 5 :
feature_points(4,2) + 5) = 256;
imshow(ImagePoints);

```

```

%right eyebrow corner
in = round(ginput(1));
feature_points(5,1) = in(2);
feature_points(5,2) = in(1);
ImagePoints(feature_points(5,1) - 5 : feature_points(5,1) + 5 ,
feature_points(5,2)) = 256;
ImagePoints(feature_points(5,1), feature_points(5,2) - 5 :
feature_points(5,2) + 5) = 256;
imshow(ImagePoints);

%left eye
in = round(ginput(1));
feature_points(6,1) = in(2);
feature_points(6,2) = in(1);
ImagePoints(feature_points(6,1) - 5 : feature_points(6,1) + 5 ,
feature_points(6,2)) = 256;
ImagePoints(feature_points(6,1), feature_points(6,2) - 5 :
feature_points(6,2) + 5) = 256;
imshow(ImagePoints);

%left eye inner corner
in = round(ginput(1));
feature_points(7,1) = in(2);
feature_points(7,2) = in(1);
ImagePoints(feature_points(7,1) - 5 : feature_points(7,1) + 5 ,
feature_points(7,2)) = 256;
ImagePoints(feature_points(7,1), feature_points(7,2) - 5 :
feature_points(7,2) + 5) = 256;
imshow(ImagePoints);

%left eye outer corner
in = round(ginput(1));
feature_points(8,1) = in(2);
feature_points(8,2) = in(1);
ImagePoints(feature_points(8,1) - 5 : feature_points(8,1) + 5 ,
feature_points(8,2)) = 256;
ImagePoints(feature_points(8,1), feature_points(8,2) - 5 :
feature_points(8,2) + 5) = 256;
imshow(ImagePoints);

%left eyebrow

```

```

in = round(ginput(1));
feature_points(9,1) = in(2);
feature_points(9,2) = in(1);
ImagePoints(feature_points(9,1) - 5 : feature_points(9,1) + 5 ,
feature_points(9,2)) = 256;
ImagePoints(feature_points(9,1), feature_points(9,2) - 5 :
feature_points(9,2) + 5) = 256;
imshow(ImagePoints);

%left eyebrow corner
in = round(ginput(1));
feature_points(10,1) = in(2);
feature_points(10,2) = in(1);
ImagePoints(feature_points(10,1) - 5 : feature_points(10,1) + 5 ,
feature_points(10,2)) = 256;
ImagePoints(feature_points(10,1), feature_points(10,2) - 5 :
feature_points(10,2) + 5) = 256;
imshow(ImagePoints);

%left mouth corner
in = round(ginput(1));
feature_points(11,1) = in(2);
feature_points(11,2) = in(1);
ImagePoints(feature_points(11,1) - 5 : feature_points(11,1) + 5 ,
feature_points(11,2)) = 256;
ImagePoints(feature_points(11,1), feature_points(11,2) - 5 :
feature_points(11,2) + 5) = 256;
imshow(ImagePoints);

%rightmouth corner
in = round(ginput(1));
feature_points(12,1) = in(2);
feature_points(12,2) = in(1);
ImagePoints(feature_points(12,1) - 5 : feature_points(12,1) + 5 ,
feature_points(12,2)) = 256;
ImagePoints(feature_points(12,1), feature_points(12,2) - 5 :
feature_points(12,2) + 5) = 256;
imshow(ImagePoints);

%upper lip
in = round(ginput(1));
feature_points(13,1) = in(2);

```

```

        feature_points(13,2) = in(1);
        ImagePoints(feature_points(13,1) - 5 : feature_points(13,1) + 5 ,
feature_points(13,2)) = 256;
        ImagePoints(feature_points(13,1), feature_points(13,2) - 5 :
feature_points(13,2) + 5) = 256;
        imshow(ImagePoints);

%lower lip
in = round(ginput(1));
feature_points(14,1) = in(2);
feature_points(14,2) = in(1);
ImagePoints(feature_points(14,1) - 5 : feature_points(14,1) + 5 ,
feature_points(14,2)) = 256;
ImagePoints(feature_points(14,1), feature_points(14,2) - 5 :
feature_points(14,2) + 5) = 256;
imshow(ImagePoints);

% nose
in = round(ginput(1));
feature_points(15,1) = in(2);
feature_points(15,2) = in(1);
ImagePoints(feature_points(15,1) - 5 : feature_points(15,1) + 5 ,
feature_points(15,2)) = 256;
ImagePoints(feature_points(15,1), feature_points(15,2) - 5 :
feature_points(15,2) + 5) = 256;
imshow(ImagePoints);

%nose corner left
in = round(ginput(1));
feature_points(16,1) = in(2);
feature_points(16,2) = in(1);
ImagePoints(feature_points(16,1) - 5 : feature_points(16,1) + 5 ,
feature_points(16,2)) = 256;
ImagePoints(feature_points(16,1), feature_points(16,2) - 5 :
feature_points(16,2) + 5) = 256;
imshow(ImagePoints);

%nose corner right
in = round(ginput(1));
feature_points(17,1) = in(2);
feature_points(17,2) = in(1);

```

```

        ImagePoints(feature_points(17,1) - 5 : feature_points(17,1) + 5 ,
feature_points(17,2)) = 256;
        ImagePoints(feature_points(17,1), feature_points(17,2) - 5 :
feature_points(17,2) + 5) = 256;
        imshow(ImagePoints);

        handles.points{i} = feature_points;
        set(handles.sFrame, 'Value', i);

        if(i == handles.neutral)
            handles.neutralImage = ImagePoints;
        end
    end
else
    for i = handles.frameNo-1:-1:1
        set(handles.sFrame, 'Value', i);
        sFrame_Callback(handles.sFrame, [], handles);
        [ImagePoints test_op_reg handles.points{i}] = run2(handles.frames{i},
handles.faceCoord, handles.points(i+1), handles.thresh, offset);
        imshow(ImagePoints), drawnow;
        if(i == handles.neutral)
            handles.neutralImage = ImagePoints;
        end
    end
    for i = handles.frameNo+1:handles.lastFrame
        set(handles.sFrame, 'Value', i);
        sFrame_Callback(handles.sFrame, [], handles);
        [ImagePoints test_op_reg handles.points{i}] = run2(handles.frames{i},
handles.faceCoord, handles.points(i-1), handles.thresh, offset);
        imshow(ImagePoints), drawnow;
        if(i == handles.neutral)
            handles.neutralImage = ImagePoints;
        end
    end
end
end
end
guidata(hObject, handles);

% --- Executes on button press in saveSetButton.
function saveSetButton_Callback(hObject, eventdata, handles)
% hObject    handle to saveSetButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

op.thresh = handles.thresh;
op.featurePoints = handles.points;
op.neutral = handles.neutral;
op.neutralImage = handles.neutralImage;
assignin('base', 'videoSet', op);
close(handles.figure1);

% --- Executes on slider movement.
function sBackground_Callback(hObject, eventdata, handles)
% hObject    handle to sBackground (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
handles.thresh.background = get(hObject,'Value');
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function sBackground_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sBackground (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function sLighting_Callback(hObject, eventdata, handles)
% hObject    handle to sLighting (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
handles.thresh.lighting = get(hObject,'Value');
guidata(hObject, handles);

```

```

% --- Executes during object creation, after setting all properties.
function sLighting_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sLighting (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function sThreshEye_Callback(hObject, eventdata, handles)
% hObject    handle to sThreshEye (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.eThreshEye, 'String', num2str(get(hObject, 'Value')));
handles.thresh.eye = get(hObject,'Value');
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function sThreshEye_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sThreshEye (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function sThreshEyebrow_Callback(hObject, eventdata, handles)
% hObject    handle to sThreshEyebrow (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB

```

```

% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%          get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.eThreshEyebrow, 'String', num2str(get(hObject, 'Value')));
handles.thresh.eyebrow = get(hObject, 'Value');
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function sThreshEyebrow_CreateFcn(hObject, eventdata, handles)
% hObject      handle to sThreshEyebrow (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

function eThreshEye_Callback(hObject, eventdata, handles)
% hObject      handle to eThreshEye (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of eThreshEye as text
%          str2double(get(hObject,'String')) returns contents of eThreshEye as a
double

% --- Executes during object creation, after setting all properties.
function eThreshEye_CreateFcn(hObject, eventdata, handles)
% hObject      handle to eThreshEye (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%          See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))

```



```

        set(hObject,'BackgroundColor','white');
    end

function eThreshEyebrow_Callback(hObject, eventdata, handles)
% hObject    handle to eThreshEyebrow (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of eThreshEyebrow as text
%         str2double(get(hObject,'String')) returns contents of eThreshEyebrow
as a double

% --- Executes during object creation, after setting all properties.
function eThreshEyebrow_CreateFcn(hObject, eventdata, handles)
% hObject    handle to eThreshEyebrow (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on slider movement.
function sThreshLipcorner_Callback(hObject, eventdata, handles)
% hObject    handle to sThreshLipcorner (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.eThreshLipcorner, 'String', num2str(get(hObject, 'Value')));
handles.thresh.lipcorner = get(hObject,'Value');
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.

```

```

function sThreshLipcorner_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sThreshLipcorner (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function sThreshLips_Callback(hObject, eventdata, handles)
% hObject    handle to sThreshLips (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.eThreshLips, 'String', num2str(get(hObject, 'Value')));
handles.thresh.lips = get(hObject,'Value');
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function sThreshLips_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sThreshLips (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

function eThreshLips_Callback(hObject, eventdata, handles)
% hObject    handle to eThreshLips (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

% Hints: get(hObject,'String') returns contents of eThreshLips as text
%         str2double(get(hObject,'String')) returns contents of eThreshLips as a
double

% --- Executes during object creation, after setting all properties.
function eThreshLips_CreateFcn(hObject, eventdata, handles)
% hObject    handle to eThreshLips (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function eThreshLipcorner_Callback(hObject, eventdata, handles)
% hObject    handle to eThreshLipcorner (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of eThreshLipcorner as text
%         str2double(get(hObject,'String')) returns contents of eThreshLipcorner
as a double

% --- Executes during object creation, after setting all properties.
function eThreshLipcorner_CreateFcn(hObject, eventdata, handles)
% hObject    handle to eThreshLipcorner (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

```

end

% --- Executes on slider movement.
function sThreshNose_Callback(hObject, eventdata, handles)
% hObject    handle to sThreshNose (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.eThreshNose, 'String', num2str(get(hObject, 'Value')));
handles.thresh.nose = get(hObject,'Value');
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function sThreshNose_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sThreshNose (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function sThreshNosetril_Callback(hObject, eventdata, handles)
% hObject    handle to sThreshNosetril (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.eThreshNosetril, 'String', num2str(get(hObject, 'Value')));
handles.thresh.nosetril = get(hObject,'Value');
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function sThreshNosetril_CreateFcn(hObject, eventdata, handles)

```

```

% hObject    handle to sThreshNose (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

function eThreshNose_Callback(hObject, eventdata, handles)
% hObject    handle to eThreshNose (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of eThreshNose as text
%         str2double(get(hObject,'String')) returns contents of eThreshNose
as a double

% --- Executes during object creation, after setting all properties.
function eThreshNose_CreateFcn(hObject, eventdata, handles)
% hObject    handle to eThreshNose (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function eThreshNose_Callback(hObject, eventdata, handles)
% hObject    handle to eThreshNose (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

% Hints: get(hObject,'String') returns contents of eThreshNose as text
%         str2double(get(hObject,'String')) returns contents of eThreshNose as a
double

% --- Executes during object creation, after setting all properties.
function eThreshNose_CreateFcn(hObject, eventdata, handles)
% hObject    handle to eThreshNose (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on slider movement.
function sThreshEyeIn_Callback(hObject, eventdata, handles)
% hObject    handle to sThreshEyeIn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.eThreshEyeIn, 'String', num2str(get(hObject, 'Value')));
handles.thresh.eyeIn = get(hObject,'Value');
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function sThreshEyeIn_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sThreshEyeIn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

```

```

function eThreshEyeIn_Callback(hObject, eventdata, handles)
% hObject    handle to eThreshEyeIn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of eThreshEyeIn as text
%        str2double(get(hObject,'String')) returns contents of eThreshEyeIn as
a double

% --- Executes during object creation, after setting all properties.
function eThreshEyeIn_CreateFcn(hObject, eventdata, handles)
% hObject    handle to eThreshEyeIn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on slider movement.
function sThreshEyeOut_Callback(hObject, eventdata, handles)
% hObject    handle to sThreshEyeOut (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.eThreshEyeOut, 'String', num2str(get(hObject, 'Value')));
handles.thresh.eyeOut = get(hObject,'Value');
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function sThreshEyeOut_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sThreshEyeOut (see GCBO)

```

```

% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

function eThreshEyeOut_Callback(hObject, eventdata, handles)
% hObject handle to eThreshEyeOut (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of eThreshEyeOut as text
% str2double(get(hObject,'String')) returns contents of eThreshEyeOut as
a double

% --- Executes during object creation, after setting all properties.
function eThreshEyeOut_CreateFcn(hObject, eventdata, handles)
% hObject handle to eThreshEyeOut (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on slider movement.
function sThreshEBcorner_Callback(hObject, eventdata, handles)
% hObject handle to sThreshEBcorner (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider

```



```

%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.eThreshEBcorner, 'String', num2str(get(hObject, 'Value')));
handles.thresh.ebCorner = get(hObject, 'Value');
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function sThreshEBcorner_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sThreshEBcorner (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

function eThreshEBcorner_Callback(hObject, eventdata, handles)
% hObject    handle to eThreshEBcorner (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of eThreshEBcorner as text
%         str2double(get(hObject,'String')) returns contents of eThreshEBcorner
as a double

% --- Executes during object creation, after setting all properties.
function eThreshEBcorner_CreateFcn(hObject, eventdata, handles)
% hObject    handle to eThreshEBcorner (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

Filename : getFaceRegion.m

Extracts the face region using the seeded region growing algorithm.

```
function Im = getFaceRegion(I, thresh_light, thresh_bg)

%Making image blurred with gabor filter.
[G,I1] = gaborfilter1(I,4,4,0,pi/4);
BW=edge(I1,'canny');
I=double(imadjust(I,stretchlim(I),[]));
% using canny edge detector to find edges
I=I-BW*255;
Bg = zeros(size(I));

% choosing 4 seed pixels background separation by seed region growing
% algorithm
g1=[20 20];
g2=[20 240];
g3=[150 20];
g4=[150 240];
splBack = [g1(1) g1(2); g2(1) g2(2); g3(1) g3(2); g4(1) g4(2)];
fcheckBack = @checkBackground;
display('Separating the background');
while splBack ~=zeros(size(splBack))
    [Bg,splBack]=feval(fcheckBack,splBack,I,Bg, thresh_light ,thresh_bg);
end

% substituing background with black color in the original image
for xcord = 1:256
    for ycord = 1:256
        I(xcord, ycord) = I(xcord, ycord) * ~Bg(xcord, ycord);
    end
end

Im = zeros(size(I));

%choosing 3 seed pixels
g1=[128 128];
g3=[160 128];
g4=[96 160];
g5=[96 128];
g6=[96 96];
```

```

% arbitrarily chosen co-ordinate on the forehead
spl = [g1; g3; g4; g5; g6];
% refining coordinates
spl = feval(@seedPixel, spl, I, 150);
% finished choosing seed pixels.

fcheck=@check3;% function handle
threshold = thresh_light*30;
First = zeros(size(I));
wrong = 0;
wrongFirst = 0;
spl = [g1(1) g1(2); g2(1) g2(2); g3(1) g3(2) ]; % co-ordinate on the fore head
while spl ~=zeros(size(spl))
    [Im,spl,wrong] = feval(fcheck,spl,I,Im,threshold);
    if (wrong == 1)
        wrongFirst = 1;
    end
end

if wrongFirst == 1
    wrong = 0;
    First = Im;
    Im = zeros(size(I));
    spl = [g1(1) g1(2); g2(1) g2(2); g3(1) g3(2) ]; % co-ordinate on the fore
head

    while spl ~=zeros(size(spl))
        [Im,spl,wrong] = feval(fcheck,spl,I,Im,threshold);
        if wrong == 1
            display('Lowering threshold...');
            Im = zeros(size(I));
            spl = [g1(1) g1(2); g2(1) g2(2); g3(1) g3(2) ]; % co-ordinate on the
fore head

            threshold = threshold - 1;
        end
    end

    for xcord = 210:255
        for ycord = 1:255
            if Im(xcord, ycord) == 0 && First(xcord, ycord) == 1
                First(xcord, ycord) = 0;
            end
        end
    end

end
end

```

```
else
    First = Im;
end
imshow(First,[]);
Im = First;
```

Filename : checkBackground.m

Separates the background using seeded region growing algorithm.

```
function [J,newspl] = checkBackground(spl,I,J, thresh_light, thresh_bg)

newspl=[];
thresh_neighbor = thresh_light * 80;
thresh_diff = thresh_bg;
for i=1:size(spl,1)
    current_posn=spl(i,:);
    N=neighbor(current_posn);
    % if all neighborhood pixels follow the condition put it in the spl
    for j=1:8
        if N(j,1) < 256 && (N(j,1) > 0) && N(j,2) < 256 && N(j,2) > 0
            if (abs(I(N(j,1),N(j,2))-I(spl(i,1),spl(i,2)))<=thresh_diff) ...
                && J(N(j,1),N(j,2))==0 && I(N(j,1),N(j,2)) > thresh_neighbor
                    newspl(size(newspl,1)+1,:)=N(j,:);    % add to the spl
                    J(N(j,1),N(j,2))=1;
                end
            end
        end
    end
end

function N=neighbor(S)

r=S(1,1);
c=S(1,2);
N=[r-1 c-1;r-1 c;r-1 c+1;r c-1;r c+1;r+1 c-1;r+1 c;r+1 c+1];
```

Filename : seedPixel.m

Initializes the seed pixels for the seeded region growing algorithm.

```
function spl = seedPixel(spl, Image, thresh)
% Method for choosing seed pixels.
%looking for pixels within a 10 pixel window around some arbitrarily
%chossen pixels with good chances of being located in the forehead and
%intensity less than 150. A maximum of 20 iterations are done to pick the
%pixel or else the original pixel is picked. But this seel pixel would
%most probably eliminated in the first cycle because its neighbourhood
%seems to be dark upon random searching.

for i = 1:size(spl,1)
    bad = 0;
    while (Image(spl(i,1), spl(i,2)) >= thresh & bad < 20)
        spl(i,1) = spl(i,1) + round(10*rand(1));
        if spl(i,1) <= 0; spl(i,1) = spl(i,1)+10; end;
        if spl(i,1) >= 255; spl(i,1) = spl(i,1)-10; end;
        spl(i,2) = spl(i,2) + round(10*rand(1));
        if spl(i,2) <= 0; spl(i,2) = spl(i,2)+10; end;
        if spl(i,2) >= 255; spl(i,2) = spl(i,2)-10; end;
        bad = bad + 1;
    end
end
```

Filename : check3.m

Compares the seed pixel with its neighbors to check if they can be included in the region.

```
function [J,newspl,wrong] = check3(spl,I,J,thresh)

newspl=[];
wrong = 0;
wrongLocal = 0;
for i=1:size(spl,1)
    current_posn=spl(i,:);
    N=neighbor(current_posn);

    % if all neighborhood pixels follow the condition put it in the spl
    for j=1:6
        if N(j,1) < 1 || N(j,1) > 250 || N(j,2) < 1 || N(j,2) > 250
            wrongLocal = 1;
        else
            if abs(I(N(j,1),N(j,2))-I(spl(i,1),spl(i,2))) <= thresh &&
J(N(j,1),N(j,2))==0 && I(N(j,1),N(j,2)) > 20

                newspl(size(newspl,1)+1,:)=N(j,:);        % add to the spl
                J(N(j,1),N(j,2))=1;
            end
        end
    end
    if (wrongLocal == 1)
        wrong = 1;
        wrongLocal = 0;
    end
end
end

function N=neighbor(S)

r=S(1,1);
c=S(1,2);
N=[r-1 c-1;r-1 c;r-1 c+1;r c-1;r c+1;r+1 c];
```

Filename : testFrame.m

This runs the feace separation and feature extraction algorithms upon the present frame.

```
function [Image_Points feature_points] = testFrame(Image, faceCoord, thresh)

load 'MouthTF.mat';

Image1 = Image;
Image2 = Image;
Image_Points = Image;

Rmin = faceCoord(1);
Rmax = faceCoord(2);
Cmin = faceCoord(3);
Cmax = faceCoord(4);

deltaRow = Rmax - Rmin;
deltaCol = Cmax - Cmin;

%Aproximate Left eye region
EyeLRmin = Rmin;
EyeLRmax = Rmin + round(deltaRow / 2);
EyeLCmin = Cmin;
EyeLCmax = Cmin + round(deltaCol / 2);

%Aproximate right eye region
EyeRRmin = Rmin;
EyeRRmax = Rmin + round(deltaRow / 2);
EyeRCmin = Cmin + round(deltaCol / 2);
EyeRCmax = Cmax;

%Aproximate left mouth half region
MouthLeftRmin = Rmin + round(deltaRow / 2);
MouthLeftRmax = Rmax;
MouthLeftCmin = Cmin;
MouthLeftCmax = Cmin + round(deltaCol / 2);

%Aproximate left mouth half region
MouthRightRmin = Rmin + round(deltaRow / 2);
MouthRightRmax = Rmax;
```



```

MouthRightCmin = Cmin + round(deltaCol / 2);
MouthRightCmax = Cmax;

width = Cmax - Cmin;

dt = floor((Cmax - Cmin) / 4);

% check if the template can get out of the image while looking for the eye
if EyeRRmin < (dt / 2)
    EyeRRmin = round(dt / 2) + 1;
end
if EyeRCmax > (255 - (dt / 2))
    EyeRCmax = 255 - round(dt / 2) + 1;
end;

% Finding Right eye
border = Cmax - round(deltaCol / 8);
display('finding right eye cordينات.....');
right_eye = Pso2_eye(EyeRRmin, EyeRRmax, EyeRCmin, EyeRCmax, Image, width, 1,
border, thresh.eye, thresh.lighting);

Image_Points(right_eye(1), (right_eye(2) - 5): (right_eye(2) + 5)) = 256;
Image_Points((right_eye(1) - 5): (right_eye(1) + 5), right_eye(2)) = 256;

% call function to extract the coordinates of corners of the right eye
display('finding right eye corners cordينات.....');
right_eye_corners = getRightEyeCorners(right_eye, Image2, dt, thresh.eyeIn);
right_eye_in_corner = [right_eye_corners(1) right_eye_corners(2)];
right_eye_out_corner = [right_eye_corners(3) right_eye_corners(4)];

% Visualising Right Eye
Image_Points(right_eye(1), (right_eye(2) - 5): (right_eye(2) + 5)) = 256;
Image_Points((right_eye(1) - 5): (right_eye(1) + 5), right_eye(2)) = 256;
Image_Points(right_eye_in_corner(1), (right_eye_in_corner(2) - 5):
(right_eye_in_corner(2) + 5))=256;
Image_Points((right_eye_in_corner(1) - 5): (right_eye_in_corner(1) + 5),
right_eye_in_corner(2))=256;
Image_Points(right_eye_out_corner(1), (right_eye_out_corner(2) - 5):
(right_eye_out_corner(2) + 5))=256;
Image_Points((right_eye_out_corner(1) - 5): (right_eye_out_corner(1) + 5),
right_eye_out_corner(2))=256;

```

```

%extracting right eyebrow region coordinates
EyebrowRRmin = right_eye(1) - floor(1.5 * dt);
EyebrowRRmax = right_eye(1) - floor(0.45 * dt);
EyebrowRCmin = right_eye(2) - floor(dt/4);
EyebrowRCmax = right_eye(2) + floor(dt/2);

% check if the eyebrow template can get out of the image when searching
% for the eyebrow
if EyebrowRRmin < round(dt/2)
    EyebrowRRmin = round(dt/2) + 1;
end

% Finding Right eyebrow
    display('finding right eyebrow coordinates.....');
    right_eyebrow =
Pso_eyebrow(EyebrowRRmin,EyebrowRRmax,EyebrowRCmin,EyebrowRCmax, Image,width);

    % call a function to extract the coordinates of the inner corner of the
    % right eyebrow
    display('finding right eyebrow inner corner coordinates.....');
    right_eyebrow_corner = getRightEyebrowCorner(right_eyebrow,Image2,dt,
thresh.ebCorner);

% Visualizing Right Eyebrow
Image_Points(right_eyebrow(1), (right_eyebrow(2) - 5): (right_eyebrow(2) +
5))=256;
Image_Points((right_eyebrow(1) - 5): (right_eyebrow(1) + 5),
right_eyebrow(2))=256;
Image_Points(right_eyebrow_corner(1), (right_eyebrow_corner(2) - 5):
(right_eyebrow_corner(2) + 5))=256;
Image_Points((right_eyebrow_corner(1) - 5): (right_eyebrow_corner(1) + 5),
right_eyebrow_corner(2))=256;

% check if the template can get out of the image while looking for the eye
if EyeLRmin < (dt / 2)
    EyeLRmin = round(dt / 2) + 1;
end
if EyeLCmin > (dt / 2)
    EyeLCmin = round(dt / 2) + 1;
end;

```

```

% Finding Left Eye
border = Cmin + round(deltaCol / 8);
display('finding left eye cordinates.....');
left_eye =
Pso2_eye(EyeLRmin, EyeLRmax, EyeLCmin, EyeLCmax, Image, width, 0, border, thresh.eye,
thresh.lighting);

% call function to extract the coordinates of corners of the left eye
display('finding left eye corners cordinates.....');
left_eye_corners = getLeftEyeCorners(left_eye, Image2, dt, thresh.eyeIn);
left_eye_in_corner = [left_eye_corners(1) left_eye_corners(2)];
left_eye_out_corner = [left_eye_corners(3) left_eye_corners(4)];

% Visualizing Left Eye
Image_Points(left_eye(1), (left_eye(2) - 5): (left_eye(2) + 5))=256;
Image_Points((left_eye(1) - 5): (left_eye(1) + 5), left_eye(2))=256;
Image_Points(left_eye_in_corner(1), (left_eye_in_corner(2) - 5):
(left_eye_in_corner(2) + 5))=256;
Image_Points((left_eye_in_corner(1) - 5): (left_eye_in_corner(1) + 5),
left_eye_in_corner(2))=256;

Image_Points(left_eye_out_corner(1), (left_eye_out_corner(2) - 5):
(left_eye_out_corner(2) + 5))=256;
Image_Points((left_eye_out_corner(1) - 5): (left_eye_out_corner(1) + 5),
left_eye_out_corner(2))=256;

%extracting left eyebrow region coordinates
EyebrowLRmin = left_eye(1) - floor(1.5 * dt);
EyebrowLRmax = left_eye(1) - floor(0.45 * dt);
EyebrowLCmin = left_eye(2) - floor(dt/2);
EyebrowLCmax = left_eye(2) + floor(dt/4);

% check if the eyebrow template can get out of the image when searching
% for the eyebrow
if EyebrowLRmin < round(dt/2)
    EyebrowLRmin = round(dt/2) + 1;
end

% Finding Left Eyebrow
display('finding left eyebrow cordinates.....');

```

```

left_eyebrow =
Pso_eyebrow(EyebrowLRmin,EyebrowLRmax,EyebrowLCmin,EyebrowLCmax, Image,width);

display('finding left eyebrow inner corner cordinates.....');
left_eyebrow_corner =
getLeftEyebrowCorner(left_eyebrow, Image2,dt,thresh.ebCorner);

% Vizualising Left Eyebrow
Image_Points(left_eyebrow_corner(1), (left_eyebrow_corner(2) - 5):
(left_eyebrow_corner(2) + 5))=256;
Image_Points((left_eyebrow_corner(1) - 5): (left_eyebrow_corner(1) + 5),
left_eyebrow_corner(2))=256;

Image_Points(left_eyebrow(1), (left_eyebrow(2) - 5): (left_eyebrow(2) +
5))=256;
Image_Points((left_eyebrow(1) - 5): (left_eyebrow(1) + 5),
left_eyebrow(2))=256;

% Finding left corner of mouth
center = left_eye(2);

% check if the template for mouth corner can get out of the image.

if MouthLeftRmax > (255 - dt/2)
    MouthLeftRmax = round(255 - dt/2) + 1;
end

display('finding left mouth cordinates.....');
mouth_left =
Pso2_mouth_left(MouthLeftRmin,MouthLeftRmax,MouthLeftCmin,MouthLeftCmax, Image,center,
thresh.lips);

mouth_corner_left = mouth_left(1:2);

% Finding Right Corner of Mouth
center = right_eye(2);

% check if the template for mouth corner can get out of the image.
if MouthRightRmax > (255 - dt/2)
    MouthRightRmax = round(255 - dt/2) + 1;
end

```

```

        display('finding right mouth cordinates.....');
        mouth_right =
Pso2_mouth_right (MouthRightRmin,MouthRightRmax,MouthRightCmin,MouthRightCmax, Image, cen
ter);

        mouth_corner_right = mouth_right(1:2);

% Checking if the mouth is located right
        vector = [(mouth_corner_right(1) - mouth_corner_left(1))
(mouth_corner_right(2) - mouth_corner_left(2))];
        Y = sim(net,vector);
        if Y == [1;0]
            display('Mouth corners located successfully. ');
        else
            display('Mouth corners were NOT located successfully. ');
            display('Fixing it... ');
            mvec = mouth_fix(mouth_left,mouth_right,net);
            mouth_corner_left = mvec(1:2);
            mouth_corner_right = mvec(3:4);
        end
% Visualizing mouth corners
        Image_Points(mouth_corner_left(1), (mouth_corner_left(2) - 5):
(mouth_corner_left(2) + 5))=256;
        Image_Points((mouth_corner_left(1) - 5): (mouth_corner_left(1) + 5),
mouth_corner_left(2) )=256;

        Image_Points(mouth_corner_right(1), (mouth_corner_right(2) - 5):
(mouth_corner_right(2) + 5))=256;
        Image_Points((mouth_corner_right(1) - 5): (mouth_corner_right(1) + 5),
mouth_corner_right(2) )=256;

% call a function to extract the upper and lower lip mid point coordinates
% Locating upper and lower lip
        display('finding mouth lips cordinates.....');
        mouth_Lips =
getMouthLipsCoordinates (Imagel,mouth_corner_left,mouth_corner_right,dt,thresh.lips);

% extracted coordinates of the midpoint of the upper lip
        mouth_lip_upper = [mouth_Lips(1)  mouth_Lips(2)];
        mouth_lip_lower = [mouth_Lips(3)  mouth_Lips(4)];

% Visualising Upper and lower Lip

```

```

    Image_Points(mouth_lip_upper(1), (mouth_lip_upper(2) - 5): (mouth_lip_upper(2)
+ 5))=256;
    Image_Points((mouth_lip_upper(1) - 5): (mouth_lip_upper(1) + 5),
mouth_lip_upper(2) )=256;

    Image_Points(mouth_lip_lower(1), (mouth_lip_lower(2) - 5): (mouth_lip_lower(2)
+ 5))=256;
    Image_Points((mouth_lip_lower(1) - 5): (mouth_lip_lower(1) + 5),
mouth_lip_lower(2) )=256;

    % check whether the upper lip was located right - must be above the corners
    % of mouth

    midpoint = floor((mouth_corner_left(1) + mouth_corner_right(1)) / 2);

    if (mouth_lip_upper(1) < midpoint)
        upper_lip = mouth_lip_upper(1);
    else
        %display('Mouth upper lip was not located successfully. ');
        upper_lip = midpoint - floor(dt / 2);
    end

    % Locating nose
    NoseRmin = upper_lip - floor(1.5 * dt);
    NoseRmax = upper_lip - floor(dt / 4);
    NoseCmin = mouth_lip_upper(2) - floor(0.75 * dt);
    NoseCmax = mouth_lip_upper(2) + floor(0.75 * dt);

    display('finding nose coordinates.....');
    nose =
Pso_nose(NoseRmin,NoseRmax,NoseCmin,NoseCmax,Image2,width,thresh.nose);

    Image_Points(nose(1), (nose(2) - 5): (nose(2) + 5))=256;
    Image_Points((nose(1) - 5): (nose(1) + 5),nose(2) )=256;

    display('finding nostril coordinates.....');
    nose_corners = getNoseCorners(nose,Image2,dt,thresh.nostril);

    % Visualising nose coordinates
    Image_Points(nose(1), (nose(2) - 5): (nose(2) + 5))=256;
    Image_Points((nose(1) - 5): (nose(1) + 5),nose(2) )=256;

```

```

        Image_Points(nose_corners(1), (nose_corners(2) - 5): (nose_corners(2) +
5))=256;
        Image_Points((nose_corners(1) - 5): (nose_corners(1) + 5),nose_corners(2)
)=256;

        Image_Points(nose_corners(3), (nose_corners(4) - 5): (nose_corners(4) +
5))=256;
        Image_Points((nose_corners(3) - 5): (nose_corners(3) + 5),nose_corners(4)
)=256;

        % vector with coordinates of all feature points
        feature_points = [right_eye; right_eye_in_corner; right_eye_out_corner;
right_eyebrow; right_eyebrow_corner;...
                        left_eye; left_eye_in_corner; left_eye_out_corner;
left_eyebrow; left_eyebrow_corner;...
                        mouth_corner_left; mouth_corner_right; mouth_lip_upper;
mouth_lip_lower;...
                        nose; nose_corners(1:2); nose_corners(3:4)];

```

Filename : Pso2_eye.m

This is the particle swarm optimization algorithm used to locate the eye.

```
function [VeryBest M]=Pso2_eye(Rmin,Rmax,Cmin,Cmax,I,width, side,border,
thresh_eye, thresh_light)

n = 5; % number of particles
group = 3; % number of groups

Vmax= 10; % Max velocity..set arbitrarily
Vmin= 3; % particle move one pixel at a time at minimum
Gbest=[0 0 0 0 0 0];
GbestFit = [0 0 0];
MaxFit = [0 0 0];
MaxCord = [0 0 0 0 0 0];
best_response = [0 0 0];
best = [0 0 0];
VeryBest = [0 0];

wi = 0.85; %inertial weight
c1=0.5;
c2=0.5;

ind=0;
coord=[];
prev_best_values=[];
threshold = 400;
dt = floor(width/4);

%initialising particles with random velocities and setting
%Pbest to the initial position
%Deploying the particles into an region where the eye is expected.
deltaR3 = round((Rmax - Rmin) / 3);
deltaR2 = round((Rmax - Rmin) / 2);
deltaC4 = round((Cmax - Cmin) / 4);
deltaC = Cmax - Cmin;

% initializing particles in all groups
for g = 0:(group - 1)
```



```

for i = 1:n
    partInd = g*n + i;
    particle(partInd).posn=[round((Rmin + 2*deltaR3)+ deltaR3*rand(1))
round((Cmin + deltaC4) + (2 * deltaC4)*rand(1))];
    particle(partInd).vel=[round(Vmin+(Vmax-Vmin)*rand(1))
round(Vmin+(Vmax-Vmin)*rand(1))];

    particle(partInd).Pbest = particle(partInd).posn;
    particle(partInd).func_response =
cost_function7subReg(I,particle(partInd).posn(1),
particle(partInd).posn(2),dt,side,border, thresh_eye, thresh_light);
    particle(partInd).func_resp_prev = particle(partInd).func_response;
end

    bestfind = find([particle((g*n + 1):(g*n + n)).func_response] ==
max([particle((g*n + 1):(g*n + n)).func_response]));
    best(g+1) = g*n + bestfind(1);
    Gbest((2*g + 1):(2*g + 2)) = particle(best(g+1)).posn;
    GbestFit(g+1) = particle(best(g+1)).func_response;
    MaxCord((2*g + 1):(2*g + 2)) = Gbest((2*g + 1):(2*g + 2));
    best_response(g+1) = particle(best(g+1)).func_response;
    MaxFit(g+1) = best_response(g+1);
end

itr=1;
t=1;
thresh_list=[];

while ind == 0 && itr < 22
    %Compute particles new position and velocities

    % if GbestFit > 0 compute new position and velocities - algorithm
    % is converging.
    if (GbestFit(g+1) > 0)
        %In each group
        for g = 0:(group-1)
            for i=1:n
                partInd = g*n + i;

particle(partInd).vel(1)=wi*particle(partInd).vel(1)+c1*rand(1)*(particle(partInd).Pbest(1)- particle(partInd).posn(1))+ c2*rand(1)*(Gbest(2*g + 1)-
particle(partInd).posn(1));

```

```

        particle(partInd).vel(2)= wi*particle(partInd).vel(2) +
c1*rand(1) * (particle(partInd).Pbest(2) -particle(partInd).posn(2))+
c2*rand(1)*(Gbest(2*g+2) - particle(partInd).posn(2));
        particle(partInd).posn(1)= round(particle(partInd).posn(1) +
particle(partInd).vel(1));
        particle(partInd).posn(2)= round(particle(partInd).posn(2) +
particle(partInd).vel(2));

        %checking if the solution lies within domain
        if particle(partInd).posn(1) <Rmin || particle(partInd).posn(1)
> Rmax || particle(partInd).posn(2) <Cmin || particle(partInd).posn(2) > Cmax
            % setting position to previous P-best
            particle(partInd).posn = particle(partInd).Pbest;
            % setting velocity to max
            particle(partInd).vel=[rand(1)*Vmax rand(1)*Vmax];
        end

        %Calculating the function response for each particle with new
        %positions
        particle(partInd).func_response = cost_function7subReg(I,
particle(partInd).posn(1), particle(partInd).posn(2), dt, side, border, thresh_eye,
thresh_light);

        %checking and updating Pbest
        if particle(partInd).func_response >
particle(partInd).func_resp_prev
            particle(partInd).Pbest = particle(partInd).posn;
        end

        particle(partInd).func_resp_prev =
particle(partInd).func_response;

    end
end

V = [particle((g*n + 1):(g*n + n)).func_response];
best_ones = find(V == max(V));
best(g+1) = g*n + best_ones(1);

% Gbest is a global best of all times
% display(particle(best(g+1)).func_response);
if particle(best(g+1)).func_response > GbestFit(g+1)

```

```

        Gbest((2*g + 1):(2*g + 2)) = particle(best(g+1)).Pbest;
        GbestFit(g+1) = particle(best(g+1)).func_response;
    end
    best_response(g+1) = particle(best(g+1)).func_response;
    if (best_response(g+1) < MaxFit(g+1))
        MaxCord((2*g + 1):(2*g + 2)) = Gbest((2*g + 1):(2*g + 2));
    end

else
% if is not > 0 than deploy te particles again in new random
% positions and with new random speeds
    if (itr < 5)
        for g = 0:(group-1)
            for i=1:n
                partInd = g*n + i;
                particle(partInd).posn=[round((Rmin + 2*deltaR3)+
deltaR3*rand(1)) round((Cmin + deltaC4) + (2 * deltaC4)*rand(1))];
                particle(partInd).vel=[round(Vmin+(Vmax-Vmin)*rand(1))
round(Vmin+(Vmax-Vmin)*rand(1))];
                particle(partInd).Pbest=particle(partInd).posn;

particle(partInd).func_response=cost_function7subReg(I,particle(partInd).posn(1),parti
cle(partInd).posn(2),dt,side,border, thresh_eye, thresh_light);

particle(partInd).func_resp_prev=particle(partInd).func_response;
            end
        end
    else
        % if we didn't find a positive fitness value before the
        % fifth iteration that the region for deployment is
        % extended.
        % it helps in the cases, when the face region is not
        % located well
        for g = 0:(group-1)
            for i=1:n
                partInd = g*n + i;
                particle(partInd).posn = [round((Rmin + deltaR2)+
deltaR2*rand(1)) round(Cmin + deltaC*rand(1))];
                particle(partInd).vel = [round(Vmin+(Vmax-Vmin)*rand(1))
round(Vmin+(Vmax-Vmin)*rand(1))];
                particle(partInd).Pbest = particle(partInd).posn;
            end
        end
    end
end

```

```

        particle(partInd).func_response = cost_function7subReg(I,
particle(partInd).posn(1), particle(partInd).posn(2), dt, side, border, thresh_eye,
thresh_light);

        particle(partInd).func_resp_prev =
particle(partInd).func_response;
            end
        end
    end

    bestfind = find([particle((g*n + 1):(g*n + n)).func_response] ==
max([particle((g*n + 1):(g*n + n)).func_response]));
    best(g+1) = g*n + bestfind(1);
    Gbest((2*g + 1):(2*g + 2)) = particle(best(g+1)).posn;
    GbestFit(g+1) = particle(best(g+1)).func_response;
    MaxCord((2*g + 1):(2*g + 2)) = Gbest((2*g + 1):(2*g + 2));
    best_response(g+1) = particle(best(g+1)).func_response;
    MaxFit(g+1) = best_response(g+1);
    t=1;
    thresh_list=[];
end % all particles updated...

%...recording co-ordinates for display
itr = itr+1;
%display(best_response);
thresh_list=[thresh_list best_response];
if itr == 20
    ind = 1;
end
end

% choose the best point
very_best_ones = find( GbestFit == max(GbestFit));
very_best = very_best_ones(1);
VeryBest = Gbest(2*(very_best - 1) + 1: 2*(very_best - 1) + 2);

```

Filename : cost_function7subReg.m

This is the cost function used to locate the eyes.

```
function y = cost_function7subReg(I,r,c,dtIn,side,border, thresh_eye,
thresh_light)
% new cost function for mouth
%8:09 pm 12/26/04

global Im;
Im = I;
global dt;
dt = dtIn;
global de;
de = floor(dt/8);
rt1 = r + floor(dt/4);
ct1 = c;
rt2 = rt1 + de;
ct2 = c;
rt3 = rt2 + de;
ct3 = c;
rt4 = rt3 + de;
ct4 = c;

y = Cm(r,c) + CcPack1(r,c,rt1,ct1,rt2,ct2,rt3,ct3,rt4,ct4, thresh_eye,
thresh_light) + CMean(r,c) + Cout(r, c, side, border);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function a = Cm(r,c)
global Im;
global dt;
Kc=0.2;
I1=double(Im( (r-floor(dt/2)) : (r+ floor(dt/2)), (c-floor(dt/2)) : (c+
floor(dt/2))));
var1=sum(std(I1));
var2=sum(std(I1'));
a=Kc*(var1+var2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function a = CMean(r,c)
global Im;
```

```

global dt;
I1=double(Im( (r - floor(dt / 6) : r + floor(dt / 6)), c - floor(dt/6) : c +
floor(dt/6)));
a = 255 - mean2(I1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function c=U(x)
% Unit step function.....
if x >=0
    c=1;
else
    c=0;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function a = CcPack(r,c,rt1,ct1,rt2,ct2,rt3,ct3,rt4,ct4, thresh_eye,
thresh_light)
global Im;
K = 50;
if (U(double(Im(rt1,ct1)) - double(Im(r,c)) - K) * U(double(Im(rt2,ct2)) -
double(Im(r,c)) - K) * U(double(Im(rt3,ct3)) - double(Im(r,c)) - K) *
U(double(Im(rt4,ct4)) - double(Im(r,c)) - K)) == 1
    a = Cc(r,c, thresh_eye, thresh_light);
else
    a = -800;
end

function a = CcPack1(r,c,rt1,ct1,rt2,ct2,rt3,ct3,rt4,ct4, thresh_eye,
thresh_light)
global Im;
global dt;
K = 50;

% Mean intensities of subregions that we will compare with the center
% region to determine whether it is eyebrow or not. We are looking at the
% region below the center point and then on the region to the lower left
% and lower right - it is because of the inner corner of eyebrow.

uc = mean2(Im(r - floor(dt/15):r + floor(dt/15), c - floor(dt/15): c +
floor(dt/15)));
u11 = mean2(Im(rt1 - floor(dt/10):rt1 + floor(dt/10),...
ct1 - floor(dt/3) - floor(dt/10): ct1 - floor(dt/3) + floor(dt/10)));

```

```

uc1 = mean2(Im(rt1 - floor(dt/10):rt1 + floor(dt/10), ct1 - floor(dt/10): ct1 +
floor(dt/10)));
ur1 = mean2(Im(rt1 - floor(dt/10):rt1 + floor(dt/10),...
ct1 + floor(dt/3) - floor(dt/10): ct1 + floor(dt/3) + floor(dt/10)));
ul2 = mean2(Im(rt2 - floor(dt/10):rt2 + floor(dt/10),...
ct2 - floor(dt/3) - floor(dt/10): ct2 + floor(dt/3) + floor(dt/10)));
uc2 = mean2(Im(rt2 - floor(dt/10):rt2 + floor(dt/10), ct2 - floor(dt/10): ct2 +
floor(dt/10)));
ur2 = mean2(Im(rt2 - floor(dt/10):rt2 + floor(dt/10),...
ct2 + floor(dt/3) - floor(dt/10): ct2 + floor(dt/3) + floor(dt/10)));
ul3 = mean2(Im(rt3 - floor(dt/10):rt3 + floor(dt/10),...
ct3 - floor(dt/3) - floor(dt/10): ct3 - floor(dt/3) + floor(dt/10)));
uc3 = mean2(Im(rt3 - floor(dt/10):rt3 + floor(dt/10), ct3 - floor(dt/10): ct3 +
floor(dt/10)));
ur3 = mean2(Im(rt3 - floor(dt/10):rt3 + floor(dt/10),...
ct3 + floor(dt/3) - floor(dt/10): ct3 + floor(dt/3) + floor(dt/10)));
ul4 = mean2(Im(rt4 - floor(dt/10):rt4 + floor(dt/10),...
ct4 - floor(dt/3) - floor(dt/10): ct4 - floor(dt/3) + floor(dt/10)));
uc4 = mean2(Im(rt4 - floor(dt/10):rt4 + floor(dt/10), ct4 - floor(dt/10): ct4 +
floor(dt/10)));
ur4 = mean2(Im(rt4 - floor(dt/10):rt4 + floor(dt/10),...
ct4 + floor(dt/3) - floor(dt/10): ct4 + floor(dt/3) + floor(dt/10)));

check1 = 0;
check2 = 0;
check3 = 0;
check4 = 0;

if (U(ul1 - uc - K) * U(uc1 - uc - K) * U(ur1 - uc - K) == 1)
    check1 = 1;
end
if (U(ul2 - uc - K) * U(uc2 - uc - K) * U(ur2 - uc - K) == 1)
    check2 = 1;
end
if (U(ul3 - uc - K) * U(uc3 - uc - K) * U(ur3 - uc - K) == 1)
    check3 = 1;
end
if (U(ul4 - uc - K) * U(uc4 - uc - K) * U(ur4 - uc - K) == 1)
    check4 = 1;
end

if check1*check2*check3*check4 == 1

```

```

    a = Cc(r,c, thresh_eye, thresh_light);
else
    a = -800;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function a=Cc(r,c,thresh_eye, thresh_light)
global dt;
global Im;
Kc=0;
a=0;
I1=Im( (r - floor(dt/2)) : (r + floor(dt/2)), (c - floor(dt/2)) : (c +
floor(dt/2)));
    % arbitrary constant
    uu=mean2(Im((r- floor(dt/2)):(r- floor(dt/4)),(c-floor(dt/2)):(c+
floor(dt/2))));
    uc=mean2(Im((r- floor(dt/10)):(r+ floor(dt/10)),(c- floor(dt/10)):(c+
floor(dt/10))));
    ul=mean2(Im((r+ floor(dt/4)):(r+ floor(dt/2)),(c- floor(dt/2)):(c+
floor(dt/2))));

    % intensity of midpoint less than 50
    % we need to change this dynamically based on the output of the slider,
    % need to make it lower for brighter eyes, or more for darker eyes
    % if uc<=50
    if uc <= thresh_eye
        Kc=1.2*(50-uc);
    end
    % lower edge region darker than 128
    % we need to change this based on lighting conditions.
    % increase for bad lighting and decrease for good lighting.
    % if ul<=128
    if ul<=(64/thresh_light)
        a=a-200;
    end
    % upper edge region darker than 128
    % we need to change this based on lighting conditions.
    % increase for bad lighting and decrease for good lighting.
    % if uu<=128
    if uu<=(64/thresh_light)
        a=a-200;
    end
end

```



```

% difernce in midpoint and upper point intensities is less than 50
% if uu-uc <50
if uu-uc < 50 %thresh_eye
    a=a-200;
end
% difernce in midpoint and lower point intensities is less than 50
% if ul-uc <50
if ul-uc < 50 %thresh_eye
    a=a-200;
end

a=a+Kc+uu+ul-2*uc;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function a = Cout(r, c, side, border)
% function discriminating points outside of the eye subregion ( it is
% supposed to prevent the algorithm from chosing hair when they are included
% in the face region)

a = 0;
if side == 0
    if c < border
        a = -800;
    end
else
    if c > border
        a = -800;
    end
end
end

```

Filename : getRightEyeCorners.m

This extracts the eye corners of the right eye based upon the location of the eye.

```
function right_eye_corners = getRightEyeCorners(right_eye,Image,dt,
thresh_eyeInner)

EyeRegionRRmin = right_eye(1) - floor(dt/5);
EyeRegionRRmax = right_eye(1) + floor(dt/4);
EyeRegionRCmin = right_eye(2) - floor(dt / 2);
EyeRegionRCmax = right_eye(2) + floor(2 * (dt/3));

%estimated eye region
Image_Eye = Image(EyeRegionRRmin : EyeRegionRRmax, EyeRegionRCmin :
EyeRegionRCmax);

% convert the right eye region into binary image with the threshold set
% to the 1/2 of the mean intensity of whole region
level = mean2(Image_Eye) / 255;
level = (0.8 * level) * thresh_eyeInner;
Image_Eye_BW = im2bw(Image_Eye,level);

% invert the colors
Image_Ones = ones(size(Image_Eye));
Image_Eye_BW = Image_Ones - Image_Eye_BW;

% get a bounding box around the thresholded area
statsE = regionprops(Image_Eye_BW,'BoundingBox');
cordEye = [statsE.BoundingBox];

% get a region aroud the inner corner of the right eyebrow
Image_Eye_Corner_In_BW = Image_Eye_BW(1 : (EyeRegionRRmax - EyeRegionRRmin),
ceil(cordEye(1)) : ceil(cordEye(1) + 1));

Eye_Corner_In_Stat = bwlabel(Image_Eye_Corner_In_BW);
statsEIC = regionprops(Eye_Corner_In_Stat,'BoundingBox');
cordEyeInCorner = [statsEIC.BoundingBox];

% get a region aroud the outter corner of the right eye
Image_Eye_Corner_Out_BW = Image_Eye_BW(1 : (EyeRegionRRmax - EyeRegionRRmin),
floor(cordEye(1) + cordEye(3) -1) : floor(cordEye(1) + cordEye(3)));
```

```

Eye_Corner_Out_Stat = bwlabel(Image_Eye_Corner_Out_BW);
statsEOC = regionprops(Eye_Corner_Out_Stat, 'BoundingBox');
cordEyeOutCorner = [statsEOC.BoundingBox];

%Check if the corner point was found. If not a default value is assigned
%and an error is reported
if numel(cordEyeOutCorner) == 0
    cordEyeOutCorner = [0 (right_eye(2) - EyeRegionRRmin) + round(dt / 4) 0 0];
    display('Outer corner of the right eye was not located right...');
end

if numel(cordEyeInCorner) == 0
    cordEyeInCorner = [0 (right_eye(2) - EyeRegionRRmin) - round(dt / 4) 0 0];
    display('Inner corner of the right eye was not located right...');
end

right_eye_corners = [floor(EyeRegionRRmin + cordEyeInCorner(2) +
cordEyeInCorner(4) / 2) floor(EyeRegionRCmin + cordEye(1))...
    floor(EyeRegionRRmin + cordEyeOutCorner(2) + cordEyeOutCorner(4) /
2) floor(EyeRegionRCmin + cordEye(1) + cordEye(3))];

```

Filename : Pso_eyebrow.m

This is the particle swarm optimization algorithm used to locate the eyebrow.

```
function [Gbest M]=Pso_eyebrow(Rmin,Rmax,Cmin,Cmax,I,width)
% PSO algorithm for extracting the eyebrow coordinates

n=5;%no of particles
Vmax= 10; % Max velocity..set arbitrarily
Vmin= 3; % particle move one pixel at a time at minimum
Gbest=[0 0];
GbestFit = 0;
MaxFit = 0;
MaxCord = [0 0];
wi = 0.8; %inertial weight
c1=0.5;
c2=0.5;

ind=0;
coord=[];
prev_best_values=[];
threshold=400;
dt = floor(width/4);

%initialising particles with random velocities and setting
%Pbest to the initial position
%Deploying the particles into an region where the eye is expected.

deltaR2 = round((Rmax - Rmin) / 2);
deltaC = Cmax - Cmin;
deltaR = Rmax - Rmin;

% initialising particles for PSO.
for i=1:n
    particle(i).posn=[round(Rmin + deltaR*rand(1)) round(Cmin +
deltaC*rand(1))];
    particle(i).vel=[round(Vmin+(Vmax-Vmin)*rand(1)) round(Vmin+(Vmax-
Vmin)*rand(1))];
    particle(i).Pbest=particle(i).posn;
```

```

particle(i).func_response=cost_functionEyebrow(I,particle(i).posn(1),particle(i).posn(
2),dt);
    particle(i).func_resp_prev=particle(i).func_response;
end % finished initializing

% Finding initial best values
bestfind = find([particle.func_response] == max([particle.func_response]));
best = bestfind(1);
Gbest=particle(best).posn;
GbestFit = particle(best).func_response;

itr=1;
t=1;
thresh_list=[];

while ind == 0 && itr < 22

    %Compute particles new position and velocities
    % if GbestFit > 0 compute new position and velocities - algorithm
    % is converging.
    if (GbestFit > 0)
        for i=1:n

particle(i).vel(1)=wi*particle(i).vel(1)+c1*rand(1)*(particle(i).Pbest(1)-
particle(i).posn(1)) + c2*rand(1)*(Gbest(1)-particle(i).posn(1));

particle(i).vel(2)=wi*particle(i).vel(2)+c1*rand(1)*(particle(i).Pbest(2)-
particle(i).posn(2)) + c2*rand(1)*(Gbest(2)-particle(i).posn(2));

            % set new position
            particle(i).posn(1)=round(particle(i).posn(1)+particle(i).vel(1));
            particle(i).posn(2)=round(particle(i).posn(2)+particle(i).vel(2));

            %checking if the solution lies within domain
            if particle(i).posn(1) < Rmin || particle(i).posn(1) > Rmax ||
particle(i).posn(2) < Cmin || particle(i).posn(2) > Cmax
                particle(i).posn=particle(i).Pbest;    % setting position to
previous P-best
                particle(i).vel=[rand(1)*Vmax rand(1)*Vmax];    %
setting velocity to max
            end
        end
    end
end

```

```

        %Calculating the function response for each particle with new
        %positions

particle(i).func_response=cost_functionEyebrow(I,particle(i).posn(1),particle(i).posn(
2),dt);

        %checking and updating Pbest
        if particle(i).func_response > particle(i).func_resp_prev
            particle(i).Pbest=particle(i).posn;
        end

        particle(i).func_resp_prev = particle(i).func_response;

    end

    V=[particle.func_response];
    best_ones=find(V==max(V));
    best=best_ones(1);

    %Gbest is a global best of all times
    %display(particle(best).func_response);
    if particle(best).func_response > GbestFit
        Gbest = particle(best).Pbest;
        GbestFit = particle(best).func_response;
    end
else
    % if is not > 0 than deploy te particles again in new random
    % positions and with new random speeds
    if (itr < 5)
        for i=1:n
            particle(i).posn=[round((Rmin + deltaR2)+ deltaR2*rand(1))
round(Cmin + deltaC*rand(1))];
            particle(i).vel=[round(Vmin+(Vmax-Vmin)*rand(1))
round(Vmin+(Vmax-Vmin)*rand(1))];
            particle(i).Pbest=particle(i).posn;

particle(i).func_response=cost_functionEyebrow(I,particle(i).posn(1),particle(i).posn(
2),dt);

            particle(i).func_resp_prev=particle(i).func_response;
        end
    else
        % if we didn't find a positive fitness value before the
        % fifth iteration that the region for deployment is

```

```

% extended.
% it helps in the cases, when the face region is not
% located well
    for i=1:n
        particle(i).posn=[round(Rmin + deltaR*rand(1)) round(Cmin +
deltaC*rand(1))];
        particle(i).vel=[round(Vmin+(Vmax-Vmin)*rand(1))
round(Vmin+(Vmax-Vmin)*rand(1))];
        particle(i).Pbest=particle(i).posn;

particle(i).func_response=cost_functionEyebrow(I,particle(i).posn(1),particle(i).posn(
2),dt);
        particle(i).func_resp_prev=particle(i).func_response;
    end
end
bestfind = find([particle.func_response] ==
max([particle.func_response]));
best = bestfind(1);
Gbest=particle(best).posn;
GbestFit = particle(best).func_response;
MaxCord = Gbest;
best_response=particle(best).func_response;
MaxFit = best_response;
t=1;
end % all particles updated...

% 2 - based on the mean value of func_response of all particles
if mean([particle.func_response]) >= 1100;%threshold;
    ind = 1;
end

%...recording co-ordinates for display
itr = itr+1;
if itr == 20
    ind = 1;
end
end

if itr == 22
    Gbest = MaxCord;
end

```

Filename : cost_functionEyebrow.m

This is the cost function that is used to locate the eyebrow.

```
function y = cost_functionEyebrow(I,r,c,dtIn)
% new cost function for Eyebrow
%8:09 pm 12/26/04

global Im;
Im = I;
global dt;
dt = dtIn;
global de;
de = floor(dt/8);

% global variables that defines the border of the template
% they are checked and eventually reset to desired values, so that
% the template doesn't go out from the image

global ri; global rx; global ci; global cx;

y = CMean(r,c);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function a = Cm(r,c)
global Im;
global dt;
global ri;
global rx;
global ci;
global cx;
Kc=0.2;

ri = r - floor(dt/2);
if ri < 1
    ri = 1;
end

rx = r + floor(dt/2);
if rx > 255
    rx = 255;
```



```

end

ci = c - floor(dt/2);
if ci < 1
    ci = 1;
end

cx = c + floor(dt/2);
if cx > 255
    cx = 255;
end

I1 = double(Im( ri : rx, ci : cx));
var1=sum(std(I1));
var2=sum(std(I1'));
a=Kc*(var1+var2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function a = CMean(r,c)
global Im;
global dt;
I1=double(Im( (r - floor(dt / 6) : r + floor(dt / 6)), c - floor(dt/3) : c +
floor(dt/3)));
a = 255 - mean2(I1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function c=U(x)
% Unit step function.....
if x >=0
    c=1;
else
    c=0;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function a=Cc(r,c)

global dt;
global Im;
global ri;
global rx;
global ci;

```

```

global cx;
Kc=0;
a=0;
I1=Im( ri : rx, ci : cx);
% arbitrary constant
uu=mean2(Im(ri:(r- floor(dt/4)), ci:cx));
uc=mean2(Im((r- floor(dt/10)):(r+ floor(dt/10)),(c- floor(dt/10)):(c+
floor(dt/10))));
ul=mean2(Im((r+ floor(dt/4)):(r+ floor(dt/2)),ci:cx));

% intensity of midpoint less than 50
if uc<=thresh_eb
    Kc=1.2*(50-uc);
else
    a = a -500;
end
% lower edge region darker than 128
if ul<=(256*thresh_light)
    a=a-200;
end
% upper edge region darker than 128
if uu<=(256*thresh_light)
    a=a-200;
end
% difernce in midpoint and upper point intensities is less than 50
if uu-uc <(100-thresh_eb)
    a=a-200;
end
% difernce in midpoint and lower point intensities is less than 50
if ul-uc < (100-thresh_eb)
    a=a-200;
end

a=a+Kc+uu+ul-2*uc;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function a = CPos(r,c)
% function saying the the higher the region is, the better.

    a = 300 - 2 * r;

```

Filename : getRightEyebrowCorner.m

This locates the inner corner of the right eyebrow.

```
function right_eyebrow_corner = getRightEyebrowCorner(right_eyebrow, Image, dt,
thresh_ebCorner)

EBCornerRRmin = right_eyebrow(1) - floor(dt/4);
EBCornerRRmax = right_eyebrow(1) + floor(dt/3);
EBCornerRCmin = right_eyebrow(2) - dt;
EBCornerRCmax = right_eyebrow(2);

Image_Eyebrow = Image(EBCornerRRmin : EBCornerRRmax, EBCornerRCmin :
EBCornerRCmax);

% convert the right eyebrow region into binary image with the threshold set
% to the 1/2 of the mean intensity of whole region
level = mean2(Image_Eyebrow) / 255;
level = (level / 2) * thresh_ebCorner;
Image_Eyebrow_BW = im2bw(Image_Eyebrow, level);

% invert the colors
Image_Ones = ones(size(Image_Eyebrow));
Image_Eyebrow_BW = Image_Ones - Image_Eyebrow_BW;

% get a bounding box around the thresholded area
statsEB = regionprops(Image_Eyebrow_BW, 'BoundingBox');
cordEyebrow = [statsEB.BoundingBox];

if numel(cordEyebrow) == 0
    display('Unable to locate the right eyebrow')
    right_eyebrow_corner = [right_eyebrow(1) + round(dt/8) right_eyebrow(2) -
round(dt/4)];
else
    % get a region aroud the inner corner of the right eyebrow\

    % check so that matrix index does not exceeds the matrix dimension

    if cordEyebrow(1) > (EBCornerRCmax - EBCornerRCmin)
        Image_Eyebrow_Corner_BW = Image_Eyebrow_BW(1 : (EBCornerRRmax -
EBCornerRRmin - 1), ceil(cordEyebrow(1) - 1) : ceil(cordEyebrow(1)));
```

```

else
    Image_Eyebrow_Corner_BW = Image_Eyebrow_BW(1 : (EBCornerRRmax -
EBCornerRRmin - 1), ceil(cordEyebrow(1)) : ceil(cordEyebrow(1) + 1));
end

statsEBC = regionprops(Image_Eyebrow_Corner_BW, 'BoundingBox');
cordEyebrowCorner = [statsEBC.BoundingBox];

if (numel(cordEyebrowCorner) == 0)
    display('Unable to locate the right eyebrow corner');
    right_eyebrow_corner = [right_eyebrow(1) + round(dt/8) right_eyebrow(2)
- round(dt/4)];
else
    right_eyebrow_corner = [floor(EBCornerRRmin + cordEyebrowCorner(2)
+ cordEyebrowCorner(4) / 2) floor(EBCornerRCmin + cordEyebrow(1))];
end
end
end

```

Filename : getLeftEyeCorners.m

This locates the corners of the left eye.

```
function left_eye_corners = getLeftEyeCorners(left_eye,Image,dt,
thresh_eyeInner)

load 'threshold.mat';

EyeRegionLRmin = left_eye(1) - floor(dt/5);
EyeRegionLRmax = left_eye(1) + floor(dt/4);
EyeRegionLCmin = left_eye(2) - floor(2 * (dt/3));
EyeRegionLCmax = left_eye(2) + floor(dt / 2);

Image_Eye = Image(EyeRegionLRmin : EyeRegionLRmax, EyeRegionLCmin :
EyeRegionLCmax);

% convert the left eye region into binary image with the threshold set
% to the mean intensity of whole region
level = mean2(Image_Eye) / 255;
level = (0.8 * level) * thresh_eyeInner;
Image_Eye_BW = im2bw(Image_Eye,level);

% invert the colors
Image_Ones = ones(size(Image_Eye));
Image_Eye_BW = Image_Ones - Image_Eye_BW;

% get a bounding box around the thresholded area
statsE = regionprops(Image_Eye_BW,'BoundingBox');
cordEye = [statsE.BoundingBox];

% get a region around the inner corner of the left eye
Image_Eye_Corner_In_BW = Image_Eye_BW(1 : (EyeRegionLRmax - EyeRegionLRmin),
floor(cordEye(1) + cordEye(3) - 1) : floor(cordEye(1)) + cordEye(3));

Eye_Corner_In_Stat = bwlable(Image_Eye_Corner_In_BW);
statsEIC = regionprops(Eye_Corner_In_Stat,'BoundingBox');
cordEyeInCorner = [statsEIC.BoundingBox];

% get a region around the outer corner of the left eye
```

```

Image_Eye_Corner_Out_BW = Image_Eye_BW(1 : (EyeRegionLRmax - EyeRegionLRmin),
ceil(cordEye(1)) : ceil(cordEye(1) + 1));

Eye_Corner_Out_Stat = bwlabel(Image_Eye_Corner_Out_BW);
statsEOC = regionprops(Eye_Corner_Out_Stat, 'BoundingBox');
cordEyeOutCorner = [statsEOC.BoundingBox];

%Check if the corner point was found. If not a default value is assigned
%and an error is reported

if numel(cordEyeOutCorner) == 0
    cordEyeOutCorner = [0 left_eye(2) - round(dt / 4) 0 0];
    display('Outer corner of the left eye was not located right...');
end

if numel(cordEyeInCorner) == 0
    cordEyeInCorner = [0 left_eye(2) + round(dt / 4) 0 0];
    display('Inner corner of the left eye was not located right...');
end

left_eye_corners = [floor(EyeRegionLRmin + cordEyeInCorner(2) +
cordEyeInCorner(4) / 2) floor(EyeRegionLCmin + cordEye(1) + cordEye(3))
floor(EyeRegionLRmin + cordEyeOutCorner(2) + cordEyeOutCorner(4) / 2)
floor(EyeRegionLCmin + cordEye(1))];

```

Filename : getLeftEyebrowCorner.m

This locates the inner corner of the left eyebrow.

```
function left_eyebrow_corner =
getLeftEyebrowCorner(left_eyebrow, Image, dt, thresh_ebCorner)

EBCornerLRmin = left_eyebrow(1) - floor(dt/4);
EBCornerLRmax = left_eyebrow(1) + floor(dt/3);
EBCornerLCmin = left_eyebrow(2);
EBCornerLCmax = left_eyebrow(2) + dt;

Image_Eyebrow = Image(EBCornerLRmin : EBCornerLRmax, EBCornerLCmin :
EBCornerLCmax);

% convert the right eyebrow region into binary image with the threshold set
% to the 1/2 of the mean intensity of whole region
level = mean2(Image_Eyebrow) / 255;
level = (level / 2) * thresh_ebCorner;
Image_Eyebrow_BW = im2bw(Image_Eyebrow, level);

% invert the colors
Image_Ones = ones(size(Image_Eyebrow));
Image_Eyebrow_BW = Image_Ones - Image_Eyebrow_BW;

% get a bounding box around the thresholded area
statsEB = regionprops(Image_Eyebrow_BW, 'BoundingBox');
cordEyebrow = [statsEB.BoundingBox];

if (numel(cordEyebrow) == 0)
    display('Unable to locate the Left eyebrow corner (1)');
    left_eyebrow_corner = [left_eyebrow(1) + round(dt/8) left_eyebrow(2) +
round(dt/4)];
else

    % get a region around the inner corner of the right eyebrow
    left = floor(cordEyebrow(1) + cordEyebrow(3) - 1);
    right = floor(cordEyebrow(1) + cordEyebrow(3));

    Image_Eyebrow_Corner_BW = Image_Eyebrow_BW(1 : (EBCornerLRmax -
EBCornerLRmin), left : right);
```

```

statsEBC = regionprops(Image_Eyebrow_Corner_BW, 'BoundingBox');
cordEyebrowCorner = [statsEBC.BoundingBox];

if (numel(cordEyebrowCorner) == 0)
    display('Unable to locate the Left eyebrow corner (2)');
    left_eyebrow_corner = [left_eyebrow(1) + round(dt/8) left_eyebrow(2) +
round(dt/4)];
    display(cordEyebrow(1));
    display(cordEyebrow(3));
else
    left_eyebrow_corner = [floor(EBCornerLRmin + cordEyebrowCorner(2) +
cordEyebrowCorner(4) / 2) floor(EBCornerLCmin + cordEyebrow(3))];
end
end
end

```


Filename : Pso2_mouth_left.m

This is the particle swarm optimization algorithm used to locate the left corner of the mouth.

```
function [VeryBest M]=Pso2_mouth_left(Rmin,Rmax,Cmin,Cmax,I,center, thresh_lip)

n = 5; % number of particles
group = 3; % number of groups
Vmax= 10; % Max velocity..set arbitrarily
Vmin= 3; % particle move one pixel at a time at minimum
Gbest=[0 0 0 0 0 0];
GbestFit = [0 0 0];
MaxFit = [0 0 0];
MaxCord = [0 0 0 0 0 0];
best_response = [0 0 0];
best = [0 0 0];
VeryBest = [0 0];

wi = 0.85; %inertial weight
c1=0.5;
c2=0.5;

ind=0;
coord=[];
prev_best_values=[];
threshold = 400;
dt = floor((Cmax - Cmin)/2);
meanInt = mean2(I(Rmin:Rmax,Cmin:Cmax));

%initialising particles with random velocities and setting
%Pbest to the initial posiion
%Deploying the particles into an region where the eye is expected.

deltaR4 = round((Rmax - Rmin) / 4);
deltaR2 = round((Rmax - Rmin) / 2);
deltaC2 = round((Cmax - Cmin) / 2);
deltaC3 = round((Cmax - Cmin) / 3);
deltaC = Cmax - Cmin;
deltaR = Rmax - Rmin;
```

```

% initializing particles in all groups
for g = 0:(group - 1)
    for i = 1:n
        partInd = g*n + i;
        particle(partInd).posn=[round((Rmin + deltaR4)+ deltaR2*rand(1))
round((Cmin + deltaC3) + (2*deltaC3)*rand(1))];
        particle(partInd).vel=[round(Vmin+(Vmax-Vmin)*rand(1))
round(Vmin+(Vmax-Vmin)*rand(1))];

        % check if the particle isn't too close to the lower edge of the
image,

        % so that the lower edge of the template would get out of the bounds.
if particle(partInd).posn(1) > (255 - (round(dt / 2)))
    particle(partInd).posn(1) = 255 - (round(dt / 2));
end

        particle(partInd).Pbest = particle(partInd).posn;
        particle(partInd).func_response =
cost_functionMouthL2(I,particle(partInd).posn(1),
particle(partInd).posn(2),dt,center,meanInt, thresh_lip);
        particle(partInd).func_resp_prev = particle(partInd).func_response;
    end

    bestfind = find([particle((g*n + 1):(g*n + n)).func_response] ==
max([particle((g*n + 1):(g*n + n)).func_response]));
    best(g+1) = g*n + bestfind(1);
    Gbest((2*g + 1):(2*g + 2)) = particle(best(g+1)).posn;
    GbestFit(g+1) = particle(best(g+1)).func_response;
    MaxCord((2*g + 1):(2*g + 2)) = Gbest((2*g + 1):(2*g + 2));
    best_response(g+1) = particle(best(g+1)).func_response;
    MaxFit(g+1) = best_response(g+1);
end

itr=1;
t=1;
thresh_list=[];

while ind == 0 && itr < 22
    %Compute particles new position and velocities

```

```

% if GbestFit > 0 compute new position and velocities - algorithm
% is converging.
if (GbestFit(g+1) > 0)
    %In each group
    for g = 0:(group-1)
        for i=1:n
            partInd = g*n + i;

particle(partInd).vel(1)=wi*particle(partInd).vel(1)+c1*rand(1)*(particle(partInd).Pbest(1) - particle(partInd).posn(1))+ c2*rand(1)*(Gbest(2*g + 1)-
particle(partInd).posn(1));

particle(partInd).vel(2)=wi*particle(partInd).vel(2)+c1*rand(1)*(particle(partInd).Pbest(2) - particle(partInd).posn(2))+ c2*rand(1)*(Gbest(2*g + 2)-
particle(partInd).posn(2));

particle(partInd).posn(1)=round(particle(partInd).posn(1)+particle(partInd).vel(1));

particle(partInd).posn(2)=round(particle(partInd).posn(2)+particle(partInd).vel(2));

            % check if the particle isn't too close to the lower edge of
the image,
            % so that the lower edge of the template would get out of the
bounds.
            % this check might be together with the following check of
particle
            % position
            if particle(partInd).posn(1) > (255 - (round(dt / 2)))
                particle(partInd).posn(1) = 255 - (round(dt / 2));
            end

            %checking if the solution lies within domain
            if particle(partInd).posn(1) <Rmin || particle(partInd).posn(1)
> Rmax || particle(partInd).posn(2) <Cmin || particle(partInd).posn(2) > Cmax
                particle(partInd).posn = particle(partInd).Pbest;    %
setting position to previous P-best
                particle(partInd).vel=[rand(1)*Vmax rand(1)*Vmax];
% setting velocity to max
            end

            %Calculating the function response for each particle with new

```

```

        %positions

particle(partInd).func_response=cost_functionMouthL2(I,particle(partInd).posn(1),
particle(partInd).posn(2),dt,center,meanInt, thresh_lip);

        %checking and updating Pbest
        if particle(partInd).func_response >
particle(partInd).func_resp_prev
            particle(partInd).Pbest = particle(partInd).posn;
        end

        particle(partInd).func_resp_prev =
particle(partInd).func_response;

    end
end

V = [particle((g*n + 1):(g*n + n)).func_response];
best_ones = find(V == max(V));
best(g+1) = g*n + best_ones(1);

%Gbest is a global best of all times
if particle(best(g+1)).func_response > GbestFit(g+1)
    Gbest((2*g + 1):(2*g + 2)) = particle(best(g+1)).Pbest;
    GbestFit(g+1) = particle(best(g+1)).func_response;
end
best_response(g+1) = particle(best(g+1)).func_response;
if (best_response(g+1) < MaxFit(g+1))
    MaxCord((2*g + 1):(2*g + 2)) = Gbest((2*g + 1):(2*g + 2));
end

else
% if is not > 0 than deploy te particles again in new random
% positions and with new random speeds
    if (itr < 5)
        for g = 0:(group-1)
            for i=1:n
                partInd = g*n + i;
                particle(partInd).posn=[round((Rmin + deltaR4)+
deltaR2*rand(1)) round((Cmin + deltaC3) + (2*deltaC3)*rand(1))];
                particle(partInd).vel=[round(Vmin+(Vmax-Vmin)*rand(1))
round(Vmin+(Vmax-Vmin)*rand(1))];
            end
        end
    end
end

```

```

of the image,
% check if the particle isn't too close to the lower edge
% so that the lower edge of the template would get out of
the bounds.
if particle(partInd).posn(1) > (255 - (round(dt / 2)))
    particle(partInd).posn(1) = 255 - (round(dt / 2));
end

particle(partInd).Pbest=particle(partInd).posn;

particle(partInd).func_response=cost_functionMouthL2(I,particle(partInd).posn(1),
particle(partInd).posn(2),dt,center,meanInt, thresh_lip);

particle(partInd).func_resp_prev=particle(partInd).func_response;
    end
    end
else
% if we didn't find a positive fitness value before the
% fifth iteration that the region for deployment is
% extended.
% it helps in the cases, when the face region is not
% located well
    for g = 0:(group-1)
        for i=1:n
            partInd = g*n + i;
            particle(partInd).posn=[round(Rmin + deltaR*rand(1))
round((Cmin + deltaC3) + (2*deltaC3)*rand(1))];
            particle(partInd).vel=[round(Vmin+(Vmax-Vmin)*rand(1))
round(Vmin+(Vmax-Vmin)*rand(1))];

% check if the particle isn't too close to the lower edge
of the image,
% so that the lower edge of the template would get out of
the bounds.
if particle(partInd).posn(1) > (255 - (round(dt / 2)))
    particle(partInd).posn(1) = 255 - (round(dt / 2));
end
particle(partInd).Pbest = particle(partInd).posn;
particle(partInd).func_response =
cost_functionMouthL2(I,particle(partInd).posn(1),
particle(partInd).posn(2),dt,center,meanInt, thresh_lip);

```

```

        particle(partInd).func_resp_prev =
particle(partInd).func_response;
        end
    end
end

bestfind = find([particle((g*n + 1):(g*n + n)).func_response] ==
max([particle((g*n + 1):(g*n + n)).func_response]));
best(g+1) = g*n + bestfind(1);
Gbest((2*g + 1):(2*g + 2)) = particle(best(g+1)).posn;
GbestFit(g+1) = particle(best(g+1)).func_response;
MaxCord((2*g + 1):(2*g + 2)) = Gbest((2*g + 1):(2*g + 2));
best_response(g+1) = particle(best(g+1)).func_response;
MaxFit(g+1) = best_response(g+1);
t=1;
thresh_list=[];

end % all particles updated...

%...recording co-ordinates for display

itr = itr+1;
thresh_list=[thresh_list best_response];
if itr == 20
    ind = 1;
end
end
end
% choose the best point
very_best_ones = find( GbestFit == max(GbestFit));
very_best = very_best_ones(1);
VeryBest = Gbest(2*(very_best - 1) + 1: 2*(very_best - 1) + 2);
GbestFit(very_best) = [];
Gbest(2*(very_best - 1) + 1: 2*(very_best - 1) + 2) = [];
very_best_ones = find( GbestFit == max(GbestFit));
very_best = very_best_ones(1);
VeryBest = [VeryBest Gbest(2*(very_best - 1) + 1: 2*(very_best - 1) + 2)];
GbestFit(very_best) = [];
Gbest(2*(very_best - 1) + 1: 2*(very_best - 1) + 2) = [];
VeryBest = [VeryBest Gbest];

```

Filename : cost_functionMouthL2.m

This is the cost function used to locate the left corner of the mouth.

```
function y = cost_functionMouthL2(I,r,c,dtIn,center,meanInt, thresh_lip)
% cost function for looking for left corner point of mouth
%8:09 pm 12/26/04

global Im;
Im = I;
global dt;
dt = dtIn;

y = CMouth(r,c,meanInt, thresh_lip);% + CMouthTemplate(r,c,meanInt);

% function calculating the mean intensity of all points within the template
function a = CMean(r,c)
global Im;
global dt;
I1=double(Im( r - floor(dt / 15) : r + floor(dt / 15), c - floor(dt / 7) :
c ));

a = (255 - mean2(I1));

function a = CMouth(r,c,meanInt, thresh_lip)
global Im;
global dt;

a = 0;

I1 = double(Im( (r - floor(dt / 10) : r + floor(dt / 10)), c : c +
floor(dt/2)));
I2 = double(Im( (r - floor(dt / 10) : r + floor(dt / 10)), c - floor(dt/4) :
c ));
I3 = double(Im( (r - floor(dt / 3) : r - floor(dt / 15)), c : c + floor(dt /
7)));
I4 = double(Im( (r + floor(dt / 15) : r + floor(dt / 3)), c : c + floor(dt /
7)));
I6 = double(Im( (r - floor(dt / 15) : r + floor(dt / 15)), c : c +
floor(dt/10) ));
```

```

I7 = double(Im( (r - floor(dt / 15) : r + floor(dt / 15)), c - floor(dt/10):
c ));

```

```

meanI1 = mean2(I1);
meanI2 = mean2(I2);
meanI3 = mean2(I3);
meanI4 = mean2(I4);
meanIc = CMean(r,c);
meanI6 = mean2(I6);
meanI7 = mean2(I7);
b = CMouthTemplate(r,c,meanInt);

```

```

if (meanI1 < 0.8 * meanInt * thresh_lip) && (meanI2 > 0.8 * meanInt *
thresh_lip)

```

```

    a = a + meanI2 - meanI1;
    a = a + meanIc;
    a = a + (meanI7 - meanI6);
    if ((meanI3 - meanIc) > 50 ) && ((meanI4 - meanIc) > 50)
        a = a + 100;
    end;

```

```

else

```

```

    a = a - 200;

```

```

end

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

function a = CMouthTemplate(r,c,meanInt)

```

```

% function trying to calcute if the region is really mouth. Calculates the
% avarage intensity of rectangular template

```

```

global Im;

```

```

global dt;

```

```

I1 = double(Im((r - floor(dt/6) : r + floor(dt/2)), c : c + dt ));

```

```

a = mean2(I1);

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

function a = CPos(r,c,center)

```

```

% function evaluating if the given point lies under the eye

```

```

a = (-2) * abs(center - c);

```


Filename : Pso2_mouth_right.m

This is the particle swarm optimization algorithm used to locate the right corner of the mouth.

```
function [VeryBest M]=Pso2_mouth_right(Rmin,Rmax,Cmin,Cmax,I,center)

n = 5; % number of particles
group = 3; % number of groups
Vmax= 10; % Max velocity..set arbitrarily
Vmin= 3; % particle move one pixel at a time at minimum
Gbest=[0 0 0 0 0 0];
GbestFit = [0 0 0];
MaxFit = [0 0 0];
MaxCord = [0 0 0 0 0 0];
best_response = [0 0 0];
best = [0 0 0];
VeryBest = [0 0];

wi = 0.85; %inertial weight
c1=0.5;
c2=0.5;

ind=0;
coord=[];
prev_best_values=[];
threshold = 400;
dt = floor((Cmax - Cmin)/2);
meanInt = mean2(I(Rmin:Rmax,Cmin:Cmax));

%initialising particles with random velocities and setting
%Pbest to the initial posiion
%Deploying the particles into an region where the eye is expected.

deltaR4 = round((Rmax - Rmin) / 4);
deltaR2 = round((Rmax - Rmin) / 2);
deltaC2 = round((Cmax - Cmin) / 2);
deltaC3 = round((Cmax - Cmin) / 3);
deltaC = Cmax - Cmin;
deltaR = Rmax - Rmin;
```

```

% initializing particles in all groups
for g = 0:(group - 1)
    for i = 1:n
        partInd = g*n + i;
        particle(partInd).posn=[round((Rmin + deltaR4)+ deltaR2*rand(1))
round(Cmin + (2*deltaC3)*rand(1))];
        particle(partInd).vel=[round(Vmin+(Vmax-Vmin)*rand(1))
round(Vmin+(Vmax-Vmin)*rand(1))];

        % check if the particle isn't too close to the lower edge of the
image,

        % so that the lower edge of the template would get out of the bounds.
if particle(partInd).posn(1) > (255 - (round(dt / 2)))
    particle(partInd).posn(1) = 255 - (round(dt / 2));
end

        particle(partInd).Pbest = particle(partInd).posn;
        particle(partInd).func_response =
cost_functionMouthR2(I,particle(partInd).posn(1),
particle(partInd).posn(2),dt,center,meanInt);
        particle(partInd).func_resp_prev = particle(partInd).func_response;
    end

    bestfind = find([particle((g*n + 1):(g*n + n)).func_response] ==
max([particle((g*n + 1):(g*n + n)).func_response]));
    best(g+1) = g*n + bestfind(1);
    Gbest((2*g + 1):(2*g + 2)) = particle(best(g+1)).posn;
    GbestFit(g+1) = particle(best(g+1)).func_response;
    MaxCord((2*g + 1):(2*g + 2)) = Gbest((2*g + 1):(2*g + 2));
    best_response(g+1) = particle(best(g+1)).func_response;
    MaxFit(g+1) = best_response(g+1);
end

itr=1;
t=1;
thresh_list=[];

while ind == 0 && itr < 22

    %Compute particles new position and velocities

    % if GbestFit > 0 compute new position and velocities - algorithm

```

```

% is converging.
if (GbestFit(g+1) > 0)
    %In each group
    for g = 0:(group-1)
        for i=1:n
            partInd = g*n + i;

particle(partInd).vel(1)=wi*particle(partInd).vel(1)+c1*rand(1)*(particle(partInd).Pbest(1)...
- particle(partInd).posn(1))+ c2*rand(1)*(Gbest(2*g + 1)-
particle(partInd).posn(1));

particle(partInd).vel(2)=wi*particle(partInd).vel(2)+c1*rand(1)*(particle(partInd).Pbest(2) -
particle(partInd).posn(2))+ c2*rand(1)*(Gbest(2*g + 2)-
particle(partInd).posn(2));

particle(partInd).posn(1)=round(particle(partInd).posn(1)+particle(partInd).vel(1));

particle(partInd).posn(2)=round(particle(partInd).posn(2)+particle(partInd).vel(2));

% check if the particle isn't too close to the lower edge of
the image,
% so that the lower edge of the template would get out of the
bounds.
% this check might be together with the following check of
particle
% position
if particle(partInd).posn(1) > (255 - (round(dt / 2)))
    particle(partInd).posn(1) = 255 - (round(dt / 2));
end

%checking if the solution lies within domain
if particle(partInd).posn(1) <Rmin || particle(partInd).posn(1)
> Rmax || particle(partInd).posn(2) <Cmin || particle(partInd).posn(2) > Cmax
    particle(partInd).posn = particle(partInd).Pbest; %
setting position to previous P-best
    particle(partInd).vel=[rand(1)*Vmax rand(1)*Vmax];
% setting velocity to max
end

%Calculating the function response for each particle with new

```

```

        %positions

particle(partInd).func_response=cost_functionMouthR2(I,particle(partInd).posn(1),
particle(partInd).posn(2),dt,center,meanInt);

        %checking and updating Pbest
        if particle(partInd).func_response >
particle(partInd).func_resp_prev
            particle(partInd).Pbest = particle(partInd).posn;
        end

        particle(partInd).func_resp_prev =
particle(partInd).func_response;

    end
end

V = [particle((g*n + 1):(g*n + n)).func_response];
best_ones = find(V == max(V));
best(g+1) = g*n + best_ones(1);

% Gbest is a global best of all times
% display(particle(best(g+1)).func_response);
if particle(best(g+1)).func_response > GbestFit(g+1)
    Gbest((2*g + 1):(2*g + 2)) = particle(best(g+1)).Pbest;
    GbestFit(g+1) = particle(best(g+1)).func_response;
end
best_response(g+1) = particle(best(g+1)).func_response;
if (best_response(g+1) < MaxFit(g+1))
    MaxCord((2*g + 1):(2*g + 2)) = Gbest((2*g + 1):(2*g + 2));
end

else
% if is not > 0 than deploy te particles again in new random
% positions and with new random speeds
    if (itr < 5)
        for g = 0:(group-1)
            for i=1:n
                partInd = g*n + i;

                particle(partInd).posn=[round((Rmin + deltaR4)+
deltaR2*rand(1)) round(Cmin + (2*deltaC3)*rand(1))];

```

```

        particle(partInd).vel=[round(Vmin+(Vmax-Vmin)*rand(1))
round(Vmin+(Vmax-Vmin)*rand(1))];

        % check if the particle isn't too close to the lower edge
of the image,
        % so that the lower edge of the template would get out of
the bounds.

        if particle(partInd).posn(1) > (255 - (round(dt / 2)))
            particle(partInd).posn(1) = 255 - (round(dt / 2));
        end

        particle(partInd).Pbest=particle(partInd).posn;

particle(partInd).func_response=cost_functionMouthR2(I,particle(partInd).posn(1),
particle(partInd).posn(2),dt,center,meanInt);

particle(partInd).func_resp_prev=particle(partInd).func_response;
        end
        end
    else
        % if we didn't find a positive fitness value before the
        % 5th iteration that the region for deployment is
        % extended.
        % it helps in the cases, when the face region is not
        % located well
        for g = 0:(group-1)
            for i=1:n
                partInd = g*n + i;

                particle(partInd).posn = [round(Rmin + deltaR*rand(1))
round(Cmin + (2*deltaC3)*rand(1))];
                particle(partInd).vel = [round(Vmin+(Vmax-Vmin)*rand(1))
round(Vmin+(Vmax-Vmin)*rand(1))];

                % check if the particle isn't too close to the lower edge
of the image,
                % so that the lower edge of the template would get out of
the bounds.

                if particle(partInd).posn(1) > (255 - (round(dt / 2)))
                    particle(partInd).posn(1) = 255 - (round(dt / 2));
                end

```

```

        particle(partInd).Pbest = particle(partInd).posn;
        particle(partInd).func_response =
cost_functionMouthR2(I,particle(partInd).posn(1),particle(partInd).posn(2),dt,center,m
eanInt);

        particle(partInd).func_resp_prev =
particle(partInd).func_response;
            end
        end
    end

    bestfind = find([particle((g*n + 1):(g*n + n)).func_response] ==
max([particle((g*n + 1):(g*n + n)).func_response]));
    best(g+1) = g*n + bestfind(1);
    Gbest((2*g + 1):(2*g + 2)) = particle(best(g+1)).posn;
    GbestFit(g+1) = particle(best(g+1)).func_response;
    MaxCord((2*g + 1):(2*g + 2)) = Gbest((2*g + 1):(2*g + 2));
    best_response(g+1) = particle(best(g+1)).func_response;
    MaxFit(g+1) = best_response(g+1);
    t=1;
    thresh_list=[];
end % all particles updated...

%...recording co-ordinates for display
itr = itr+1;
thresh_list=[thresh_list best_response];
if itr == 20
    ind = 1;
end
end

% choose the best point
very_best_ones = find( GbestFit == max(GbestFit));
very_best = very_best_ones(1);
VeryBest = Gbest(2*(very_best - 1) + 1: 2*(very_best - 1) + 2);
GbestFit(very_best) = [];
Gbest(2*(very_best - 1) + 1: 2*(very_best - 1) + 2) = [];
very_best_ones = find( GbestFit == max(GbestFit));
very_best = very_best_ones(1);
VeryBest = [VeryBest Gbest(2*(very_best - 1) + 1: 2*(very_best - 1) + 2)];
GbestFit(very_best) = [];
Gbest(2*(very_best - 1) + 1: 2*(very_best - 1) + 2) = [];
VeryBest = [VeryBest Gbest];

```

Filename : mouth_fix.m

This used a neural network to check if the right and the left corner of the mouth are located correctly, or else it relocates the mouth corners to a more appropriate solution.

```
function a = mouth_fix(ml,mr,net)
% function looking at all found mouth coordinates and trying to decide
% which pair is the right one

% mr2 - ml1
vector = [(mr(3) - ml(1)) (mr(4) - ml(2))];
vector = vector';
Y = sim(net,vector);
if Y == [1;0]
    display('1');
    a = [ml(1) ml(2) mr(3) mr(4)];
else
    %mr1 - ml2
    vector = [(mr(1) - ml(3)) (mr(2) - ml(4))];
    vector = vector';
    Y = sim(net,vector);
    if Y == [1;0]
        display('2');
        a = [ml(3) ml(4) mr(1) mr(2)];
    else
        %mr3 - ml1
        vector = [(mr(5) - ml(1)) (mr(6) - ml(2))];
        vector = vector';
        Y = sim(net,vector);
        if Y == [1;0]
            display('3');
            a = [ml(1) ml(2) mr(5) mr(6)];
        else
            %mr1 - ml3
            vector = [(mr(1) - ml(5)) (mr(2) - ml(6))];
            vector = vector';
            Y = sim(net,vector);
            if Y == [1;0]
                display('4');
                a = [ml(5) ml(6) mr(1) mr(2)];
            else
```

```

%mr2 - ml2
vector = [(mr(3) - ml(3)) (mr(4) - ml(4))];
vector = vector';
Y = sim(net,vector);
if Y == [1;0]
    display('5');
    a = [ml(3) ml(4) mr(3) mr(4)];
else
%mr3 - ml2
vector = [(mr(5) - ml(3)) (mr(6) - ml(4))];
vector = vector';
Y = sim(net,vector);
if Y == [1;0]
    display('6');
    a = [ml(3) ml(4) mr(5) mr(6)];
else
%mr2 - ml3
vector = [(mr(3) - ml(5)) (mr(4) - ml(6))];
vector = vector';
Y = sim(net,vector);
if Y == [1;0]
    display('7');
    a = [ml(5) ml(6) mr(3) mr(4)];
else
    display('Unable to fix the mouth points. ');
    a = [ml(1) ml(2) mr(1) mr(2)];
    display(a);
end
end
end
end
end
end
end

% check if the choosen point lies within the matric dimension of the image
% this is important for the program not throw an error when the mouth is no
% located right, so that it keeps running and does not stop

if a(1) > 251
    a(1) = 250;
end

```



```
if a(3) > 251
    a(3) = 250;
end
```

Filename : getMouthLipsCoordinates.m

This locates the upper and lower lips.

```
function mouth_lips =
getMouthLipsCoordinates (Image1,mouth_corner_left,mouth_corner_right,dt,thresh_lips)

    dif = floor(2 * (dt/3));
    mid = mouth_corner_left(2) + floor((mouth_corner_right(2) -
mouth_corner_left(2)) / 2);

    % Separet the estimated mouth region
    Image_Mouth_Mean = Image1(mouth_corner_left(1) - floor(dt/3) :
mouth_corner_left(1) + dif, mouth_corner_left(2) : mouth_corner_right(2));

    Image_Mouth = Image1(mouth_corner_left(1) - floor(dt/3) : mouth_corner_left(1)
+ dif, mid - floor(dt/10) : mid + floor(dt/10));

    % convert mouth region into binary image with the threshold set to the 2/3
    % of the mean intensity of whole region
    level = mean2(Image_Mouth_Mean) / 255;
    level = 0.8 * level * thresh_lips;
    Image_Mouth_BW = im2bw(Image_Mouth,level);

    % invert the colors
    Image_Ones = ones(size(Image_Mouth));
    Image_Mouth_BW = Image_Ones - Image_Mouth_BW;

    % get a bounding box around the thresholded area
    statsM = regionprops(Image_Mouth_BW,'BoundingBox');
    cordMouth = [statsM.BoundingBox];

    if numel(cordMouth) == 0
        display('Mouth lips were not located right. Setting default values...');
        default = ((mouth_corner_right(1) - mouth_corner_left(1)) / 2);
        cordMouth = [0 default 0 0];
    end

    %exrtact the corner points of mouth region
    MouthRmin = mouth_corner_left(1) - floor(dt/3) + round(cordMouth(2));
```

```
MouthRmax = mouth_corner_left(1) - floor(dt/3) +  
round(cordMouth(2)+cordMouth(4));
```

```
% extracted coordinates of the midpoint of the lips  
mouth_lips = [MouthRmin  mouth_corner_left(2) + floor(( mouth_corner_right(2) -  
mouth_corner_left(2)) / 2) MouthRmax  mouth_corner_left(2) +  
floor(( mouth_corner_right(2) - mouth_corner_left(2)) / 2)];
```

Filename : Pso_nose.m

This is the particle swarm optimization algorithm used to locate the nose.

```
function [Gbest M] = Pso_nose(Rmin,Rmax,Cmin,Cmax,I,width, thresh_nose)

% PSO algorithm for extracting the nose midpoint coordinates

n=10;%no of particles
Vmax= 10; % Max velocity..set arbitrarily
Vmin= 3; % particle move one pixel at a time at minimum
Gbest=[0 0];
GbestFit = 0;
MaxFit = 0;
MaxCord = [0 0];
wi = 0.8; %inertial weight
c1=0.5;
c2=0.5;

ind=0;
coord=[];
prev_best_values=[];
threshold=400;
dt = floor(width/4);
center = Rmin + (Rmax - Rmin)/2.0;

%initialising particles with random velocities and setting
%Pbest to the initial position
%Deploying the particles into an region where the nose is expected.

deltaR2 = round((Rmax - Rmin) / 2);
deltaR4 = round((Rmax - Rmin) / 4);
deltaC2 = round((Cmax - Cmin) / 2);
deltaC4 = round((Cmax - Cmin) / 4);
deltaC = Cmax - Cmin;
deltaR = Rmax - Rmin;

for i=1:n
    particle(i).posn=[round((Rmin + deltaR4)+ deltaR2*rand(1)) round((Cmin +
deltaC4) + deltaC2*rand(1))];
```

```

        particle(i).vel=[round(Vmin+(Vmax-Vmin)*rand(1)) round(Vmin+(Vmax-
Vmin)*rand(1))];
        particle(i).Pbest=particle(i).posn;

particle(i).func_response=cost_functionNose(I,particle(i).posn(1),particle(i).posn(2),
dt,center, thresh_nose);
        particle(i).func_resp_prev=particle(i).func_response;
    end

    % initiating values
    bestfind = find([particle.func_response] == max([particle.func_response]));
    best = bestfind(1);
    Gbest=particle(best).posn;
    GbestFit = particle(best).func_response;
    MaxCord = Gbest;
    best_response=particle(best).func_response;
    MaxFit = best_response;

    itr=1;
    t=1;
    thresh_list=[];

    while ind == 0 && itr < 22

        %Compute particles new position and velocities
        % if GbestFit > 0 compute new position and velocities - algorithm
        % is converging.
        if (GbestFit > 0)
            for i=1:n
                %set new velocities

particle(i).vel(1)=wi*particle(i).vel(1)+c1*rand(1)*(particle(i).Pbest(1)-
particle(i).posn(1))+ c2*rand(1)*(Gbest(1)-particle(i).posn(1));

particle(i).vel(2)=wi*particle(i).vel(2)+c1*rand(1)*(particle(i).Pbest(2)-
particle(i).posn(2))+ c2*rand(1)*(Gbest(2)-particle(i).posn(2));

                % set new position
                particle(i).posn(1)=round(particle(i).posn(1)+particle(i).vel(1));
                particle(i).posn(2)=round(particle(i).posn(2)+particle(i).vel(2));

                %checking if the solution lies within domain

```

```

        if particle(i).posn(1) <Rmin || particle(i).posn(1) > Rmax ||
particle(i).posn(2) <Cmin || particle(i).posn(2) > Cmax
            particle(i).posn=particle(i).Pbest;    % setting position to
previous P-best

            particle(i).vel=[rand(1)*Vmax rand(1)*Vmax];    %
setting velocity to max
        end
        %Calculating the function response for each particle with new
        %positions

particle(i).func_response=cost_functionNose(I,particle(i).posn(1),particle(i).posn(2),
dt,center, thresh_nose);
        %checking and updating Pbest
        if particle(i).func_response > particle(i).func_resp_prev
            particle(i).Pbest=particle(i).posn;
        end

        particle(i).func_resp_prev = particle(i).func_response;

    end
    V=[particle.func_response];
    best_ones=find(V==max(V));
    best=best_ones(1);

    %Gbest is a global best of all times
    %display(particle(best).func_response);
    if particle(best).func_response > GbestFit
        Gbest = particle(best).Pbest;
        GbestFit = particle(best).func_response;
    end
    best_response=particle(best).func_response;
    if (best_response < MaxFit)
        MaxCord = Gbest;
    end
else
    % if is not > 0 than deploy te particles again in new random
    % positions and with new random speeds
    if (itr < 5)
        for i=1:n
            particle(i).posn=[round((Rmin + deltaR4)+ deltaR2*rand(1))
round((Cmin + deltaC4) + deltaC2*rand(1))];

```

```

        particle(i).vel=[round(Vmin+(Vmax-Vmin)*rand(1))
round(Vmin+(Vmax-Vmin)*rand(1))];
        particle(i).Pbest=particle(i).posn;

particle(i).func_response=cost_functionNose(I,particle(i).posn(1),particle(i).posn(2),
dt,center, thresh_nose);
        particle(i).func_resp_prev=particle(i).func_response;
    end
else
    % if we didn't find a positive fitness value before the
    % fifth iteration that the region for deployment is
    % extended.
    % it helps in the cases, when the face region is not
    % located properly

        for i=1:n
            particle(i).posn=[round(Rmin + deltaR*rand(1)) round(Cmin +
deltaC*rand(1))];
            particle(i).vel=[round(Vmin+(Vmax-Vmin)*rand(1))
round(Vmin+(Vmax-Vmin)*rand(1))];
            particle(i).Pbest=particle(i).posn;

particle(i).func_response=cost_functionNose(I,particle(i).posn(1),particle(i).posn(2),
dt,center, thresh_nose);
            particle(i).func_resp_prev=particle(i).func_response;
        end
    end
    best=max_position([particle.func_response]);
    Gbest=particle(best).posn;
    GbestFit = particle(best).func_response;
    MaxCord = Gbest;
    best_response=particle(best).func_response;
    MaxFit = best_response;
    t=1;
    thresh_list=[];

end % all particles updated...

% 2 - based on the mean value of func_response of all particles
if mean([particle.func_response]) >= 1100;%threshold;
    ind = 1;
end

```

```
%...recording co-ordinates for display

itr = itr+1;
thresh_list=[thresh_list best_response];
if itr == 20
    ind = 1;
end
end

if itr == 22
    Gbest = MaxCord;
end
```


Filename : cost_functionNose.m

This the cost function used to compute the fitness of the particle in the swarm looking for the nose.

```
function y = cost_functionNose(I ,r ,c ,dtIn ,center, thresh_nose)
global Im;
Im = I;
global dt;
dt = dtIn;
global de;
de = floor(dt/8);
rt1 = r + floor(dt/2);
ct1 = c;
rt2 = rt1 + de;
ct2 = c;
rt3 = rt2 + de;
ct3 = c;
rt4 = rt3 + de;
ct4 = c;
K = 50;

y = Cc(r,c, thresh_nose) + CPos(r,center);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function a = Cm(r,c)
global Im;
global dt;
Kc=0.2;
I1=double(Im( (r-floor(dt/2)) : (r+ floor(dt/2)), (c-floor(dt/2)) : (c+
floor(dt/2))));
var1=sum(std(I1));
var2=sum(std(I1'));
a=Kc*(var1+var2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function a = CMean(r,c)
global Im;
global dt;
I1=double(Im( (r - floor(dt / 6) : r + floor(dt / 6)), c - floor(dt/6) : c +
floor(dt/6)));
```

```

a = mean2(I1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function c=U(x)
% Unit step function.....
if x >=0
    c=1;
else
    c=0;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function a=Cc(r,c, thresh_nose)
global dt;
global Im;
Kc=0;
a=0;

%mean intensity central area
uc = mean2(Im((r- floor(dt/10)):(r+ floor(dt/10)),(c- floor(dt/10)):(c+
floor(dt/10))));

%mean intensity of the region to the left
ul = mean2(Im(r : (r + floor(dt/4)),(c - floor(dt/2)):(c - floor(dt/4))));

%mean intensity of the region to the right
ur = mean2(Im(r : (r + floor(dt/4)),(c + floor(dt/4)):(c + floor(dt/2))));

% intensity of midpoint more than 200
if uc >= thresh_nose
    Kc=1.2*(uc - 200);
end
% left region brighter than 150
if ul >= (thresh_nose*0.75)
    a = a - 200;
end
% right region brighter than 150
if ur >= (thresh_nose*0.75)
    a = a - 200;
end
% difernce between midpoint and left region intensities is less than 50
if uc - ul < (0.25*thresh_nose)

```

```

    a=a-200;
end
% diference between midpoint and right point intensities is less than 50
if uc - ur < (0.25*thresh_nose)
    a=a-200;
end

a = Kc - ur - ul + 2*uc;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function a = CPos(r,center)
% function saying the the higher the region is, the better.

a = 200 - 3 * abs(center - r);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

Filename : getNoseCorners.m

This locates the nostrils based upon the coordinates of the nose.

```
function nose_corners = getNoseCorners(nose,Image,dt,thresh_nostrils)

NoseCornerRmin = nose(1);
NoseCornerRmax = nose(1) + floor(dt/2);
NoseCornerCmin = nose(2) - floor(0.5 * dt);
NoseCornerCmax = nose(2) + floor(0.5 * dt);

% extract nose corners

deltaNose2 = floor((NoseCornerCmax - NoseCornerCmin) / 2);

Image_Nose_Left = Image(NoseCornerRmin : NoseCornerRmax, NoseCornerCmin :
NoseCornerCmin + deltaNose2);

% convert the left nose region into binary image with the threshold set
% to the 1/3 of the mean intensity of whole region
level = mean2(Image_Nose_Left) / 255;
level = (level / 3) * thresh_nostrils;
Image_NoseL_BW = im2bw(Image_Nose_Left,level);

% invert the colors
Image_Ones = ones(size(Image_Nose_Left));
Image_NoseL_BW = Image_Ones - Image_NoseL_BW;

% get a bounding box around the thresholded area
NoseL_Stat = bwlabel(Image_NoseL_BW);
statsNL = regionprops(NoseL_Stat,'BoundingBox');
cordNoseL = [statsNL.BoundingBox];

Image_Nose_Right = Image(NoseCornerRmin : NoseCornerRmax, NoseCornerCmin +
deltaNose2 : NoseCornerCmax);

% convert the l nose region into binary image with the threshold set
% to the 1/4 of the mean intensity of whole region
level = mean2(Image_Nose_Right) / 255;
level = (level / 3) * thresh_nostrils;
Image_NoseR_BW = im2bw(Image_Nose_Right,level);
```

```

% invert the colors
Image_Ones = ones(size(Image_Nose_Right));
Image_NoseR_BW = Image_Ones - Image_NoseR_BW;

% get a bounding box around the thresholded area
NoseR_Stat = bwlabel(Image_NoseR_BW);
statsNR = regionprops(NoseR_Stat,'BoundingBox');
cordNoseR = [statsNR.BoundingBox];

if (numel(statsNR) == 0)
else
    offsetR = numel(statsNR) - 1;
end

% check if the nostrils were located right.
% 1) both of them were not found => use default values
% 2) one them was located ok => us the offset from the midpoint of the good
%    one to estimate the other one.

left_ok = 1;
right_ok = 1;

if numel(cordNoseL) == 0
    cordNoseL = [(nose(2) - floor(dt/3)) - NoseCornerCmin) ((nose(1) +
floor(dt/4)) - NoseCornerRmin) 2 1];
    left_ok = 0;
end
if numel(cordNoseR) == 0
    cordNoseR = [(nose(2) + floor(dt/3)) - deltaNose2 - NoseCornerCmin)
((nose(1) + floor(dt/4)) - NoseCornerRmin) 1 1];
    offsetR = 0;
    right_ok = 0;
end

nose_corners = [floor(NoseCornerRmin + cordNoseL(2) + cordNoseL(4) / 2)
floor(NoseCornerCmin + cordNoseL(1)) floor(NoseCornerRmin + cordNoseR((4 * offsetR) +
2) + cordNoseR((4 * offsetR) + 4) / 2) floor(NoseCornerCmin + deltaNose2 +
cordNoseR((4 * offsetR) + 1) + cordNoseR((4 * offsetR) + 3))];

if (left_ok == 0) && (right_ok == 1)
    nose_corners(1) = nose_corners(3);

```

```
nose_corners(2) = nose(2) - (nose_corners(4) - nose(2));
display('Left nostril was NOT located right. Estimated.');
```

end

```
if (left_ok == 1) && (right_ok == 0)
    nose_corners(3) = nose_corners(1);
    nose_corners(4) = nose(2) + (nose(2) - nose_corners(2));
    display('Right nostril was NOT located right. Estimated.');
```

end

```
if (left_ok == 0) && (right_ok == 0)
    display('Both nostrils were NOT located right. Just guess.');
```

end

Filename : run2.m

This runs the feature location algorithm on the whole video sequence. It uses the features from the previous frame as a reference for the next frame and does not locate the face in every frame. This executed when “Test Video” button on the video toolbox is clicked.

```
function [Image_Points Image_Regions feature_points] = run2(Image, faceCoord,
feature_points, thresh, offset)

load 'MouthTF.mat';

feature_points = cell2mat(feature_points);

Image1 = Image;
Image2 = Image;
Image_Points = Image;

Rmin = faceCoord(1);
Rmax = faceCoord(2);
Cmin = faceCoord(3);
Cmax = faceCoord(4);
deltaRow = Rmax - Rmin;
deltaCol = Cmax - Cmin;

%Aproximate Left eye region
EyeLRmin = Rmin;
EyeLRmax = Rmin + round(deltaRow / 2);
EyeLCmin = Cmin;
EyeLCmax = Cmin + round(deltaCol / 2);

%Aproximate right eye region
EyeRRmin = Rmin;
EyeRRmax = Rmin + round(deltaRow / 2);
EyeRCmin = Cmin + round(deltaCol / 2);
EyeRCmax = Cmax;

%Aproximate left mouth half region
MouthLeftRmin = Rmin + round(deltaRow / 2);
MouthLeftRmax = Rmax;
MouthLeftCmin = Cmin;
```

```

MouthLeftCmax = Cmin + round(deltaCol / 2);

%Aproximate left mouth half region
MouthRightRmin = Rmin + round(deltaRow / 2);
MouthRightRmax = Rmax;
MouthRightCmin = Cmin + round(deltaCol / 2);
MouthRightCmax = Cmax;

width = Cmax - Cmin;
dt = floor((Cmax - Cmin) / 4);

% check if the template can get out of the image while looking for the eye
if EyeRRmin < (dt / 2)
    EyeRRmin = round(dt / 2) + 1;
end

if EyeRCmax > (255 - (dt / 2))
    EyeRCmax = 255 - round(dt / 2) + 1;
end;

border = Cmax - round(deltaCol / 8);
display('finding right eye cordinates.....');
right_eye = Pso2_eye_run(EyeRRmin, EyeRRmax, EyeRCmin, EyeRCmax, Image, width, 1,
border, feature_points(1,:), thresh);

Image_Points(right_eye(1), (right_eye(2) - 5): (right_eye(2) + 5)) = 256;
Image_Points((right_eye(1) - 5): (right_eye(1) + 5), right_eye(2)) = 256;

% check if the right eyebrow is getting closer to the right eye, if yes then
lower
% the region where we are looking for the inner corner of the right eye.
% => this should prevent the algorithm from locating the inner corner of
eyebrows
% instead of the inner corner of eyes.

threshold_Eb2E = (3*dt/4);
right_eb_low = 0;

if (feature_points(1,1) - feature_points(4,1)) < threshold_Eb2E
    right_eb_low = 1;
    display('Right eyebrow lowered.');
```



```

% call function to extract the coordinates of corners of the right eye
display('finding right eye corners cordينات.....');
right_eye_corners = getRightEyeCorners2(right_eye,Image2,dt,
right_eb_low,thresh.eyeIn);

right_eye_in_corner = [right_eye_corners(1) right_eye_corners(2)];
right_eye_out_corner = [right_eye_corners(3) right_eye_corners(4)];

Image_Points(right_eye_in_corner(1), (right_eye_in_corner(2) - 5):
(right_eye_in_corner(2) + 5))=256;
Image_Points((right_eye_in_corner(1) - 5): (right_eye_in_corner(1) + 5),
right_eye_in_corner(2))=256;

Image_Points(right_eye_out_corner(1), (right_eye_out_corner(2) - 5):
(right_eye_out_corner(2) + 5))=256;
Image_Points((right_eye_out_corner(1) - 5): (right_eye_out_corner(1) + 5),
right_eye_out_corner(2))=256;

%extracting right eyebrow region coordinates
EyebrowRRmin = right_eye(1) - floor(1.5 * dt);
EyebrowRRmax = right_eye(1) - floor(0.45 * dt);
EyebrowRCmin = right_eye(2) - floor(dt/4);
EyebrowRCmax = right_eye(2) + floor(dt/2);

% check if the eyebrow template can get out of the image when searching
% for the eyebrow
if EyebrowRRmin < round(dt/2)
    EyebrowRRmin = round(dt/2) + 1;
end

display('finding right eyebrow cordينات.....');
right_eyebrow = Pso_eyebrow_run2(EyebrowRRmin, EyebrowRRmax, EyebrowRCmin,
EyebrowRCmax, Image, width, feature_points(4,:), right_eye, 1, offset);

Image_Points(right_eyebrow(1), (right_eyebrow(2) - 5): (right_eyebrow(2) +
5))=256;
Image_Points((right_eyebrow(1) - 5): (right_eyebrow(1) + 5),
right_eyebrow(2))=256;

% call a function to extract the coordinates of the inner corner of the

```

```

% right eyebrow

display('finding right eyebrow inner corner cordinates.....');
right_eyebrow_corner =
getRightEyebrowCorner(right_eyebrow, Image2, dt, thresh.ebCorner);

Image_Points(right_eyebrow_corner(1), (right_eyebrow_corner(2) - 5):
(right_eyebrow_corner(2) + 5))=256;
Image_Points((right_eyebrow_corner(1) - 5): (right_eyebrow_corner(1) + 5),
right_eyebrow_corner(2))=256;

% check if the template can get out of the image while looking for the eye

if EyeLRmin < (dt / 2)
    EyeLRmin = round(dt / 2) + 1;
end

if EyeLCmin > (dt / 2)
    EyeLCmin = round(dt / 2) + 1;
end;

border = Cmin + round(deltaCol / 8);
display('finding left eye cordinates.....');
left_eye =
Pso2_eye_run(EyeLRmin, EyeLRmax, EyeLCmin, EyeLCmax, Image, width, 0, border,
feature_points(6,:), thresh);

Image_Points(left_eye(1), (left_eye(2) - 5): (left_eye(2) + 5))=256;
Image_Points((left_eye(1) - 5): (left_eye(1) + 5), left_eye(2))=256;

% check if the right eyebrow is getting closer to the right eye, if yes then
lower
% the region where we are looking for the inner corner of the right eye.
% => this should prevent the algorithm from locating the inner corner of
eyebrows
% instead of the inner corner of eyes.

threshold_Eb2E = (3*dt/4);
left_eb_low = 0;

if (feature_points(6,1) - feature_points(9,1)) < threshold_Eb2E
    left_eb_low = 1;

```

```

        display('Left eyebrow lowered. ');
    end

    % call function to extract the coordinates of corners of the left eye

    display('finding left eye corners cordinates.....');
    left_eye_corners = getLeftEyeCorners2(left_eye, Image2, dt, left_eb_low,
thresh.eyeIn);

    left_eye_in_corner = [left_eye_corners(1) left_eye_corners(2)];
    left_eye_out_corner = [left_eye_corners(3) left_eye_corners(4)];

    Image_Points(left_eye_in_corner(1), (left_eye_in_corner(2) - 5):
(left_eye_in_corner(2) + 5))=256;
    Image_Points((left_eye_in_corner(1) - 5): (left_eye_in_corner(1) + 5),
left_eye_in_corner(2))=256;

    Image_Points(left_eye_out_corner(1), (left_eye_out_corner(2) - 5):
(left_eye_out_corner(2) + 5))=256;
    Image_Points((left_eye_out_corner(1) - 5): (left_eye_out_corner(1) + 5),
left_eye_out_corner(2))=256;

    %extracting left eyebrow region coordinates
    EyebrowLRmin = left_eye(1) - floor(1.5 * dt);
    EyebrowLRmax = left_eye(1) - floor(0.45 * dt);
    EyebrowLCmin = left_eye(2) - floor(dt/2);
    EyebrowLCmax = left_eye(2) + floor(dt/4);

    % check if the eyebrow template can get out of the image when searching
    % for the eyebrow
    if EyebrowLRmin < round(dt/2)
        EyebrowLRimn = round(dt/2) + 1;
    end

    display('finding left eyebrow cordinates.....');
    left_eyebrow = Pso_eyebrow_run2(EyebrowLRmin, EyebrowLRmax, EyebrowLCmin,
EyebrowLCmax, Image, width, feature_points(9,:), left_eye, 0, offset);

    display('finding left eyebrow inner corner cordinates.....');
    left_eyebrow_corner =
getLeftEyebrowCorner(left_eyebrow, Image2, dt, thresh.ebCorner);

```

```

        Image_Points(left_eyebrow_corner(1), (left_eyebrow_corner(2) - 5):
(left_eyebrow_corner(2) + 5))=256;
        Image_Points((left_eyebrow_corner(1) - 5): (left_eyebrow_corner(1) + 5),
left_eyebrow_corner(2))=256;

        Image_Points(left_eyebrow(1), (left_eyebrow(2) - 5): (left_eyebrow(2) +
5))=256;
        Image_Points((left_eyebrow(1) - 5): (left_eyebrow(1) + 5),
left_eyebrow(2))=256;

        center = left_eye(2);

        % check if the template for mouth corner can get out of the image.
        if MouthLeftRmax > (255 - dt/2)
            MouthLeftRmax = round(255 - dt/2) + 1;
        end

        display('finding left mouth cordinates.....');
        mouth_left = Pso2_mouth_left_run(MouthLeftRmin, MouthLeftRmax, MouthLeftCmin,
MouthLeftCmax, Image, center,feature_points(11,:), thresh);

        mouth_corner_left = mouth_left(1:2);

        center = right_eye(2);

        % check if the template for mouth corner can get out of the image.
        if MouthRightRmax > (255 - dt/2)
            MouthRightRmax = round(255 - dt/2) + 1;
        end

        display('finding right mouth cordinates.....');
        mouth_right = Pso2_mouth_right_run(MouthRightRmin, MouthRightRmax,
MouthRightCmin, MouthRightCmax, Image, center,feature_points(12,:));

        mouth_corner_right = mouth_right(1:2);
        vector = [(mouth_corner_right(1) - mouth_corner_left(1)) (mouth_corner_right(2)
- mouth_corner_left(2))];

        % using neural network to analyze if the mouth corners were located right
        % if not, then we look at other combinations of the rest of the located
        % mouth corner points

```

```

Y = sim(net,vector);

if Y == [1;0]
    display('Mouth corners located successfully.');
```

Image_Points(mouth_corner_left(1), (mouth_corner_left(2) - 5):
(mouth_corner_left(2) + 5)) = 256;

Image_Points((mouth_corner_left(1) - 5): (mouth_corner_left(1) + 5),
mouth_corner_left(2)) = 256;

Image_Points(mouth_corner_right(1), (mouth_corner_right(2) - 5):
(mouth_corner_right(2) + 5)) = 256;

Image_Points((mouth_corner_right(1) - 5): (mouth_corner_right(1) + 5),
mouth_corner_right(2)) = 256;

```

else
    display('Mouth corners were NOT located successfully.');
```

display('Fixing it...');

mvec = mouth_fix(mouth_left,mouth_right,net);

mouth_corner_left = mvec(1:2);

mouth_corner_right = mvec(3:4);

Image_Points(mouth_corner_left(1), (mouth_corner_left(2) - 5):
(mouth_corner_left(2) + 5)) = 256;

Image_Points((mouth_corner_left(1) - 5): (mouth_corner_left(1) + 5),
mouth_corner_left(2)) = 256;

Image_Points(mouth_corner_right(1), (mouth_corner_right(2) - 5):
(mouth_corner_right(2) + 5)) = 256;

Image_Points((mouth_corner_right(1) - 5): (mouth_corner_right(1) + 5),
mouth_corner_right(2)) = 256;

```

end

display('finding mouth lips coordinates.....');
mouth_Lips =
getMouthLipsCoordinates(Imagel,mouth_corner_left,mouth_corner_right,dt,thresh.lips);

% extracted coordinates of the midpoint of the upper lip
mouth_lip_upper = [mouth_Lips(1) mouth_Lips(2)];
mouth_lip_lower = [mouth_Lips(3) mouth_Lips(4)];

```

```

    Image_Points(mouth_lip_upper(1), (mouth_lip_upper(2) - 5): (mouth_lip_upper(2)
+ 5))=256;
    Image_Points((mouth_lip_upper(1) - 5): (mouth_lip_upper(1) + 5),
mouth_lip_upper(2) )=256;

    Image_Points(mouth_lip_lower(1), (mouth_lip_lower(2) - 5): (mouth_lip_lower(2)
+ 5))=256;
    Image_Points((mouth_lip_lower(1) - 5): (mouth_lip_lower(1) + 5),
mouth_lip_lower(2) )=256;

%extracting the nose region coordinates

% check whether the upper lip was located right - must be above the corners
% of mouth

midpoint = floor((mouth_corner_left(1) + mouth_corner_right(1)) / 2);

if (mouth_lip_upper(1) < midpoint)
    upper_lip = mouth_lip_upper(1);
else
    display('Mouth upper lip was not located successfully.');
```

```

    upper_lip = midpoint - floor(dt / 2);
end

% extract vector of the eye movement and use it for navigating the nose PSO

eye_move_now = mean([left_eye(1),right_eye(1)]);
eye_move_prev = mean([feature_points(1,1),feature_points(6,1)]);

eye_move_vector = round(eye_move_prev - eye_move_now);

NoseRmin = upper_lip - floor(1.5 * dt);
NoseRmax = upper_lip - floor(dt / 4);
NoseCmin = mouth_lip_upper(2) - floor(0.75 * dt);
NoseCmax = mouth_lip_upper(2) + floor(0.75 * dt);

display('finding nose cordinates.....');
nose =
Pso_nose_run_vector(NoseRmin,NoseRmax,NoseCmin,NoseCmax,Image2,width,feature_points(15
,:),eye_move_vector, thresh.nose);

    Image_Points(nose(1), (nose(2) - 5): (nose(2) + 5))=256;
```

```

Image_Points((nose(1) - 5): (nose(1) + 5),nose(2) )=256;

display('finding nostril cordinates.....');
nose_corners = getNoseCorners(nose,Image2,dt,thresh.nosetril);

Image_Points(nose_corners(1), (nose_corners(2) - 5): (nose_corners(2) +
5))=256;
Image_Points((nose_corners(1) - 5): (nose_corners(1) +
5),nose_corners(2) )=256;

Image_Points(nose_corners(3), (nose_corners(4) - 5): (nose_corners(4) +
5))=256;
Image_Points((nose_corners(3) - 5): (nose_corners(3) +
5),nose_corners(4) )=256;

Image_Regions = Image_Points;

% visualization of the feature regions
Image_Regions(Rmin, Cmin:Cmax) = 256;
Image_Regions(Rmax, Cmin:Cmax) = 256;
Image_Regions(Rmin:Rmax, Cmin) = 256;
Image_Regions(Rmin:Rmax, Cmax) = 256;
Image_Regions(Rmin + round(deltaRow / 2), Cmin:Cmax) = 256;
Image_Regions(Rmin:Rmax, Cmin + round(deltaCol / 2)) = 256;

%left eye subregion
Image_Regions(Rmin + round(deltaRow / 3) : Rmin + round(deltaRow / 2), Cmin +
round(deltaCol / 8) ) = 256;
Image_Regions(Rmin + round(deltaRow / 3) : Rmin + round(deltaRow / 2), Cmin +
3*round(deltaCol / 8) ) = 256;
Image_Regions(Rmin + round(deltaRow / 3), Cmin + round(deltaCol / 8) : Cmin +
3*round(deltaCol / 8)) = 256;

%right eye subregion
Image_Regions(Rmin + round(deltaRow / 3) : Rmin + round(deltaRow / 2), Cmin +
5*round(deltaCol / 8) ) = 256;
Image_Regions(Rmin + round(deltaRow / 3) : Rmin + round(deltaRow / 2), Cmin +
7*round(deltaCol / 8) ) = 256;
Image_Regions(Rmin + round(deltaRow / 3), Cmin + 5*round(deltaCol / 8) : Cmin +
7*round(deltaCol / 8)) = 256;

%left mouth subregion

```

```

        Image_Regions(Rmin + 5*round(deltaRow / 8) : Rmin + 7*round(deltaRow / 8), Cmin
+ round(deltaCol / 6)) = 256;
        Image_Regions(Rmin + 5*round(deltaRow / 8), Cmin + round(deltaCol / 6): Cmin +
round(deltaCol / 2)) = 256;
        Image_Regions(Rmin + 7*round(deltaRow / 8), Cmin + round(deltaCol / 6): Cmin +
round(deltaCol / 2)) = 256;

        %right mouth subregion
        Image_Regions(Rmin + 5*round(deltaRow / 8) : Rmin + 7*round(deltaRow / 8), Cmin
+ 5*round(deltaCol / 6)) = 256;
        Image_Regions(Rmin + 5*round(deltaRow / 8), Cmin + round(deltaCol / 2): Cmin +
5*round(deltaCol / 6)) = 256;
        Image_Regions(Rmin + 7*round(deltaRow / 8), Cmin + round(deltaCol / 2): Cmin +
5*round(deltaCol / 6)) = 256;

        % vector with coordinates of all feature points
        feature_points = [right_eye; right_eye_in_corner; right_eye_out_corner;
right_eyebrow; right_eyebrow_corner; left_eye; left_eye_in_corner;
left_eye_out_corner; left_eyebrow; left_eyebrow_corner; mouth_corner_left;
mouth_corner_right; mouth_lip_upper; mouth_lip_lower; nose; nose_corners(1:2);
nose_corners(3:4)];

        end

```


Filename : Pso2_eye_run.m

This is the particle swarm optimization algorithm used to locate the eyes while testing the whole video sequence.

```
function [VeryBest]=Pso2_eye_run(Rmin,Rmax,Cmin,Cmax,I,width, side,border,
prev_eye, thresh)

n = 10; % number of particles
group = 1; % number of groups
% particle = zeros(8, n*group);
Vmax= 5; % Max velocity..set arbitrarily
Vmin= 3; % particle move one pixel at a time at minimum
Gbest=[0 0 0 0 0 0];
GbestFit = [0 0 0];
best = [0 0 0];
VeryBest = [0 0];

wi = 0.85; %inertial weight
c1=0.5;
c2=0.5;

ind=0;
coord=[];
prev_best_values=[];
threshold = 400;
dt = floor(width/4);

%initialising particles with random velocities and setting
%Pbest to the initial posiion
%Deploying the particles into an region where the eye is expected.

deltaR3 = round((Rmax - Rmin) / 3);
deltaR2 = round((Rmax - Rmin) / 2);
deltaC4 = round((Cmax - Cmin) / 4);
deltaC = Cmax - Cmin;

deltaRunR = 5; % round(dt/5);
deltaRunC = 5; % round(dt/8);

% initializing particles in all groups
```

```

for g = 0:(group - 1)
    for i = 1:n
        partInd = g*n + i;
        particle(partInd).posn=[round((prev_eye(1) - deltaRunR) + 2*deltaRunR
* rand(1)) round((prev_eye(2) - deltaRunC) + 2*deltaRunC * rand(1))];
        particle(partInd).vel = [round(Vmin+(Vmax-Vmin)*rand(1))
round(Vmin+(Vmax-Vmin)*rand(1))];

        particle(partInd).Pbest = particle(partInd).posn;
        particle(partInd).func_response =
cost_function7subReg(I,particle(partInd).posn(1),
particle(partInd).posn(2),dt,side,border, thresh.eye, thresh.lighting);
        particle(partInd).func_resp_prev = particle(partInd).func_response;
    end

    bestfind = find([particle((g*n + 1):(g*n + n)).func_response] ==
max([particle((g*n + 1):(g*n + n)).func_response]));
    best(g+1) = g*n + bestfind(1);
    Gbest((2*g + 1):(2*g + 2)) = particle(best(g+1)).posn;
    GbestFit(g+1) = particle(best(g+1)).func_response;
end

itr=1;
t=1;
thresh_list=[];

while ind == 0 && itr < 22
    %Compute particles new position and velocities

    % if GbestFit > 0 compute new position and velocities - algorithm
    % is converging.
    for g = 0:(group-1)
        if (GbestFit(g+1) > 0)
            %In each group

            for i=1:n
                partInd = g*n + i;

particle(partInd).vel(1)=wi*particle(partInd).vel(1)+c1*rand(1)*(particle(partInd).Pbe

```

```

st(1) = particle(partInd).posn(1) + c2*rand(1)*(Gbest(2*g + 1) -
particle(partInd).posn(1));

particle(partInd).vel(2) = wi*particle(partInd).vel(2) + c1*rand(1)*(particle(partInd).Pbest(2) -
particle(partInd).posn(2)) + c2*rand(1)*(Gbest(2*g + 2) -
particle(partInd).posn(2));

particle(partInd).posn(1) = round(particle(partInd).posn(1) + particle(partInd).vel(1));

particle(partInd).posn(2) = round(particle(partInd).posn(2) + particle(partInd).vel(2));

        %checking if the solution lies within domain
        if particle(partInd).posn(1) < Rmin || particle(partInd).posn(1)
> Rmax || particle(partInd).posn(2) < Cmin || particle(partInd).posn(2) > Cmax
            particle(partInd).posn = particle(partInd).Pbest;    %
setting position to previous P-best
            particle(partInd).vel = [rand(1)*Vmax rand(1)*Vmax];
% setting velocity to max
            end

        %Calculating the function response for each particle with new
        %positions

particle(partInd).func_response = cost_function7subReg(I, particle(partInd).posn(1),
particle(partInd).posn(2), dt, side, border, thresh.eye, thresh.lighting);

        %checking and updating Pbest
        if particle(partInd).func_response >
particle(partInd).func_resp_prev
            particle(partInd).Pbest = particle(partInd).posn;
            end
            particle(partInd).func_resp_prev =
particle(partInd).func_response;
            end

V = [particle((g*n + 1):(g*n + n)).func_response];
best_ones = find(V == max(V));
best(g+1) = g*n + best_ones(1);

% Gbest is a global best of all times
% display(particle(best(g+1)).func_response);

```

```

        if particle(best(g+1)).func_response > GbestFit(g+1)
            Gbest((2*g + 1):(2*g + 2)) = particle(best(g+1)).Pbest;
            GbestFit(g+1) = particle(best(g+1)).func_response;
        end
    end
else
    % if is not > 0 than deploy te particles again in new random
    % positions and with new random speeds
    if (itr < 22)

        for i=1:n
            partInd = g*n + i;
            particle(partInd).posn=[round((prev_eye(1) - deltaRunR) +
2*deltaRunR * rand(1)) round((prev_eye(2) - deltaRunC) + 2*deltaRunC * rand(1))];
            particle(partInd).vel=[round(Vmin+(Vmax-Vmin)*rand(1))
round(Vmin+(Vmax-Vmin)*rand(1))];
            particle(partInd).Pbest=particle(partInd).posn;

particle(partInd).func_response=cost_function7subReg(I,particle(partInd).posn(1),
particle(partInd).posn(2),dt,side,border, thresh.eye, thresh.lighting);

particle(partInd).func_resp_prev=particle(partInd).func_response;
            end
        end

        bestfind = find([particle((g*n + 1):(g*n + n)).func_response] ==
max([particle((g*n + 1):(g*n + n)).func_response]));
        best(g+1) = g*n + bestfind(1);
        Gbest((2*g + 1):(2*g + 2)) = particle(best(g+1)).posn;
        GbestFit(g+1) = particle(best(g+1)).func_response;
        thresh_list=[];
    end % all particles updated...
end
%...recording co-ordinates for display

itr = itr+1;
end
% choose the best point
very_best_ones = find( GbestFit == max(GbestFit(1:group)));
very_best = very_best_ones(1);
VeryBest = Gbest(2*(very_best - 1) + 1: 2*(very_best - 1) + 2);

```

Filename : getRightEyeCorners2.m

This extracts the corner coordinates of the eyes while testing the complete video sequence.

```
function right_eye_corners =
getRightEyeCorners2(right_eye,Image,dt,right_eb_low,thresh_eyeInner)

if right_eb_low == 0
    EyeRegionRRmin = right_eye(1) - floor(dt/5);
else
    EyeRegionRRmin = right_eye(1);
end
EyeRegionRRmax = right_eye(1) + floor(dt/4);
EyeRegionRCmin = right_eye(2) - floor(dt / 2);
EyeRegionRCmax = right_eye(2) + floor(2 * (dt/3));

%estimated eye region
Image_Eye = Image(EyeRegionRRmin : EyeRegionRRmax, EyeRegionRCmin :
EyeRegionRCmax);

% convert the right eye region into binary image with the threshold set
% to the 1/2 of the mean intensity of whole region
level = mean2(Image_Eye) / 255;
level = (0.8 * level) * thresh_eyeInner;
Image_Eye_BW = im2bw(Image_Eye,level);

% invert the colors
Image_Ones = ones(size(Image_Eye));
Image_Eye_BW = Image_Ones - Image_Eye_BW;

% get a bounding box around the thresholded area
statsE = regionprops(Image_Eye_BW, 'BoundingBox');
cordEye = [statsE.BoundingBox];

% get a region around the inner corner of the right eyebrow
Image_Eye_Corner_In_BW = Image_Eye_BW(1 : (EyeRegionRRmax - EyeRegionRRmin),
ceil(cordEye(1)) : ceil(cordEye(1) + 1));

Eye_Corner_In_Stat = bwlabel(Image_Eye_Corner_In_BW);
statsEIC = regionprops(Eye_Corner_In_Stat, 'BoundingBox');
cordEyeInCorner = [statsEIC.BoundingBox];
```

```

% get a region around the outer corner of the right eye
Image_Eye_Corner_Out_BW = Image_Eye_BW(1 : (EyeRegionRRmax - EyeRegionRRmin),
floor(cordEye(1) + cordEye(3) - 1) : floor(cordEye(1) + cordEye(3)));

Eye_Corner_Out_Stat = bwlabel(Image_Eye_Corner_Out_BW);
statsEOC = regionprops(Eye_Corner_Out_Stat, 'BoundingBox');
cordEyeOutCorner = [statsEOC.BoundingBox];

%Check if the corner point was found. If not a default value is assigned
%and an error is reported

if numel(cordEyeOutCorner) == 0
    cordEyeOutCorner = [0 (right_eye(2) - EyeRegionRRmin) + round(dt / 4) 0 0];
    display('Outer corner of the right eye was not located right...');
end

if numel(cordEyeInCorner) == 0
    cordEyeInCorner = [0 (right_eye(2) - EyeRegionRRmin) - round(dt / 4) 0 0];
    display('Inner corner of the right eye was not located right...');
end

right_eye_corners = [floor(EyeRegionRRmin + cordEyeInCorner(2) +
cordEyeInCorner(4) / 2) floor(EyeRegionRCmin + cordEye(1)) floor(EyeRegionRRmin +
cordEyeOutCorner(2) + cordEyeOutCorner(4) / 2) floor(EyeRegionRCmin + cordEye(1) +
cordEye(3))];

```

Filename : Pso_eyebrow_run2.m

This is the particle swarm optimization algorithm used to locate the eyebrow while testing the whole video sequence.

```
function [Gbest M]=Pso_eyebrow_run2(Rmin, Rmax, Cmin, Cmax, I, width,
prev_eyebrow, eye,side, offset)
% PSO algorithm for extracting the eyebrow coordinates
% variation of PSO_eyebrow_run, it looks for the eye only in a tiny region
% which is always at the same ofset from the eye as was at the test image

n=10;%no of particles
Vmax= 5; % Max velocity..set arbitrarily
Vmin= 3; % particle move one pixel at a time at minimum
Gbest=[0 0];
GbestFit = 0;
MaxFit = 0;
MaxCord = [0 0];
wi = 0.8; %inertial weight
c1=0.5;
c2=0.5;

ind=0;
coord=[];
prev_best_values=[];
threshold=400;
dt = floor(width/4);

%initialising particles with random velocities and setting
%Pbest to the initial posiion
%Deploying the particles into an region where the eye is expected.

deltaR2 = round((Rmax - Rmin) / 2);
deltaC = Cmax - Cmin;
deltaR = Rmax - Rmin;

deltaRun = 10;

% get coordinates of the expected eyebrow region
if side == 0
    midP = eye(2) - offset.EBL;
```

```

else
    midP = eye(2) - offset.EBR;
end

% get coordinates of the lower border of the eyebrow region. This should
% prevent the PSO looking for eyebrow get confused by the eye, when
% eyebrows are lowered close to the eyes

lowerEdge = eye(1) - round(dt/3);

if abs(prev_eyebrow(2) - midP) > (dt/2)
    prev_eyebrow(1) = eye(1) - round(dt/2);
    display('prev_eyebrow was not used');
end

% check if the previous eyebrow does not lie too close to the current eye.
% if this true, then it is likely that there was a big movement of the
% eyebrow or of the whole head. Therefore we will estimate the position of
% the eyebrow and won't use the value of the previous eyebrow ( it probably
% lies within the current eye region)

if (prev_eyebrow(1) > lowerEdge)
    eyeBR = lowerEdge - round(dt/3);
    display('Too big movement of eye was detected. Eyebrow moved upwards.');
```

```

else
    eyeBR = prev_eyebrow(1);
end

for i=1:n
    posR = (eyeBR - deltaRun) + 2*deltaRun * rand(1);
    if posR > lowerEdge
        posR = lowerEdge - round(dt/5);
    end
    particle(i).posn=[round(posR) round((midP - 1) + rand(1))];
    particle(i).vel=[round(Vmin+(Vmax-Vmin)*rand(1)) round(Vmin+(Vmax-
Vmin)*rand(1))];
    particle(i).Pbest=particle(i).posn;

particle(i).func_response=cost_functionEyebrow(I,particle(i).posn(1),particle(i).posn(
2),dt);
    particle(i).func_resp_prev=particle(i).func_response;
end

```



```

% initiating values
bestfind = find([particle.func_response] == max([particle.func_response]));
best = bestfind(1);
Gbest=particle(best).posn;
GbestFit = particle(best).func_response;
MaxCord = Gbest;
best_response=particle(best).func_response;
MaxFit = best_response;

itr=1;
t=1;
converge = 1;
thresh_list=[];

while ind == 0 && itr < 22

    %Compute particles new position and velocities
    % if GbestFit > 0 compute new position and velocities - algorithm
    % is converging.
    if (GbestFit > 0)
        for i=1:n

particle(i).vel(1)=wi*particle(i).vel(1)+c1*rand(1)*(particle(i).Pbest(1)-
particle(i).posn(1)) + c2*rand(1)*(Gbest(1)-particle(i).posn(1));
                particle(i).vel(2)=wi*particle(i).vel(2) +
c1*rand(1)*(particle(i).Pbest(2) - particle(i).posn(2)) + c2*rand(1)*(Gbest(2)-
particle(i).posn(2));

                % set new position
                % if possible move only in the column around the expected
                % eyebrow point
                particle(i).posn(1) =
round(particle(i).posn(1)+particle(i).vel(1));

                if converge == 1
                    particle(i).posn(2) = midP;
                else

particle(i).posn(2)=round(particle(i).posn(2)+particle(i).vel(2));
                    end
        end
    end
end

```

```

        %checking if the solution lies within domain
        if particle(i).posn(1) < Rmin || particle(i).posn(1) > Rmax ||
particle(i).posn(2) < Cmin || particle(i).posn(2) > Cmax
            particle(i).posn=particle(i).Pbest;    % setting position to
previous P-best
            particle(i).vel=[rand(1)*Vmax rand(1)*Vmax];    %
setting velocity to max
        end
        %Calculating the function response for each particle with new
%positions

particle(i).func_response=cost_functionEyebrow(I,particle(i).posn(1),particle(i).posn(
2),dt);

        %checking and updating Pbest
        if particle(i).func_response > particle(i).func_resp_prev
            particle(i).Pbest=particle(i).posn;
        end

        particle(i).func_resp_prev = particle(i).func_response;

    end

    V=[particle.func_response];
    best_ones=find(V==max(V));
    best=best_ones(1);

    %Gbest is a global best of all times
    %display(particle(best).func_response);
    if particle(best).func_response > GbestFit
        Gbest = particle(best).Pbest;
        GbestFit = particle(best).func_response;
    end
    best_response=particle(best).func_response;
    if (best_response < MaxFit)
        MaxCord = Gbest;
    end

else
    % if is not > 0 than deploy te particles again in new random
% positions and with new random speeds
    if (itr < 22)
        for i=1:n
            posR = (eyeBR - deltaRun) + 2*deltaRun * rand(1);

```

```

        if posR > lowerEdge
            posR = lowerEdge;
        end
        particle(i).posn=[round(posR) round((midP - 1) + rand(1))];
        particle(i).vel=[round(Vmin+(Vmax-Vmin)*rand(1))
round(Vmin+(Vmax-Vmin)*rand(1))];
        particle(i).Pbest=particle(i).posn;

particle(i).func_response=cost_functionEyebrow(I,particle(i).posn(1),particle(i).posn(
2),dt);

        particle(i).func_resp_prev=particle(i).func_response;
    end
end
bestfind = find([particle.func_response] ==
max([particle.func_response]));
best = bestfind(1);
Gbest=particle(best).posn;
GbestFit = particle(best).func_response;
MaxCord = Gbest;
best_response=particle(best).func_response;
MaxFit = best_response;
t=1;
thresh_list=[];
end % all particles updated...

% 2 - based on the mean value of func_response of all particles
if mean([particle.func_response]) >= 1100;%threshold;
    ind = 1;
end

%...recording co-ordinates for display
itr = itr+1;
thresh_list=[thresh_list best_response];
if itr == 20
    ind = 1;
end
end

if converge == 0
    display ('Did not converged. Region enlarged');
else
    display('Eyebrow was located at the defined offset');
end

```

```
end

if itr == 22
    Gbest = MaxCord;
end
```

Filename : getLeftEyeCorners2.m

This locates the corners of left eye while testing the complete video sequence.

```
function left_eye_corners =  
getLeftEyeCorners2(left_eye,Image,dt,left_eb_low,thresh_eyeInner)  
  
if left_eb_low == 0  
    EyeRegionLRmin = left_eye(1) - floor(dt/5);  
else  
    EyeRegionLRmin = left_eye(1);  
end  
EyeRegionLRmax = left_eye(1) + floor(dt/4);  
EyeRegionLCmin = left_eye(2) - floor(2 * (dt/3));  
EyeRegionLCmax = left_eye(2) + floor(dt / 2);  
  
Image_Eye = Image(EyeRegionLRmin : EyeRegionLRmax, EyeRegionLCmin :  
EyeRegionLCmax);  
  
% convert the left eye region into binary image with the threshold set  
% to the mean intensity of whole region  
level = mean2(Image_Eye) / 255;  
level = (0.8 * level) * thresh_eyeInner;  
Image_Eye_BW = im2bw(Image_Eye,level);  
  
% invert the colors  
Image_Ones = ones(size(Image_Eye));  
Image_Eye_BW = Image_Ones - Image_Eye_BW;  
  
% get a bounding box around the thresholded area  
statsE = regionprops(Image_Eye_BW,'BoundingBox');  
cordEye = [statsE.BoundingBox];  
  
% get a region around the inner corner of the left eye  
Image_Eye_Corner_In_BW = Image_Eye_BW(1 : (EyeRegionLRmax - EyeRegionLRmin),  
floor(cordEye(1) + cordEye(3) - 1) : floor(cordEye(1) + cordEye(3)));  
  
Eye_Corner_In_Stat = bwlabel(Image_Eye_Corner_In_BW);  
statsEIC = regionprops(Eye_Corner_In_Stat,'BoundingBox');  
cordEyeInCorner = [statsEIC.BoundingBox];
```

```

% get a region around the outer corner of the left eye
Image_Eye_Corner_Out_BW = Image_Eye_BW(1 : (EyeRegionLRmax - EyeRegionLRmin),
ceil(cordEye(1)) : ceil(cordEye(1) + 1));

Eye_Corner_Out_Stat = bwlabel(Image_Eye_Corner_Out_BW);
statsEOC = regionprops(Eye_Corner_Out_Stat, 'BoundingBox');
cordEyeOutCorner = [statsEOC.BoundingBox];

%Check if the corner point was found. If not a default value is assigned
%and an error is reported

if numel(cordEyeOutCorner) == 0
    cordEyeOutCorner = [0 left_eye(2) - round(dt / 4) 0 0];
    display('Outer corner of the left eye was not located right...');
end

if numel(cordEyeInCorner) == 0
    cordEyeInCorner = [0 left_eye(2) + round(dt / 4) 0 0];
    display('Inner corner of the left eye was not located right...');
end

left_eye_corners = [floor(EyeRegionLRmin + cordEyeInCorner(2) +
cordEyeInCorner(4) / 2) floor(EyeRegionLCmin + cordEye(1) + cordEye(3))
floor(EyeRegionLRmin + cordEyeOutCorner(2) + cordEyeOutCorner(4) / 2)
floor(EyeRegionLCmin + cordEye(1))];

```

Filename : Pso2_mouth_left_run.m

This is the particle swarm optimization algorithm used to locate the left corner of the mouth while testing the whole video sequence.

```
function [VeryBest]=Pso2_mouth_left_run(Rmin, Rmax, Cmin, Cmax, I, center,
prev_left_mouth, thresh)

n = 10; % number of particles
group = 3; % number of groups
% particle = zeros(8, n*group);
Vmax= 8; % Max velocity..set arbitrarily
Vmin= 3; % particle move one pixel at a time at minimum
Gbest=[0 0 0 0 0 0];
GbestFit = [0 0 0];
MaxFit = [0 0 0];
MaxCord = [0 0 0 0 0 0];
best_response = [0 0 0];
best = [0 0 0];
VeryBest = [0 0];

wi = 0.85; %inertial weight
c1=0.5;
c2=0.5;

ind=0;
coord=[];
prev_best_values=[];
threshold = 400;

Rmin = prev_left_mouth(1) - 20;
Rmax = prev_left_mouth(1) + 20;
Cmin = prev_left_mouth(2) - 20;
Cmax = prev_left_mouth(2) + 20;

dt = floor((Cmax - Cmin)/2);
meanInt = mean2(I(Rmin:Rmax,Cmin:Cmax));

%initialising particles with random velocities and setting
%Pbest to the initial posiion
%Deploying the particles into an region where the eye is expected.
```

```

deltaR4 = round((Rmax - Rmin) / 4);
deltaR2 = round((Rmax - Rmin) / 2);
deltaC2 = round((Cmax - Cmin) / 2);
deltaC3 = round((Cmax - Cmin) / 3);
deltaC = Cmax - Cmin;
deltaR = Rmax - Rmin;

deltaRun = 5; % round(dt/4);

% initializing particles in all groups
for g = 0:(group - 1)
    for i = 1:n
        partInd = g*n + i;
        particle(partInd).posn=[round((prev_left_mouth(1) - deltaRun) +
2*deltaRun * rand(1)) round((prev_left_mouth(2) - deltaRun) + 2*deltaRun * rand(1))];
        particle(partInd).vel=[round(Vmin+(Vmax-Vmin)*rand(1))
round(Vmin+(Vmax-Vmin)*rand(1))];

        % check if the particle isn't too close to the lower edge of the
image,

        % so that the lower edge of the template would get out of the bounds.
if particle(partInd).posn(1) > (255 - (round(dt / 2)))
            particle(partInd).posn(1) = 255 - (round(dt / 2));
        end

        particle(partInd).Pbest = particle(partInd).posn;
        particle(partInd).func_response =
cost_functionMouthL2(I,particle(partInd).posn(1),
particle(partInd).posn(2),dt,center,meanInt, thresh.lips);
        particle(partInd).func_resp_prev = particle(partInd).func_response;
    end

    bestfind = find([particle((g*n + 1):(g*n + n)).func_response] ==
max([particle((g*n + 1):(g*n + n)).func_response]));
    best(g+1) = g*n + bestfind(1);
    Gbest((2*g + 1):(2*g + 2)) = particle(best(g+1)).posn;
    GbestFit(g+1) = particle(best(g+1)).func_response;
    MaxCord((2*g + 1):(2*g + 2)) = Gbest((2*g + 1):(2*g + 2));
    best_response(g+1) = particle(best(g+1)).func_response;
    MaxFit(g+1) = best_response(g+1);
end
end

```



```

itr=1;
t=1;
thresh_list=[];

while ind == 0 && itr < 22
    %Compute particles new position and velocities

    % if GbestFit > 0 compute new position and velocities - algorithm
    % is converging.
    if (GbestFit(g+1) > 0)
        %In each group
        for g = 0:(group-1)
            for i=1:n
                partInd = g*n + i;

particle(partInd).vel(1)=wi*particle(partInd).vel(1)+c1*rand(1)*(particle(partInd).Pbest(1) - particle(partInd).posn(1))+ c2*rand(1)*(Gbest(2*g + 1)-
particle(partInd).posn(1));

particle(partInd).vel(2)=wi*particle(partInd).vel(2)+c1*rand(1)*(particle(partInd).Pbest(2) - particle(partInd).posn(2))+ c2*rand(1)*(Gbest(2*g + 2)-
particle(partInd).posn(2));

particle(partInd).posn(1)=round(particle(partInd).posn(1)+particle(partInd).vel(1));

particle(partInd).posn(2)=round(particle(partInd).posn(2)+particle(partInd).vel(2));

                % check if the particle isn't too close to the lower edge of
the image,
                % so that the lower edge of the template would get out of the
bounds.
                % this check might be together with the following check of
particle
                % position
                if particle(partInd).posn(1) > (255 - (round(dt / 2)))
                    particle(partInd).posn(1) = 255 - (round(dt / 2));
                end

                %checking if the solution lies within domain
            end
        end
    end
end

```

```

        if particle(partInd).posn(1) <Rmin || particle(partInd).posn(1)
> Rmax || particle(partInd).posn(2) <Cmin || particle(partInd).posn(2) > Cmax
            particle(partInd).posn = particle(partInd).Pbest;    %
setting position to previous P-best
            particle(partInd).vel=[rand(1)*Vmax rand(1)*Vmax];
% setting velocity to max
        end

        %Calculating the function response for each particle with new
%positions

particle(partInd).func_response=cost_functionMouthL2(I,particle(partInd).posn(1),
particle(partInd).posn(2),dt,center,meanInt, thresh.lips);

        %checking and updating Pbest
        if particle(partInd).func_response >
particle(partInd).func_resp_prev
            particle(partInd).Pbest = particle(partInd).posn;
        end

        particle(partInd).func_resp_prev =
particle(partInd).func_response;

    end
end

V = [particle((g*n + 1):(g*n + n)).func_response];
best_ones = find(V == max(V));
best(g+1) = g*n + best_ones(1);

%Gbest is a global best of all times
if particle(best(g+1)).func_response > GbestFit(g+1)
    Gbest((2*g + 1):(2*g + 2)) = particle(best(g+1)).Pbest;
    GbestFit(g+1) = particle(best(g+1)).func_response;
end
best_response(g+1) = particle(best(g+1)).func_response;
if (best_response(g+1) < MaxFit(g+1))
    MaxCord((2*g + 1):(2*g + 2)) = Gbest((2*g + 1):(2*g + 2));
end
else
% if is not > 0 than deploy te particles again in new random
% positions and with new random speeds

```

```

if (itr < 22)
    for g = 0:(group-1)
        for i=1:n
            partInd = g*n + i;
            particle(partInd).posn=[round((prev_left_mouth(1) -
deltaRun) + 2*deltaRun * rand(1)) round((prev_left_mouth(2) - deltaRun) + 2*deltaRun *
rand(1))];

            particle(partInd).vel=[round(Vmin+(Vmax-Vmin)*rand(1))
round(Vmin+(Vmax-Vmin)*rand(1))];

            % check if the particle isn't too close to the lower edge
of the image,
            % so that the lower edge of the template would get out of
the bounds.

            if particle(partInd).posn(1) > (255 - (round(dt / 2)))
                particle(partInd).posn(1) = 255 - (round(dt / 2));
            end

            particle(partInd).Pbest=particle(partInd).posn;

particle(partInd).func_response=cost_functionMouthL2(I,particle(partInd).posn(1),
particle(partInd).posn(2),dt,center,meanInt, thresh.lips);

particle(partInd).func_resp_prev=particle(partInd).func_response;
            end
        end
    end

    bestfind = find([particle((g*n + 1):(g*n + n)).func_response] ==
max([particle((g*n + 1):(g*n + n)).func_response]));
    best(g+1) = g*n + bestfind(1);
    Gbest((2*g + 1):(2*g + 2)) = particle(best(g+1)).posn;
    GbestFit(g+1) = particle(best(g+1)).func_response;
    MaxCord((2*g + 1):(2*g + 2)) = Gbest((2*g + 1):(2*g + 2));
    best_response(g+1) = particle(best(g+1)).func_response;
    MaxFit(g+1) = best_response(g+1);
    t=1;
    thresh_list=[];

end % all particles updated...

%...recording co-ordinates for display

```

```

    itr = itr+1;
    thresh_list=[thresh_list best_response];
    if itr == 20
        ind = 1;
    end
end
end
% choose the best point
very_best_ones = find( GbestFit == max(GbestFit));
very_best = very_best_ones(1);
VeryBest = Gbest(2*(very_best - 1) + 1: 2*(very_best - 1) + 2);
GbestFit(very_best) = [];
Gbest(2*(very_best - 1) + 1: 2*(very_best - 1) + 2) = [];
very_best_ones = find( GbestFit == max(GbestFit));
very_best = very_best_ones(1);
VeryBest = [VeryBest Gbest(2*(very_best - 1) + 1: 2*(very_best - 1) + 2)];
GbestFit(very_best) = [];
Gbest(2*(very_best - 1) + 1: 2*(very_best - 1) + 2) = [];
VeryBest = [VeryBest Gbest];

```

Filename : Pso2_mouth_right_run.m

This is the particle swarm optimization algorithm used to locate the eyes while testing the whole video sequence.

```
function [VeryBest M]=Pso2_mouth_right_run(Rmin, Rmax, Cmin, Cmax, I, center,
prev_right_mouth)

n = 10; % number of particles
group = 3; % number of groups
Vmax= 8; % Max velocity..set arbitrarily
Vmin= 3; % particle move one pixel at a time at minimum
Gbest=[0 0 0 0 0 0];
GbestFit = [0 0 0];
MaxFit = [0 0 0];
MaxCord = [0 0 0 0 0 0];
best_response = [0 0 0];
best = [0 0 0];
VeryBest = [0 0];

wi = 0.85; %inertial weight
c1=0.5;
c2=0.5;

ind=0;
coord=[];
prev_best_values=[];
threshold = 400;
dt = floor((Cmax - Cmin)/2);
meanInt = mean2(I(Rmin:Rmax,Cmin:Cmax));

%initialising particles with random velocities and setting
%Pbest to the initial position
%Deploying the particles into an region where the eye is expected.

deltaR4 = round((Rmax - Rmin) / 4);
deltaR2 = round((Rmax - Rmin) / 2);
deltaC2 = round((Cmax - Cmin) / 2);
deltaC3 = round((Cmax - Cmin) / 3);
deltaC = Cmax - Cmin;
deltaR = Rmax - Rmin;
```

```

deltaRun = round(dt/4);

% initializing particles in all groups
for g = 0:(group - 1)
    for i = 1:n
        partInd = g*n + i;
        particle(partInd).posn=[round((prev_right_mouth(1) - deltaRun) +
2*deltaRun * rand(1)) round((prev_right_mouth(2) - deltaRun) + 2*deltaRun * rand(1))];
        particle(partInd).vel=[round(Vmin+(Vmax-Vmin)*rand(1))
round(Vmin+(Vmax-Vmin)*rand(1))];

        % check if the particle isn't too close to the lower edge of the
image,

        % so that the lower edge of the template would get out of the bounds.
if particle(partInd).posn(1) > (255 - (round(dt / 2)))
            particle(partInd).posn(1) = 255 - (round(dt / 2));
        end

        particle(partInd).Pbest = particle(partInd).posn;
        particle(partInd).func_response =
cost_functionMouthR2(I,particle(partInd).posn(1),
particle(partInd).posn(2),dt,center,meanInt);
        particle(partInd).func_resp_prev = particle(partInd).func_response;
    end

    bestfind = find([particle((g*n + 1):(g*n + n)).func_response] ==
max([particle((g*n + 1):(g*n + n)).func_response]));
    best(g+1) = g*n + bestfind(1);
    Gbest((2*g + 1):(2*g + 2)) = particle(best(g+1)).posn;
    GbestFit(g+1) = particle(best(g+1)).func_response;
    MaxCord((2*g + 1):(2*g + 2)) = Gbest((2*g + 1):(2*g + 2));
    best_response(g+1) = particle(best(g+1)).func_response;
    MaxFit(g+1) = best_response(g+1);
end

itr=1;
t=1;
thresh_list=[];

while ind == 0 && itr < 22

```

```

%Compute particles new position and velocities

% if GbestFit > 0 compute new position and velocities - algorithm
% is converging.
if (GbestFit(g+1) > 0)
    %In each group
    for g = 0:(group-1)
        for i=1:n
            partInd = g*n + i;

particle(partInd).vel(1)=wi*particle(partInd).vel(1)+c1*rand(1)*(particle(partInd).Pbest(1) - particle(partInd).posn(1))+ c2*rand(1)*(Gbest(2*g + 1) - particle(partInd).posn(1));

particle(partInd).vel(2)=wi*particle(partInd).vel(2)+c1*rand(1)*(particle(partInd).Pbest(2) - particle(partInd).posn(2))+ c2*rand(1)*(Gbest(2*g + 2) - particle(partInd).posn(2));

particle(partInd).posn(1)=round(particle(partInd).posn(1)+particle(partInd).vel(1));

particle(partInd).posn(2)=round(particle(partInd).posn(2)+particle(partInd).vel(2));

            % check if the particle isn't too close to the lower edge of
the image,
            % so that the lower edge of the template would get out of the
bounds.
            % this check might be together with the following check of
particle
            % position
            if particle(partInd).posn(1) > (255 - (round(dt / 2)))
                particle(partInd).posn(1) = 255 - (round(dt / 2));
            end

            %checking if the solution lies within domain
            if particle(partInd).posn(1) <Rmin || particle(partInd).posn(1)
> Rmax || particle(partInd).posn(2) <Cmin || particle(partInd).posn(2) > Cmax
                particle(partInd).posn = particle(partInd).Pbest; %
                setting position to previous P-best
                particle(partInd).vel=[rand(1)*Vmax rand(1)*Vmax];
            % setting velocity to max
            end

```

```

        %Calculating the function response for each particle with new
        %positions

particle(partInd).func_response=cost_functionMouthR2(I,particle(partInd).posn(1),
particle(partInd).posn(2),dt,center,meanInt);

        %checking and updating Pbest
        if particle(partInd).func_response >
particle(partInd).func_resp_prev
            particle(partInd).Pbest = particle(partInd).posn;
        end

        particle(partInd).func_resp_prev =
particle(partInd).func_response;

    end
end

V = [particle((g*n + 1):(g*n + n)).func_response];
best_ones = find(V == max(V));
best(g+1) = g*n + best_ones(1);

% Gbest is a global best of all times
% display(particle(best(g+1)).func_response);
if particle(best(g+1)).func_response > GbestFit(g+1)
    Gbest((2*g + 1):(2*g + 2)) = particle(best(g+1)).Pbest;
    GbestFit(g+1) = particle(best(g+1)).func_response;
end
best_response(g+1) = particle(best(g+1)).func_response;
if (best_response(g+1) < MaxFit(g+1))
    MaxCord((2*g + 1):(2*g + 2)) = Gbest((2*g + 1):(2*g + 2));
end

else
% if is not > 0 than deploy te particles again in new random
% positions and with new random speeds
    if (itr < 5)
        for g = 0:(group-1)
            for i=1:n
                partInd = g*n + i;

```



```

        particle(partInd).posn=[round((prev_right_mouth(1) -
deltaRun) + 2*deltaRun * rand(1)) round((prev_right_mouth(2) - deltaRun) + 2*deltaRun
* rand(1))];

        particle(partInd).vel=[round(Vmin+(Vmax-Vmin)*rand(1))
round(Vmin+(Vmax-Vmin)*rand(1))];

        % check if the particle isn't too close to the lower edge
of the image,
        % so that the lower edge of the template would get out of
the bounds.

        if particle(partInd).posn(1) > (255 - (round(dt / 2)))
            particle(partInd).posn(1) = 255 - (round(dt / 2));
        end

        particle(partInd).Pbest=particle(partInd).posn;

particle(partInd).func_response=cost_functionMouthR2(I,particle(partInd).posn(1),
particle(partInd).posn(2),dt,center,meanInt);

particle(partInd).func_resp_prev=particle(partInd).func_response;
        end
        end
    else
        % if we didn't find a positive fitness value before the
        % 5th iteration that the region for deployment is
        % extended.
        % it helps in the cases, when the face region is not
        % located well
        for g = 0:(group-1)
            for i=1:n
                partInd = g*n + i;

                particle(partInd).posn=[round((prev_right_mouth(1) -
2*deltaRun) + 4*deltaRun * rand(1)) round((prev_right_mouth(2) - 2*deltaRun) +
4*deltaRun * rand(1))];

                particle(partInd).vel = [round(Vmin+(Vmax-Vmin)*rand(1))
round(Vmin+(Vmax-Vmin)*rand(1))];

                % check if the particle isn't too close to the lower edge
of the image,

```

```

% so that the lower edge of the template would get out of
the bounds.

if particle(partInd).posn(1) > (255 - (round(dt / 2)))
    particle(partInd).posn(1) = 255 - (round(dt / 2));
end

particle(partInd).Pbest = particle(partInd).posn;
particle(partInd).func_response =
cost_functionMouthR2(I,particle(partInd).posn(1),
particle(partInd).posn(2),dt,center,meanInt);
particle(partInd).func_resp_prev =
particle(partInd).func_response;

end
end
end

bestfind = find([particle((g*n + 1):(g*n + n)).func_response] ==
max([particle((g*n + 1):(g*n + n)).func_response]));
best(g+1) = g*n + bestfind(1);
Gbest((2*g + 1):(2*g + 2)) = particle(best(g+1)).posn;
GbestFit(g+1) = particle(best(g+1)).func_response;
MaxCord((2*g + 1):(2*g + 2)) = Gbest((2*g + 1):(2*g + 2));
best_response(g+1) = particle(best(g+1)).func_response;
MaxFit(g+1) = best_response(g+1);
t=1;
thresh_list=[];

end % all particles updated...

%...recording co-ordinates for display

itr = itr+1;
thresh_list=[thresh_list best_response];
if itr == 20
    ind = 1;
end
end

% choose the best point
very_best_ones = find( GbestFit == max(GbestFit));
very_best = very_best_ones(1);
VeryBest = Gbest(2*(very_best - 1) + 1: 2*(very_best - 1) + 2);

```

```
GbestFit(very_best) = [];  
Gbest(2*(very_best - 1) + 1: 2*(very_best - 1) + 2) = [];  
very_best_ones = find( GbestFit == max(GbestFit));  
very_best = very_best_ones(1);  
VeryBest = [VeryBest Gbest(2*(very_best - 1) + 1: 2*(very_best - 1) + 2)];  
GbestFit(very_best) = [];  
Gbest(2*(very_best - 1) + 1: 2*(very_best - 1) + 2) = [];  
VeryBest = [VeryBest Gbest];
```

Filename : cost_functionMouthR2.m

This is the cost function used to locate the right corner of the mouth while testing the whole video sequence.

```
function y = cost_functionMouthR2(I,r,c,dtIn,center,meanInt)
% cost function for looking for the right corner point of mouth

global Im;
Im = I;
global dt;
dt = dtIn;

K = 50;

y = CMouth(r,c,meanInt);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function a = CMean(r,c)
% function calculating the mean intensity of all points within the template

global Im;
global dt;
I1=double(Im( r - floor(dt / 15) : r + floor(dt / 15), c - floor(dt / 7) :
c ));
a = (255 - mean2(I1));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function a = CMouth(r,c,meanInt)
global Im;
global dt;
a = 0;
I1 = double(Im( (r - floor(dt / 10) : r + floor(dt / 10)), c - floor(dt/2) :
c ));
I2 = double(Im( (r - floor(dt / 10) : r + floor(dt / 10)), c : c +
floor(dt/4)));
I3 = double(Im( (r - floor(dt / 3) : r - floor(dt / 15)), c - floor(dt / 7) :
c ));
I4 = double(Im( (r + floor(dt / 15) : r + floor(dt / 3)), c - floor(dt / 7) :
c ));
```

```

I5 = double(Im( (r - floor(dt / 4) : r + floor(dt / 4)), c : c + floor(dt /
5)));
I6 = double(Im( (r - floor(dt / 15) : r + floor(dt / 15)), c - floor(dt/10) :
c ));
I7 = double(Im( (r - floor(dt / 15) : r + floor(dt / 15)), c : c +
floor(dt/10)));

meanI1 = mean2(I1);
meanI2 = mean2(I2);
meanI3 = mean2(I3);
meanI4 = mean2(I4);
meanIc = CMean(r,c);
meanI5 = mean2(I5);
meanI6 = mean2(I6);
meanI7 = mean2(I7);

b = CMouthTemplate(r,c,meanInt);

if (meanI1 < 0.8 * meanInt * thresh_lip) && (meanI2 > 0.8 * meanInt *
thresh_lip)
    a = a + (meanI2 - meanI1);
    a = a + meanIc;
    a = a + (meanI7 - meanI6);
    if ((meanI3 - meanIc) > 50 ) && ((meanI4 - meanIc) > 50)
        a = a + 100;
    end;
else
    a = a - 200;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function a = CMouthTemplate(r,c,meanInt)
% function trying to calcute if the region is really mouth. Calculates the
% avarage intensity of rectangular template

global Im;
global dt;

I1 = double(Im((r - floor(dt/6) : r + floor(dt/2)), c - dt : c ));

a = mean2(I1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
function a = CPos(r,c,center)
% function evaluating if the given point lies under the eye

a = (-2) * abs(center - c);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Filename : Pso_nose_run_vector.m

This is the particle swarm optimization algorithm used to locate the nose while testing the whole video sequence.

```
function [Gbest M] = Pso_nose_run_vector(Rmin,Rmax,Cmin,Cmax,I,width,
prev_nose, eye_move_vector, thresh_nose)

% PSO algorithm for extracting the nose midpoint coordinates

n=5;%no of particles
Vmax= 5; % Max velocity..set arbitrarily
Vmin= 3; % particle move one pixel at a time at minimum
Gbest=[0 0];
GbestFit = 0;
MaxFit = 0;
MaxCord = [0 0];

wi = 0.8; %inertial weight
c1=0.5;
c2=0.5;

ind=0;
coord=[];
prev_best_values=[];
threshold=400;
dt = floor(width/4);
center = Rmin + (Rmax - Rmin)/2.0;

%initialising particles with random velocities and setting
%Pbest to the initial position
%Deploying the particles into an region where the nose is expected.

deltaR2 = round((Rmax - Rmin) / 2);
deltaR4 = round((Rmax - Rmin) / 4);
deltaC2 = round((Cmax - Cmin) / 2);
deltaC4 = round((Cmax - Cmin) / 4);
deltaC = Cmax - Cmin;
deltaR = Rmax - Rmin;

deltaRun = 5;
```

```

nose_moved = prev_nose(1) - eye_move_vector;

% new borders of the nose region based on the vector of eye movement

Rmin = nose_moved - round(dt/4);
Rmax = nose_moved + round(dt/4);

Cmin = prev_nose(2) - round(dt/4);
Cmax = prev_nose(2) + round(dt/4);

for i=1:n
    particle(i).posn=[round((nose_moved - deltaRun) + 2*deltaRun * rand(1))
round((prev_nose(2) + deltaRun) + 2*deltaRun * rand(1))];
    particle(i).vel=[round(Vmin+(Vmax-Vmin)*rand(1)) round(Vmin+(Vmax-
Vmin)*rand(1))];
    particle(i).Pbest=particle(i).posn;

particle(i).func_response=cost_functionNose(I,particle(i).posn(1),particle(i).posn(2),
dt,center, thresh_nose);
    particle(i).func_resp_prev=particle(i).func_response;
    % I(particle(i).posn(1),particle(i).posn(2))=250;
end

% initiating values
bestfind = find([particle.func_response] == max([particle.func_response]));
best = bestfind(1);
Gbest=particle(best).posn;
GbestFit = particle(best).func_response;
MaxCord = Gbest;
best_response=particle(best).func_response;
MaxFit = best_response;

itr=1;
t=1;
thresh_list=[];

while ind == 0 && itr < 22

    %Compute particles new position and velocities
    % if GbestFit > 0 compute new position and velocities - algorithm
    % is converging.

```



```

if (GbestFit > 0)
    for i=1:n
        % set new velocities

particle(i).vel(1)=wi*particle(i).vel(1)+c1*rand(1)*(particle(i).Pbest(1)-
particle(i).posn(1))+ c2*rand(1)*(Gbest(1)-particle(i).posn(1));

particle(i).vel(2)=wi*particle(i).vel(2)+c1*rand(1)*(particle(i).Pbest(2)-
particle(i).posn(2))+ c2*rand(1)*(Gbest(2)-particle(i).posn(2));

        % set new position
particle(i).posn(1)=round(particle(i).posn(1)+particle(i).vel(1));
particle(i).posn(2)=round(particle(i).posn(2)+particle(i).vel(2));

        % checking if the solution lies within domain

        if particle(i).posn(1) <Rmin || particle(i).posn(1) > Rmax ||
particle(i).posn(2) <Cmin || particle(i).posn(2) > Cmax
            particle(i).posn=particle(i).Pbest;    % setting position to
previous P-best
            particle(i).vel=[rand(1)*Vmax rand(1)*Vmax];    %
setting velocity to max
        end

        % Calculating the function response for each particle with new
% positions

particle(i).func_response=cost_functionNose(I,particle(i).posn(1),particle(i).posn(2),
dt,center, thresh_nose);

        %checking and updating Pbest
if particle(i).func_response > particle(i).func_resp_prev
            particle(i).Pbest=particle(i).posn;
        end

particle(i).func_resp_prev = particle(i).func_response;

    end
V=[particle.func_response];
best_ones=find(V==max(V));
best=best_ones(1);

```

```

%Gbest is a global best of all times
%display(particle(best).func_response);
if particle(best).func_response > GbestFit
    Gbest = particle(best).Pbest;
    GbestFit = particle(best).func_response;
end
best_response=particle(best).func_response;
if (best_response < MaxFit)
    MaxCord = Gbest;
end
else
% if is not > 0 than deploy te particles again in new random
% positions and with new random speeds
    if (itr < 22)
        for i=1:n
            particle(i).posn=[round((nose_moved - deltaRun) + 2*deltaRun
* rand(1)) round((prev_nose(2) - deltaRun) + 2*deltaRun * rand(1))];
            particle(i).vel=[round(Vmin+(Vmax-Vmin)*rand(1))
round(Vmin+(Vmax-Vmin)*rand(1))];
            particle(i).Pbest=particle(i).posn;

particle(i).func_response=cost_functionNose(I,particle(i).posn(1),particle(i).posn(2),
dt,center);

            particle(i).func_resp_prev=particle(i).func_response;
        end

    end

    best=max_position([particle.func_response]);
    Gbest=particle(best).posn;
    GbestFit = particle(best).func_response;
    MaxCord = Gbest;
    best_response=particle(best).func_response;
    MaxFit = best_response;
    t=1;
    thresh_list=[];

end % all particles updated...

% 2 - based on the mean value of func_response of all particles
if mean([particle.func_response]) >= 1100;%threshold;
    ind = 1;
end

```

```
%...recording co-ordinates for display

itr = itr+1;
thresh_list=[thresh_list best_response];
if itr == 20
    ind = 1;
end
end

if itr == 22
    Gbest = MaxCord;
End
```

Appendix B - Facial Action Coding System

Figure B.1 Anatomy of Facial Muscles

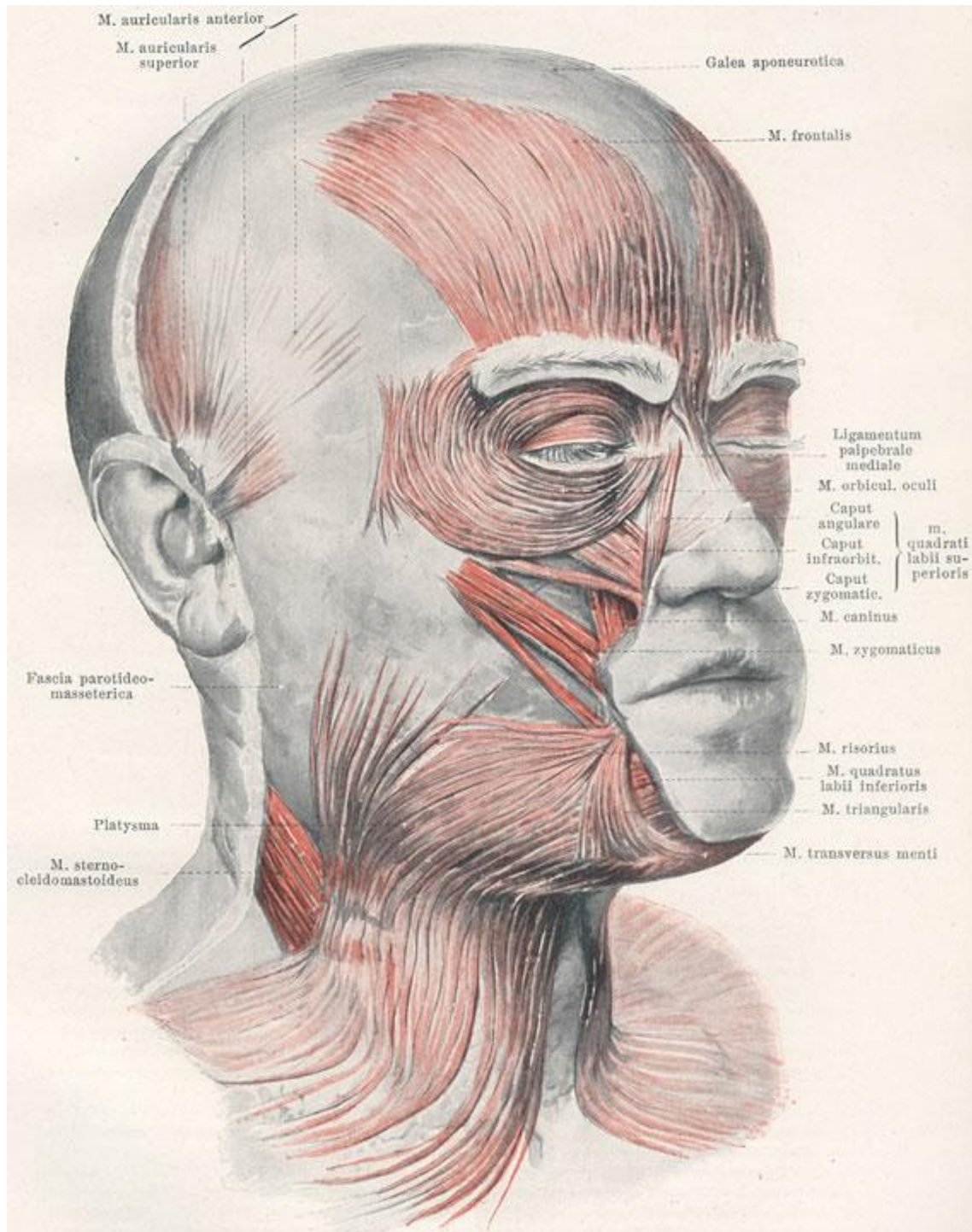












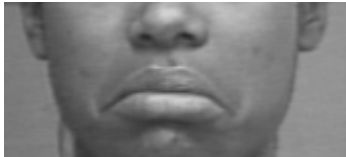






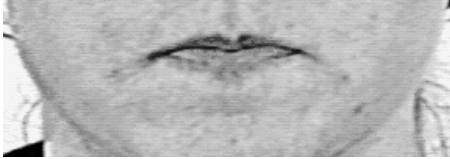










Table B.1 Facial Action Coding System








Source URL: <http://www.cs.cmu.edu/afs/cs/project/face/www/facs.htm> (05/09/2008)






AU	Description	Facial muscle	Example image
1	Inner Brow Raiser	<i>Frontalis, pars medialis</i>	
2	Outer Brow Raiser	<i>Frontalis, pars lateralis</i>	
4	Brow Lowerer	<i>Corrugator supercilii, Depressor supercilii</i>	
5	Upper Lid Raiser	<i>Levator palpebrae superioris</i>	
6	Cheek Raiser	<i>Orbicularis oculi, pars orbitalis</i>	
7	Lid Tightener	<i>Orbicularis oculi, pars palpebralis</i>	
9	Nose Wrinkler	<i>Levator labii superioris alaquae nasi</i>	
10	Upper Lip Raiser	<i>Levator labii superioris</i>	

11	Nasolabial Deepener	<i>Zygomaticus minor</i>	
12	Lip Corner Puller	<i>Zygomaticus major</i>	
13	Cheek Puffer	<i>Levator anguli oris</i> (a.k.a. <i>Caninus</i>)	
14	Dimpler	<i>Buccinator</i>	
15	Lip Corner Depressor	<i>Depressor anguli oris</i> (a.k.a. <i>Triangularis</i>)	
16	Lower Lip Depressor	<i>Depressor labii inferioris</i>	
17	Chin Raiser	<i>Mentalis</i>	

18	Lip Puckerer	<i>Incisivii labii superioris and Incisivii labii inferioris</i>	
20	Lip stretcher	<i>Risorius w/ platysma</i>	
22	Lip Funneler	<i>Orbicularis oris</i>	
23	Lip Tightener	<i>Orbicularis oris</i>	
24	Lip Pressor	<i>Orbicularis oris</i>	
25	Lips part	<i>Depressor labii inferioris or relaxation of Mentalis, or Orbicularis oris</i>	
26	Jaw Drop	<i>Masseter, relaxed Temporalis and internal Pterygoid</i>	

27	Mouth Stretch	<i>Pterygoids, Digastric</i>	
28	Lip Suck	<i>Orbicularis oris</i>	
41	Lid droop	<i>Relaxation of Levator palpebrae superioris</i>	
42	Slit	<i>Orbicularis oculi</i>	
43	Eyes Closed	<i>Relaxation of Levator palpebrae superioris; Orbicularis oculi, pars palpebralis</i>	
44	Squint	<i>Orbicularis oculi, pars palpebralis</i>	
45	Blink	<i>Relaxation of Levator palpebrae superioris; Orbicularis oculi, pars palpebralis</i>	
46	Wink	<i>Relaxation of Levator palpebrae superioris; Orbicularis oculi, pars palpebralis</i>	

51	Head turn left		
52	Head turn right		
53	Head up		
54	Head down		
55	Head tilt left		
56	Head tilt right		
57	Head forward		

58	Head back		
61	Eyes turn left		
62	Eyes turn right		
63	Eyes up		
64	Eyes down		

Appendix C - Neural Network as a Useful Tool for Real-Time Facial Expression Recognition

Linda O., Chandrapati S., Tokuhiro A., 2007, "Neural Network as a Useful Tool for Real-Time Facial Expression Recognition", *Proceedings of 17th Conference on Artificial Neural Networks in Engineering*, St Louis, USA, Nov 2007.

NEURAL NETWORKS AS A USEFUL TOOL FOR REAL-TIME FACIAL EXPRESSION RECOGNITION

ONDREJ LINDA

Graduate student
Computing and Information
Sciences,
Kansas State University,
Manhattan, Kansas, USA
olinda@ksu.edu

SRIVARDHAN CHANDRAPATI

Graduate student
Mechanical and Nuclear
Engineering,
Kansas State University,
Manhattan, Kansas, USA
csri@ksu.edu

AKIRA TOKUHIRO

Associate Professor,
Mechanical Engineering,
University of Idaho,
Idaho Falls, Idaho, USA
tokuhiro@uidaho.edu

ABSTRACT

Human beings communicate by rather skillful projection and reception of facial and voiced-expressions, as well as gestures (hand, posture, etc.). In order to develop “tools” by which humans can interact with a computer, we need to develop a software tool that is capable of processing, for example a wide range of facial expressions (FEs) in time. Facial expressions, in their variety and in addition, digital quality when captured presents itself as nontrivial challenge in applied biometrics. In this paper we present results to date on development of a real time and computationally “light” facial expression recognition software tool. Contrary to past studies reporting elaborate processing and FE classification methods, we undertook an approach extracting a small number of facial feature points, and one that realistically contained noise. We additionally proposed a vectoral descriptor and amplitude for a given FE, relative to a reference, neutral FE image. Moreover we trained and tested across FE databases that were ethnically/culturally and gender-wise different. Our results to date opened for consideration the following in brief, that: 1) there are similarities/differences across FE database, 2) vectoral descriptors and amplitude per FE appear effective, 3) “happy” and “surprise” FEs are well classified across different database, while 4) “angry-distress” and “fear-surprise” pairs are linked by misclassification across databases; that is, there is some evidence that these are ethnicity/culture specific and therefore misinterpreted.

INTRODUCTION

During one-on-one communication it is widely accepted that significant amount of information is conveyed through non-verbal communication, such as gestures or facial expressions (Mehrabian, 1968). Developing a software tool for automatic facial expression recognition would significantly improve the communication between human and computer not only during normal use, but also when communicating with an handicapped person.

Amongst an wide range of functional skills, we often take our ability to recognize and project facial expressions for granted, this in fact is a challenge when tried to achive using computers. The range of human expressions and the cognitive state to which the expression is attached is vast, even a simple expression such as a simle is different form person to person. Additionally, there is some evidence of differences amongst different ethnic groups and cultures. However, as humans are able to distinguish between basic expressions/emotions irrespective of ethnicity and culture, it is reasonable to assert that there exists a common functional modality for each expression. We believe that discovering this modality and developing a tool to accurately extract an inherent "pattern" from an expression is the key to building flexible and robust facial expression recognition software.

Automated facial expression recognition has been proposed by a number of investigators; a notable, detailed survey is that given by Fasel and Luetttin (2003) and also by Pantic and Rothkrantz (2003). However, most of the approaches are far from being applicable in a real time environment except perhaps for that developed by Littlewort et al. (2003). This is mainly due to the limitation on computing power; that is, considerable computing is needed for detailed image preprocessing and feature point extraction. In the present work, our objective is to develop a useful real time application that will be capable of a relatively accurate classification of the facial expressions. Because each captured frame needs to be processed as quickly as possible, we plan to limit the processing time per image. This poses a number of challenges. First of all we cannot extract a large number of (facial) feature points. Instead we search for the most distinguishable points in the face which can consistently be tracked spatiotemporally (frame-to-frame if needed) with high accuracy. Secondly, we exercise the option to limit the search for feature points if an initial "pass" does not easily reveal them. Thus, at the onset we expect a lower overall accuracy in facial expression recognition and from a learning point of view (from a dataset), we acknowledge the existence of noise in our data.

It is here at the pre-processing stage that we significantly differ from Littlewort et al. (2006) where they began with the whole face region to get 92,160 possible features which they later reduced to a minimum of 500 features using machine learning techniques. Under such circumstances they found an optimum set of facial features, but without specifics on what features are being learned and are important. Based on Bassili's (1978) experiment, we instead choose to select points on the face.

In the present work we will introduce results to date of work toward developing a real time facial expression recognition tool. We used a feed-forward Artificial Neural Network (ANN) trained with back propagation training using gradient descent with momentum rule, to adapt to the characteristics of our facial expression image database. The ANN was built using the neural network toolkit in MATLAB™. In this paper we present our results on classifying captured facial expressions from snapshot and video frames, extracted from ethnically different databases.

LEARNING

One of the most common techniques for facial expression classification is via Artificial Neural Network (ANN). Based on our previous work on facial expression biometrics (Chennamsetty and Tokuhiko, 2004; Chennamsetty, 2005), we sought to use ANN to perform (quasi) real-time FE-classification.

There are two basic approaches to classifying expressions, one being identifying the presence of each (facial) action unit (AU) as defined by FACS (Ekman and Friesen, 1978) and then classifying based upon combinations of AUs, and the other, classifying the FE into one of the six basic emotions as defined by Ekman (1992). Here, we chose to classify the given face into one of the six basic expressions: Anger, Distress, Fear, Happy, Sad and Surprise. The tracking of AUs requires high resolution and standardization of images, which is ultimately far from reality. We also recognize that *precisely* defining a FE is difficult (perhaps irrelevant) and it is often unrealistic (questionable) to seek a “pure” FE. In fact, as FEs are highly spatiotemporal, each given expression can be a mixture of several FEs. Thus, ideally the ANN should output its “degree of belief/confidence” in how much a given FE is similar to one of our six expressions. A feed-forward ANN trained with back propagation learning rule is capable of giving such a response (Kobayashi and Hara, 1992).



Figure 1. Images from the JAFFE database. You can see the six basic expressions that we are trying to classify. From left to right: Angry (An), Distress (Di), Fear (Fe), Happy (Ha), Sad (Sa), Surprise (Su).

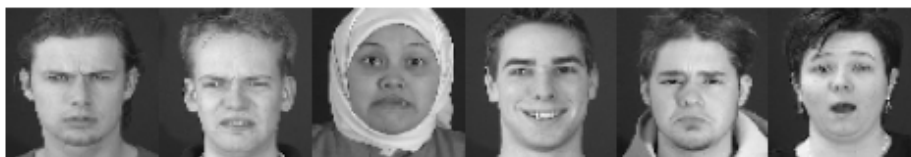


Figure 2. Frames from the video sequences from the MMI database. From left to right: Angry, Distress, Fear, Happy, Sad, Surprise.

INPUT DATA

For our experiments we used two facial image databases. The first, known as the JAFFE database, consists of 213 images of 10 Japanese females (Lyons, 1998). The second database, known as MMI, consists of video sequences of 15 different subjects (Pantic et al., 2005). The MMI subjects are mostly young white males and females. Each video sequence starts with a neutral face and then slowly transitions to an expression and back to a neutral face.

Feature Points Extraction

Although some may contest Johansson's (1973) point light experiment and Bassili's (1978) experiments based on Johansson's work, there is evidence that

extraction of only the key facial features or points are enough for classification of FEs. Further, as we are cognizant of in-the-field application issues, we decided to locate 17 of the most distinguishable feature points on the eyebrows, eyes, nose and mouth, as shown in the Fig. 3. To locate these points we used an algorithm consisting of three main stages; 1) *Face region estimation*, 2) *Feature region detection*, and 3) *Feature point detection*. In the first stage, face region location is estimated using seeded region growing method. This face region information is passed to the feature region detection algorithm that identifies key points of the eyes, mouth, nose and eyebrow using a Particle Swarm Optimization-based search. The feature points associated with these regions are defined based on the assumption that the feature is continuous at least along its edge and separated from other features, even if large variations appear within the feature itself. This is accomplished by using the bounding box function built in MatlabTM.



Figure 3. Face with located feature points that we select on the face.

Descriptors

In Fig. 3, the absolute coordinates of the feature points only tell us where they are located in the image. Rather one needs to construct ‘a descriptor’ that describes the position of the feature points with respect to the other points, for a given FE. This descriptor is also important in order to compensate for head movement and differences in placement of the face in the field-of-view (frame). In other words, we want to extract information about the feature points’ movement on the face. We constructed a descriptor ‘vector’ consisting of 42 different distances, angles and vectors among the points. This descriptor vector served as the input vector for the ANN.

Tracking changes in expression

Rather than processing each FE image as single entity, in the present work we focused our interest on changes in FE between two images; that is, what Cohen et al. (2003) called static processing. As FEs constantly change, we sought to extract information like whether the corners of the mouth move relative to one another or whether the eyebrows are rising relative to the mouth. Our goal throughout was to classify the expression into one of the six accepted FE groups. Therefore we generated a descriptor vector for a given FE image and compared it with the descriptor vector of the neutral FE for the subject. In this manner we derive the change in descriptor vector relative to the neutral vector FE for the subject. For the JAFFE database we have three neutral FE images for each subject. So we took each of these as a reference neutral descriptor and calculated difference vectors between these reference vectors and the FE vector

of interest. Where-as in the MMI video-frame sequence we first split the video into individual frames and took the first FE frame (always neutral) in the sequence as the reference and then calculated the relative changes between subsequent frames relative to this reference. We thus generated a sequence of vectoral descriptors tracking the spatio-temporality of FEs.

Target function

In order to train the ANN we constructed a training set containing labeled instances. In the case of the JAFFE database we assigned the same apriori expression label from the database to the derived descriptor relative to the reference neutral FE. However, for the MMI database, lacking apriori labels, we devised an approach as follows. Here we sought to attach one of the six FE labels to each frame from the video sequence. The question is what label we should assign to each FE frame, as the expression changes in time from a neutral face. In addition, in some frame-to-frame sequence, we had to assign incrementally different FE labels; that is, at some point, one frame for instance, was labeled 'neutral' and the next 'happy', even if it represented the first of several images expressing a happy FE. We recognize that this incremental evaluation introduces some noise into the database and may influence the training exercise.

EXPERIMENTAL RESULTS

In considering ANN methods, we initially used a feed forward NN with back propagation training rule. We began with the JAFFE database as our training set and the preprocessed video sequences from the MMI database as our testing set, since our eventual goal is to run this on real time video. As a comparison we also constructed 10 pairs of training and testing sets from the JAFFE database. In each pair there are 9 subjects in the training set and the tenth was used for cross validation. Splitting the data in this way for each subject gave us 10 cross validation sets. By tabulating the result for each cross validation set, we were able to assess and compare the performance of the ANN of the JAFFE and the MMI database.

Training and testing on the JAFFE database

We first present our results of training the ANN on the JAFFE database and testing it on the JAFFE database cross validation sets as was described above. We present these results first, because we anticipate that testing the ANN on a non-JAFFE images would potentially yield either misclassification of FEs or a strong indication of the goodness of a minimal feature point set that we used per FE. In fact, this baseline should yield expectedly *good* results. The following results were achieved by using ANN with one hidden layer containing 30 neurons. We trained 10 different networks and then averaged the results.

Table 1. True positive rate for ANN trained and tested on the JAFFE database.

TP	Angry	Distress	Fear	Happy	Sad	Surprise
58.43%	57.33%	42.00%	50.89%	74.17%	44.22%	90.00%

As shown in Table 1, the average, overall accuracy of the trained ANN was 58.43%, but ranges from a low of 42% for “distress” to as high of 90% for “surprise”. This average figure of merit (FOM) is disappointing when compared to higher figures reported by other studies. However, we emphasize the limited number of feature points extracted per FE and additionally, noise contained in the data. In fact, other studies when tested with FE images beyond its training database show a comparable true positive rate. Thus, in the task of classifying six different FEs, we conditionally accepted this figure of merit. In fact, this figure is consistent with the drop in accuracy of FE recognition by human subjects (Bassili 1978). Also the table shows accuracies for each particular FE. From above we can clearly see that our methodology is suited for classifying FEs with clear movement of feature points, i.e. “happy” and “surprise” relative to the neutral FE. We assert that these FEs are well-suited for the ANN to learn and classify. On the other hand, FEs such as “angry, distress, fear and sad” are “weakly” characterized by our descriptor and therefore not well-suited for the ANN to learn. Another possible source of the lower FOM is the inherent noise, that could mask any pattern in our training or testing data.

Training on the JAFFE database and testing on MMI database

It is impractical to assume that we will always have associated training dataset available for a FE which we want to analyze. Therefore to evaluate the ability of our ANN-based application to recognize FEs on unseen (external) faces, we first trained the ANN with the JAFFE database and then tested images taken from the MMI database. In fact, not only are the training/test sets different, they span ethnic and cultural characteristics. Thus, again using the same ANN, we averaged the results of 10 networks. Validation based early stopping was used while training the ANN, this method uses a sample validation set that is used to test the network after every epoch of training and stops further training upon increase in the error for the validation set, even if the maximum number of training epochs has not been reached. In this case we used a sample of the MMI database as the validation set so as to improve performance. This also prevents the ANN from overfitting, which is when the ANN begins to memorize the training set rather than identifying a trend.

Table 2. True positive rate for ANN trained on the JAFFE database and tested on the MMI database.

TP	Angry	Distress	Fear	Happy	Sad	Surprise
55.96%	57.38%	17.22%	13.05%	90.80%	50.24%	92.00%

Table 2 shows that the average accuracy achieved on a different database was 55.96%, which is comparable to the accuracy we achieved using the JAFFE database. This is also comparable to the nonlinear AdaSVM developed by Littlewort et al. (2006) who reported impressive results for FE recognition on one database; however, this FOM dropped to ~60% when tested on another database.

One can again see that “surprise(Su)” and “happy(Ha)” FEs consistently produce higher figures of merit and support the view that these FEs are easiest to train on and classify. Interestingly, the ANN is some 15% better at recognizing the happy expression in the MMI than in JAFFE on which it was trained. Though limited, we learned through our own FE database development that subjects from some ethnicities and cultures (here Asians) do not express a sense of “happy” with the same “intensity” as from other backgrounds. However, the vectoral descriptor appears to be consistent for both JAFFE and MMI. We can also see that “angry” and “sad” FEs are still relatively difficult for the ANN to classify; that is, the descriptor for these MMI FEs is not particularly distinct. Interestingly the true positive rates for FEs “distress” and “fear” exhibited a significant decrease relative to Table 1. In fact, in the case of “fear”, the rate is worse than that achieved by random guess. So overall the performance of the ANN trained on the JAFFE and tested on the MMI database appears to be consistent for Angry, Happy, Sad and Surprise expressions and (provisionally) suspect for Distress and Fear, relative to Table 1.

INTERPRETATION OF OUR RESULTS

We will be discussing a curious observation where there was a drop in FOM with respect to Distress and Fear expressions on the MMI database suggesting that indeed there may be some potential ethnic and cultural constraints (within a group) and equally, bias (beyond a group) when training and testing on different database.

Lower FOM on Distress and Fear FEs

We observed from Table 1 and Table 2 that the FOM resulting from the ANN were consistent except for the FEs, “distress” and “fear”. To investigate the potential source of these results, we generated a simple “confusion matrix”, Table 4, for ten networks of our ANN. Here the top *row* FE labels represent the classification FE and the leftmost column FE labels represent the true (known) classification. Thus the diagonal elements signify a (desired) one-to-one “correct” correspondence; the off-diagonal elements per column signify incidence of misclassification. The asterisk (*) is discussed below.

Table 4. The confusion matrix of ANN trained on the JAFFE database and tested on the MMI database.

true\classif.	Angry	Distress	Fear	Happy	Sad	Surprise
Angry	30	5	5	0	2	0
Distress	26*	6	1	0	2	1
Fear	0	4	2	1	8	31*
Happy	0	1	2	44	2	1
Sad	3	6	3	0	23	6
Surprise	0	0	0	0	3	47

For for all FEs “Angry, Distress, Fear, Happy, Sad and Surprise”, we see that one element (box) contains the largest number of incidents. In fact, “Angry,

Happy, Sad and Surprise” are correctly classified with conditions to be discussed. Further, by the low number of incidents on the diagonal, both “Distress and Fear” are inconclusively classified. That is, these two FEs learnt from JAFFE training set is weakly correlated to the same FEs from the MMI test set, and thus misclassified as other FEs, except surprise. At the least, “distress and fear” is not taken as “surprise” by the paired JAFFE-MMI datasets.

Next, we noted that a large number of incidents of “distress and fear (column)” test examples were misclassified as “angry and surprise(row)” respectively. Here, the network is simply not able to distinguish between “fear and surprise” and “angry and distress” FEs; that is, even though human experience may indicate a measure of confidence to the contrary (that one can indeed distinguish Fear, Surprise, Angry and Distress), from a image processing and ANN perspective, these FEs “look” similar and are paired. Moreover, there were not recorded incidents of “surprise” misclassified as “fear” and only a few cases of “angry” misclassified as “distress”. Based on this, the vectoral descriptor corresponding to “Distress and Fear” learned via ANN/JAFFE appears to be similar to the descriptor corresponding to the MMI FE for “Angry and Surprise” respectively.

Finally, from an application perspective we suggest the following ethnic and cultural inferences. As noted, the JAFFE database used here for ANN training consists of young to middle-aged (20-35 years old) Japanese female subjects. On the other hand, the MMI database for training, consists of diverse (gender, age) subjects but contains in majority, white (Caucasian, European) men and women. There are individual subjects (see Fig. 2) with different ethnic/cultural appearance. This leads us to the partial interesting conclusion that ethnic, cultural and gender differences between these two databases can lead to misclassification and expectedly, a lower FOM. These results may also suggest that, although six basic FEs are widely accepted as common across ethnicities and cultures, the subtleties associated with FEs that are either mixed FEs and/or lower in amplitude/“intensity” (not a distinct vectoral descriptor) can be misclassified when the training occurs on a ethnicity/culture-specific database and testing on another specific or non-specific database. Further, results from our work suggests that the FEs, “distress-fear” and “fear-surprise”, are prone to a paired misclassification when training occurs on JAFFE (Japanese) and testing on MMI (mostly Caucasian/European) FE datasets.

CONCLUSION

Human beings communicate by many means but in spoken conversation with one or more subjects, the voiced and facial expressions (FEs) are dominant modes in conveying information. In the present work we sought to process facial expressions images and classify them into one of six standard FEs; that is, “happy, sad, surprised, fear, distress and angry”. A seventh neutral expression was taken as a reference for both of the two FE databases we used in the study. To maintain minimal computational effort, we used a limited number of facial feature points (per FE image) and accepted a measure of noise as well.

In spite of these conditions and constraints, we showed that the ANN conditionally performs in a reasonable manner. That is, although the overall

figure of merit (FOM) of our method does not attain a high value reported by previous investigators, we note that these higher FOMs are achieved at the expense of off-line and computationally-intensive FE recognition, image processing and classification. From the present objective applications in the field, quasi-real time acquisition, processing and classification of lower quality facial images are anticipated. Our results to date indicate the following for consideration, mainly that:

- Training with an ethnicity/culture specific (JAFFE) FE database and testing with different database reveals some consistencies and differences among FEs.
- Generating vectoral descriptors for a given FE and tracking change in its amplitude, relative to a reference, neutral FE is a simple way of analyzing spatiotemporal changes in FEs.
- The “happy” and “surprise” FEs are consistently recognized and classified across these FE database via use of vectoral descriptors.
- The “angry-distress” and “fear-surprise” FE pairs are evidently linked by misclassification across different FE databases; that is, there is some evidence that these are ethnicity/culture specific.

ONGOING WORK

In our research we are working toward improving the performance of our algorithm. And also improve the performance of the ANN using a larger database of ethnically/culturally diverse facial images or video sequences. We are also investigating initial indications that differences in FEs (and thus emotions) between different ethnicities and cultures can indeed be revealed by the systematic misclassification as indicated by our “confusion matrix”. Finally, in order to reduce computational time while simultaneously improving FOM, we are developing a means to quasi-simultaneously process the voiced expression along with the FE since the voice also conveys a measure of the state of emotion (Ramamohan and Dandapat, 2006).

REFERENCES

- [1] Bassili, N. J., 1978, "Facial Motion in perception of faces and of Emotional Expressions", *Journal of Experimental Psychology : Human Perception and Performance*, Vol. 4, No. 3, pp. 373-379.
- [2] Chennamsetty, N. and Tokuhiko, A., 2004, "Facial Expression Classification Using Learning Vector Quantisation Networks", ANNIE 2004, Smart Engineering System Design, St. Louis, MO, USA, Nov. 7-10.
- [3] Chennamsetty, N., 2005, "Development of Automatic Facial Expression Recognition System Using Gabor Wavelets and Learning Vector Quantisation Networks", M.S. Thesis, University of Missouri-Rolla.
- [4] Cohen, I., Sebe, N., Garg, A., Chen, L. and Huang, T. S., 2003, "Facial expression recognition from video sequences: Temporal and static modeling.", *Computer Vision and Image Understanding*, Vol. 91, No. 1-2, pp. 160-187.
- [5] Ekman, P. and Friesen, W. V., 1978, "Facial Action Coding System Manual", Palo Alto: Consulting Psychologists Press.
- [6] Ekman, P., 1992, "Are there Basic Emotions?", *Psychological Review*, Vol. 99, No. 3, pp.550-553.

- [7] Fasel, B. and Luetten, J., 2003, "Automatic Facial Expression Analysis: A Survey", *Pattern Recognition*, Vol. 36, pp. 259--275.
- [8] Gunnar Johansson, 1973, "Visual Perception of Biological Motion and an order for its Analysis", *Perception and Psychophysics*, Vol. 14, No 2, pp. 201-211
- [9] Kobayashi, H. and Hara, F., 1992, "Recognition of Six Basic Facial Expressions and their Strength by Neural Network", Proceedings, International Workshop Robot and Human Communication, pp. 381-389.
- [10] Littlewort, G., Stewart, M., Fasel, I., Chenu, J. and Movellan, J., 2003, "Analysis of Machine Learning Methods for Real-Time Recognition of Facial Expression from Video", Technical Report, Machine Perception Laboratory, Institute for Neural Computation, University of California, San Diego.
- [11] Littlewort, G., Bartlett, M., Fasel, I., Susskind, J. and Movellan, J., 2006, "Dynamics of facial expression extracted automatically from video", *Image and Vision Computing*, Vol. 24, No. 6, pp. 615-625.
- [12] Lyons, M. J., Akamatsu, S., Kamachi, M. and Gyoba, J., 1998, "Coding Facial Expressions with Gabor Wavelets", Proceedings, Third IEEE International Conference on Automatic Face and Gesture Recognition, Nara Japan, IEEE Computer Society, pp. 200-205.
- [13] Mehrabian, A., 1968, "Communication without Words", *Psychology Today*, Vol. 2, No. 4, pp. 53-56.
- [14] Pantic, M. and Rothkrantz, L. J. M., 2003, "Automatic Analysis of Facial Expressions: The State of the Art", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 22, No. 12, pp. 1424-1455.
- [15] Pantic, M., Valstar, M. F., Rademaker, R. and Maat, L., 2005, "Web-based Database for Facial Expression Analysis", Proceedings IEEE International Conference on Multimedia and Expo, Amsterdam, The Netherlands.
- [16] Ramamohan, S. and Dandapat, S. 2006, "Sinusoidal Model-Based Analysis and Classification of Stressed Speech", *IEEE Transactions on Speech and Audio Processing*, Vol. 14, No. 3, pp. 737-746

Appendix D - Defense Presentation

2nd May, 2008

8:30 AM

Mechanical and Nuclear Engineering Conference Room

3056 Rathbone Hall

Kansas State University, Manhattan

Presented in partial fulfillment of the requirement for the degree
Master of Science in Mechanical Engineering
MNE Conference room 8:30 am, 2nd May 2008.

MULTI-MODAL EXPRESSION RECOGNITION

Under the guidance of
Dr Akira Tokuhiro

SRIVARDHAN CHANDRAPATI
Mechanical and Nuclear Engineering
Kansas State University, Manhattan.





Outline

- Background & Motivation
- Objectives
- Facial Expression Recognition
- Expression Recognition in Speech
- Multi-Modal Emotion Recognition
- Conclusions



Background & Motivation

Expressions are “observable verbal and nonverbal behaviors that communicate and/or symbolize emotional experience. Expression can occur with or without self-awareness. It is at least somewhat controllable, and it can involve varying degrees of deliberate intent.”

- (Kennedy-Moore & Watson, 1999)

- Applied Biometrics
- Human Computer Interaction (HCI)
- Social Robots



Human Computer Interaction

- Robots will soon be a part of everyday life
- Social skills are the most important
- Humans communicate more through expression in face and voice than through words





Objectives

- Develop a facial expression recognition system
- Develop a system for emotion recognition in voice
- Build a Multimodal Emotion Recognition system

Facial Expression Recognition



Facial Expression Recognition

- Databases
 - JAFFE
 - MMI
- Feature Extraction
 - Seeded Region Growing Technique
 - Particle Swarm Optimization Technique
 - Feature Mask
- Artificial Neural Networks
 - Introduction
 - Input Vector Generation
 - Methodology
- Results & Analysis



Japanese Female Facial Expression (JAFFE) Database

- 10 Japanese female models
- 7 facial expressions (6 basic + 1 neutral)
- 213 images in all

-(Lyons, 1998)





MMI database

- Subset of MMI used
- 15 subjects
- 6 basic facial expressions

-(Pantic, 2005)





MMI FE video



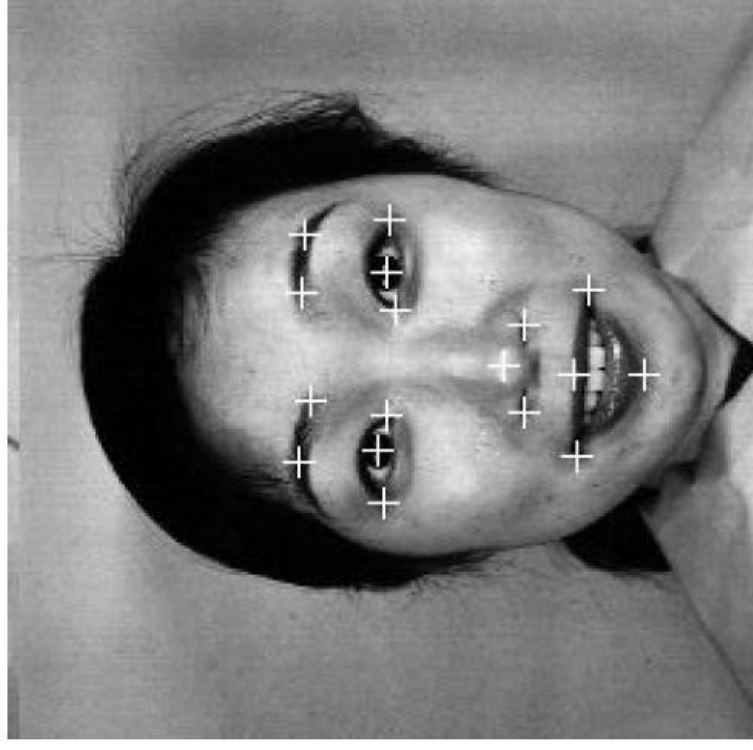
Srivardhan Chandrapati
Master of Science Defense

Multi-Modal Expression Recognition
2nd May 2008, 8:30AM



Feature Extraction

- Face Region Extraction: using Seeded Region Growing
- Feature Extraction: using Particle Swarm Optimization
- Generate a mask using the extracted points [44 X 1]

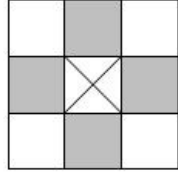
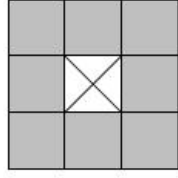


Face with located feature points that we select on the face

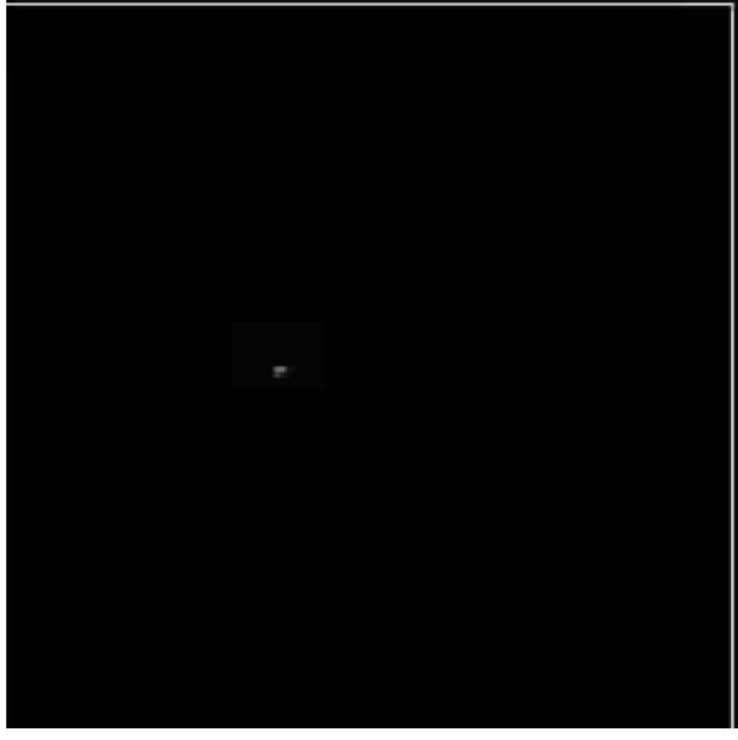


Seeded Region Growing

- Seed one/more pixels
- Compare the seed with its neighbors

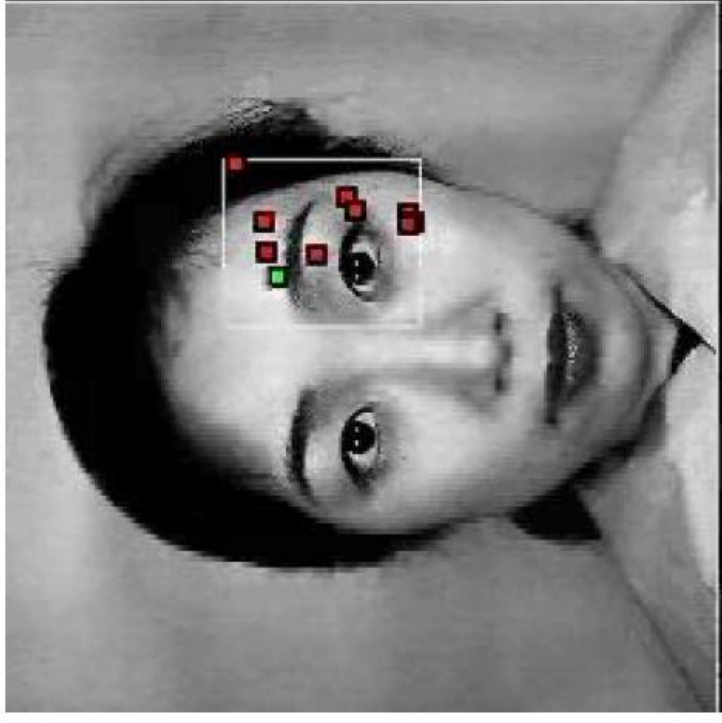


- Add them to the region if $|I_s - I| < T$
- Use newly added pixels as seed pixels



Particle Swarm Optimization

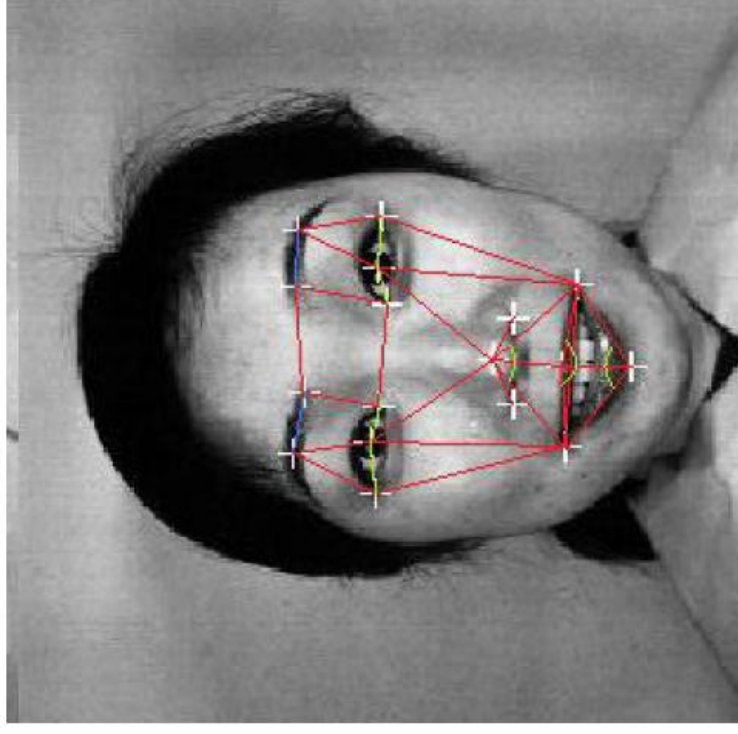
- Initialize particles randomly in the solution space
- Compute fitness of each particle
- Particles move towards
 - Best Global solution found until this point
 - Best solution for the given particle
 - Momentum of the particle
- Continue until particles converge





Feature Mask

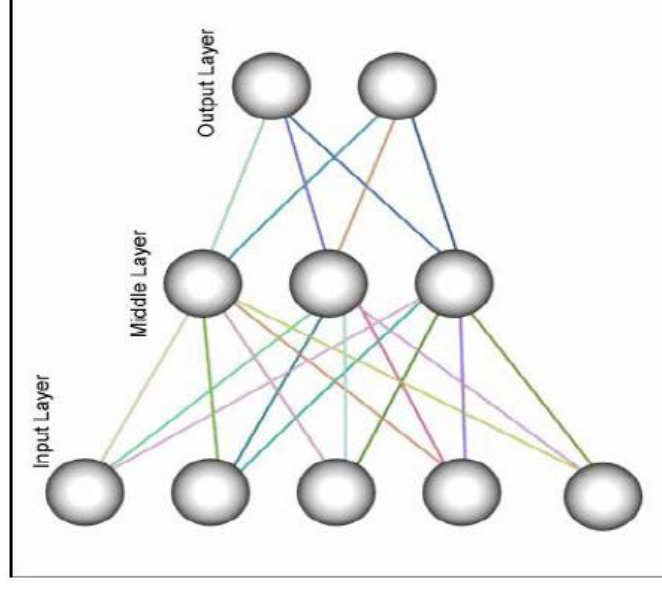
- A total of 38 features
- Distances between various feature points (25)
- Angles subtended by various points (5)
- Angles with the horizontal (2)
- Difference Vectors (6 x 2)
- [44 x 1] feature vector



Face with located feature points that we select on the face

Artificial Neural Networks (ANN)

- They are artificial networks inspired by the central nervous system
- They are easy to use for complex problems which are hard to define with equations
- They can learn from examples and generalize the acquired knowledge
- We use them for classification of expressions





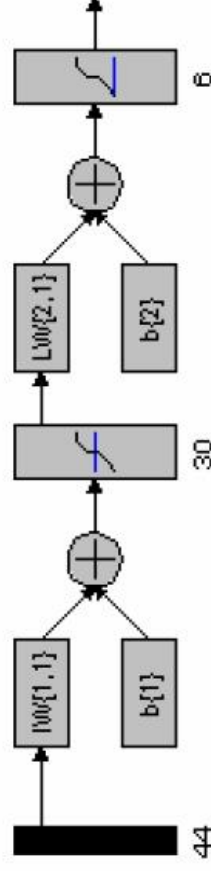
Methodology: Neural network

- Inputs to the ANN is the vector subtraction of the mask defining the present image and that of a pre-defined neutral image for the subject

[44 X 1 Input Vector] = [44 X 1 Neutral Mask] – [44 X 1 Expression Mask]

- JAFFE Database had on an average 3 neutral faces per subject. All images for a given subject were compared to each of the neutral images
- MIMI database consisted of videos starting with a neutral expression and gradually changed to the peak of expression and back to neutral.

- The network is a 3 layered feed-forward network with 44 nodes at input, 30 nodes in the hidden layer and 6 nodes at the output





Methodology: Neural network

- Training Method
 - Back Propagation

- Learning Rule
 - Gradient Descent with Momentum

- Training Database
 - JAFFE Database (9 subjects)

- Testing Database
 - JAFFE Database (1 Subject, Control Experiment)
 - MMI Database (15 Subjects)



Results : Control

True positive rate for ANN trained and tested on the JAFFE database.

True Positive	Anger	Disgust	Fear	Happy	Sad	Surprise
58.43%	57.33%	42.00%	50.89%	74.17%	44.22%	90.00%

- Testing was done on one subject from the database that was separated and not used for training purposes
- True positive is when an expression is correctly identified

Diagonal Element of Confusion Matrix
Sum of Corresponding Row

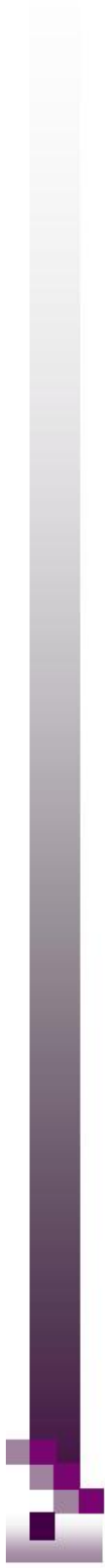


Results : Experiment

True positive rate when and tested on the MMI database.

True Positive	Anger	Disgust	Fear	Happy	Sad	Surprise
55.96%	57.38%	17.22%	13.05%	90.80%	50.24%	92.00%

- MMI database is culturally different from JAFFE since most of the subjects in MMI are young white people of European origin
- Significant differences from the control are the recognition rates for
 - Disgust ↓
 - Fear ↓
 - Happy ↑



Analysis

- The confusion matrix is a complete representation of results of the ANN
- This confusion matrix represents the results of the ANN when tested on MMI

true\classified	Anger	Disgust	Fear	Happy	Sad	Surprise
Anger	30	5	5	0	2	0
Disgust	26*	6	1	0	2	1
Fear	0	4	2	1	8	31*
Happy	0	1	2	44	2	1
Sad	3	6	3	0	23	6
Surprise	0	0	0	0	3	47



Analysis

- This confusion matrix represents the results of the ANN when trained on MMI and tested on JAFFE

true\classified	Anger	Disgust	Fear	Happy	Sad	Surprise
Anger	51	29*	1	3	6	0
Disgust	19	38	4	1	25	0
Fear	19	14	40	7	14	2
Happy	4	0	1	82	0	0
Sad	24	8	13	4	39	2
Surprise	1	3	27*	3	1	55

Expression Recognition in Speech

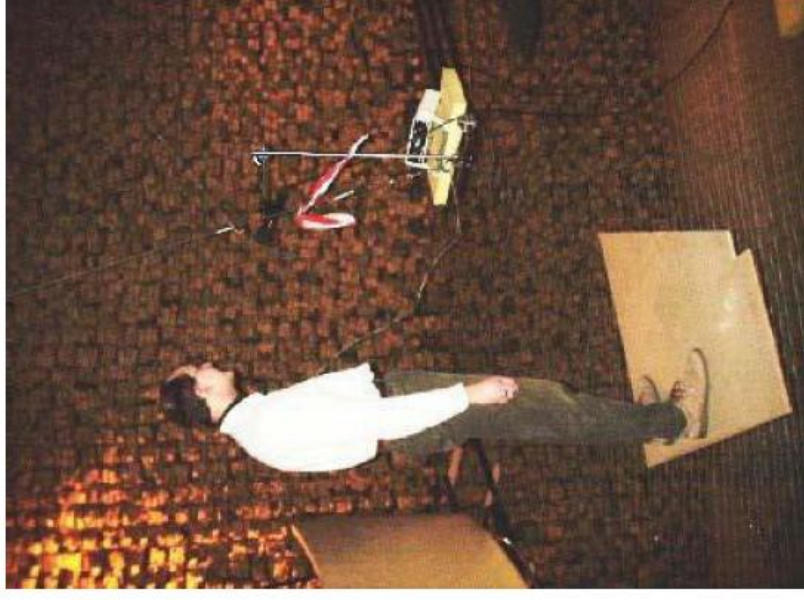


Expression Recognition in Speech

- Databases
 - Berlin Database of Emotional Speech
- Feature Extraction
 - Word Separation
 - Spectrogram
 - Formant Extraction
 - Human Speech Production Model
 - Linear Speech Model
 - Linear Predictive Coding
 - Mel Energy
 - Mel Scale
 - Mel Spectrum
 - Feature Vector
- Results & Analysis

Berlin Database of Emotional Speech

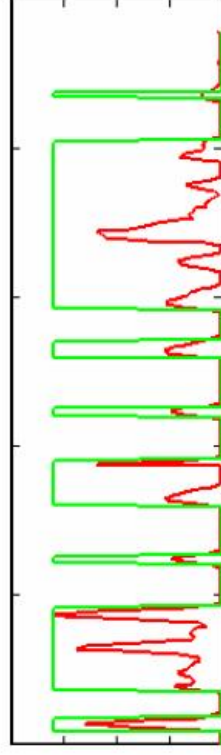
- 10 speakers (5 male, 5 female)
- 7 emotions (anger, boredom, disgust, fear, happy, sad and neutral)
- 10 sentences
- 535 recording in all





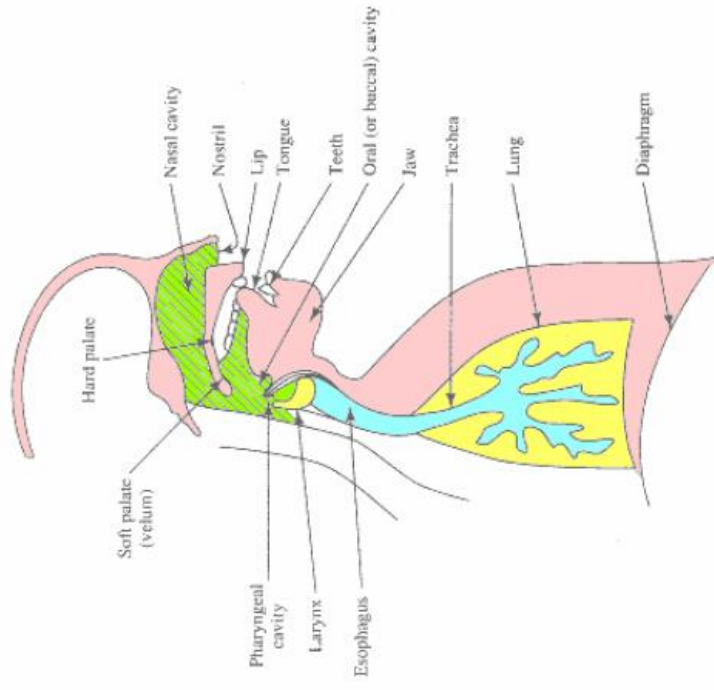
Word Separation

- Spectrogram is a plot of short term Fourier transform
- Power plot is the sum of energies at various frequencies
- Power below a threshold for longer than a set time is considered as a pause
- Pause is used as criterion for word separation



Human Speech Production System

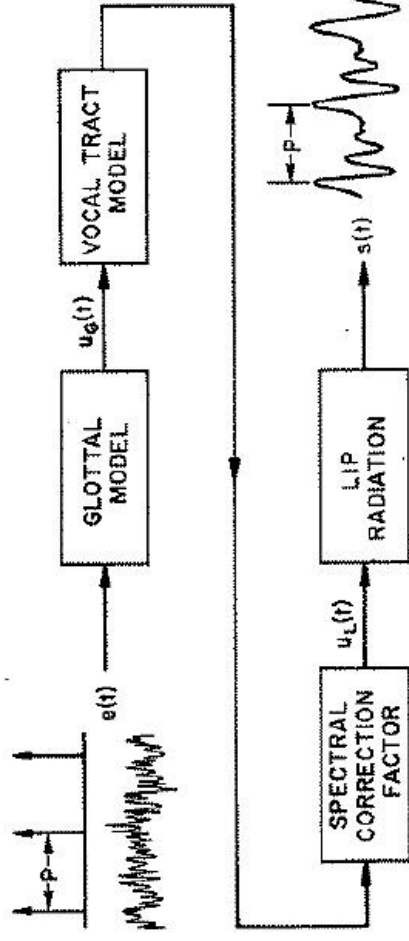
- Speech is produced when air flow is obstructed by vocal cords
- Vocal tract modulates the sound produced by the glottis
- Vocal tract is described in terms of its resonant frequencies or formants

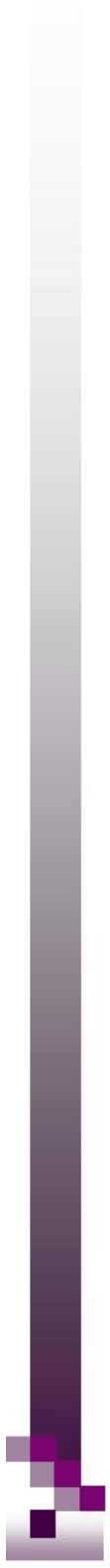


Linear Speech Production Model

- Source is a combination of white noise and regular impulses

- Output, $S(z) = E(z).G(z).V(z).L(z)$
 $\approx E(z).V(z)$





Linear Predictive Coding

- Predicting present time domain based on linear combination of previous samples
- LPC coefficients are computed using least squares method to reduce prediction error
- LPC coefficients can be used to define the filter

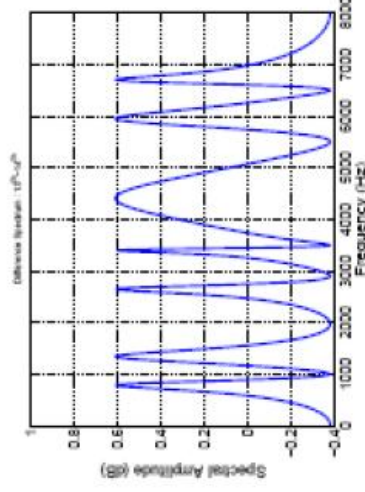
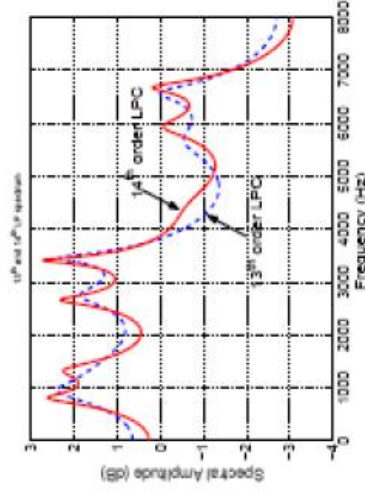
$$E(z) = S(z) \cdot A(z)$$



Formant Extraction

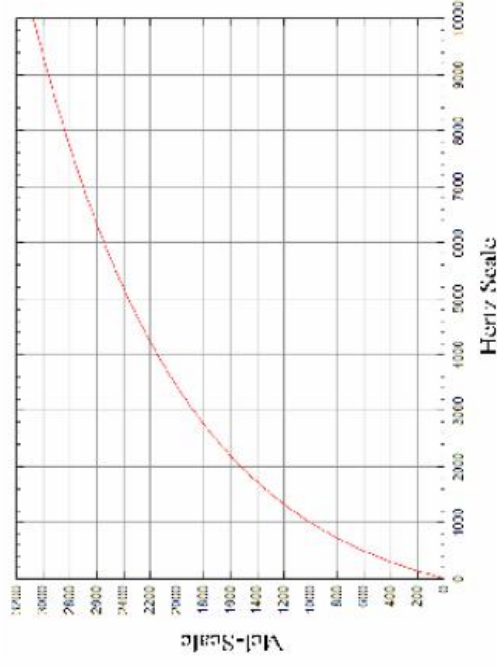
- Using LPC to model the vocal tract we get
- Formants can be extracted by peak picking on the gain plot
- The difference spectrum of two filters emphasizes the formant frequencies

$$V(z) = 1/A(z)$$



Mel Scale

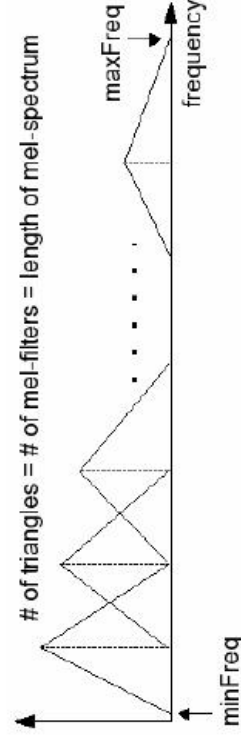
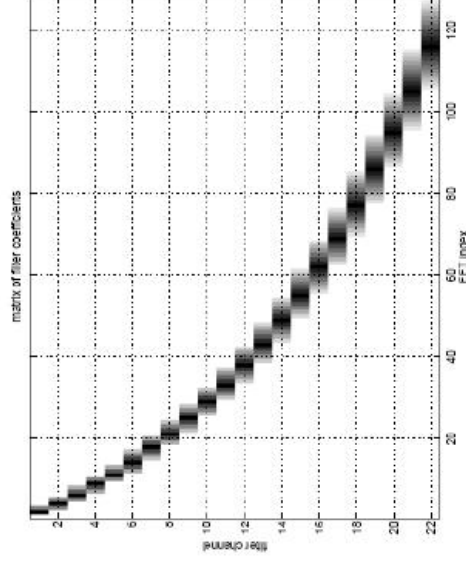
- Mel scale is perception based where listeners judge the scale such that the distance from one pitch to another is equal
- 1000Hz and 40dB over the listener's threshold is set as 1000mels



$$f_{mel}(f) = 2595 \log(1 + f/700\text{hz})$$

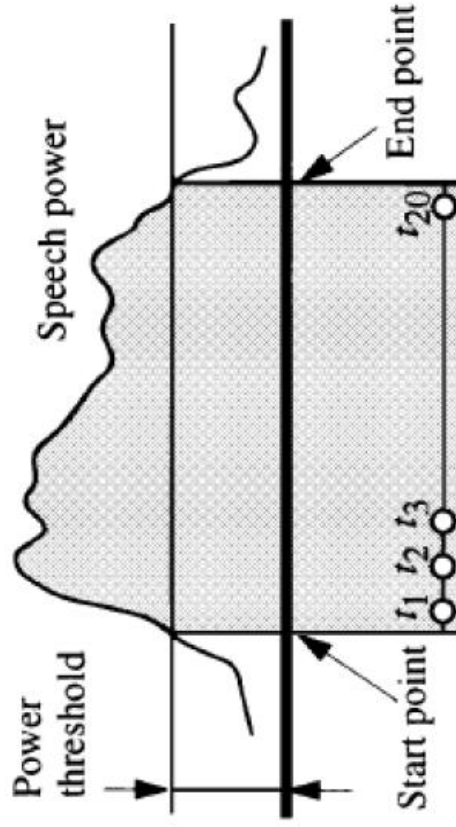
Mel Spectrum

- Is computed by multiplying spectrogram with the mel filter
- Mel filter is a collection of a number of triangular windows with increasing bandwidth as the center frequency increases
- Models the non-linear response of the human ear, the frequency resolution decreases with increasing frequency



Feature Vector

- Word is divided into five segments
- Features averaged over each segment
- Mean and medians of formants, mel-energy, rate of change of formants and mel-energy, word speed, ratio of voiced and unvoiced speech, time length of word and average energy

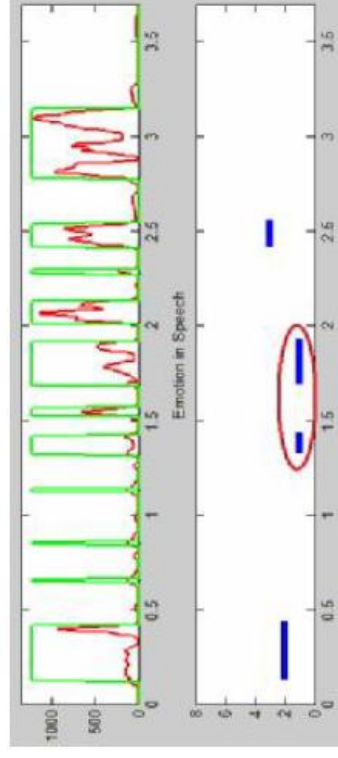




Results and Analysis

TP	An	Bo	Di	Fe	Ha	Sa	Ne
43.87	81.40	36.54	29.03	3.57	0.00	58.97	42.86

- Classification is for a complete word and not frame wise
- Happy and Fear are toughest to recognize
- Anger, Boredom, Disgust, Sad and Neutral are recognized with higher accuracy as compared to facial expressions



Multimodal Emotion Recognition



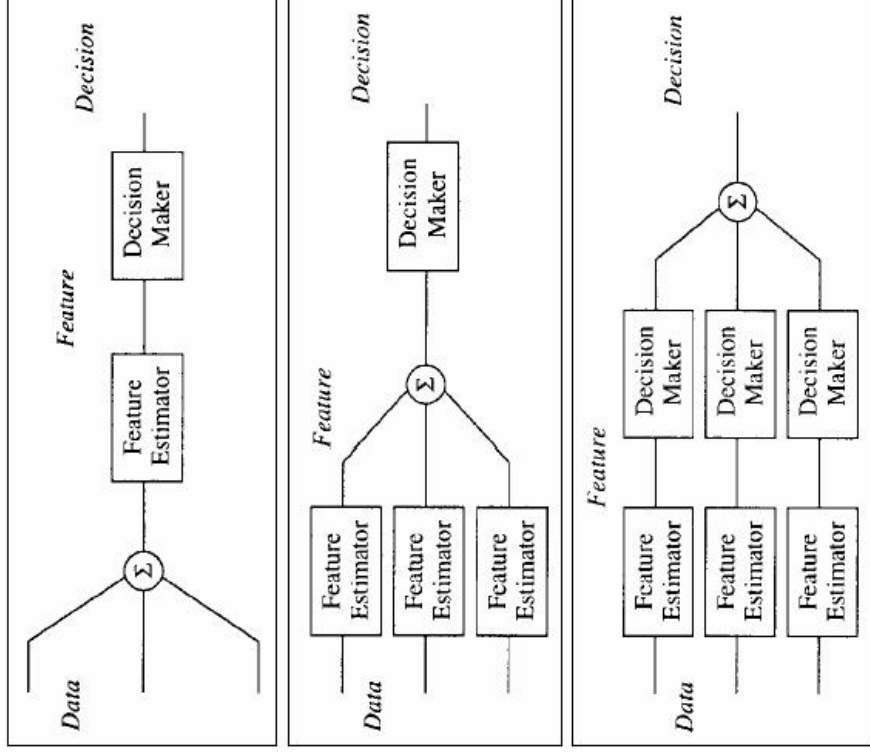
Multimodal Emotion Recognition

- Multimodal fusion
 - Signal level
 - Feature level
 - Decision level
- User Interface
 - File Selection
 - Audio Toolbox
 - Video Toolbox
 - Emotion Recognition
- Testing and Analysis



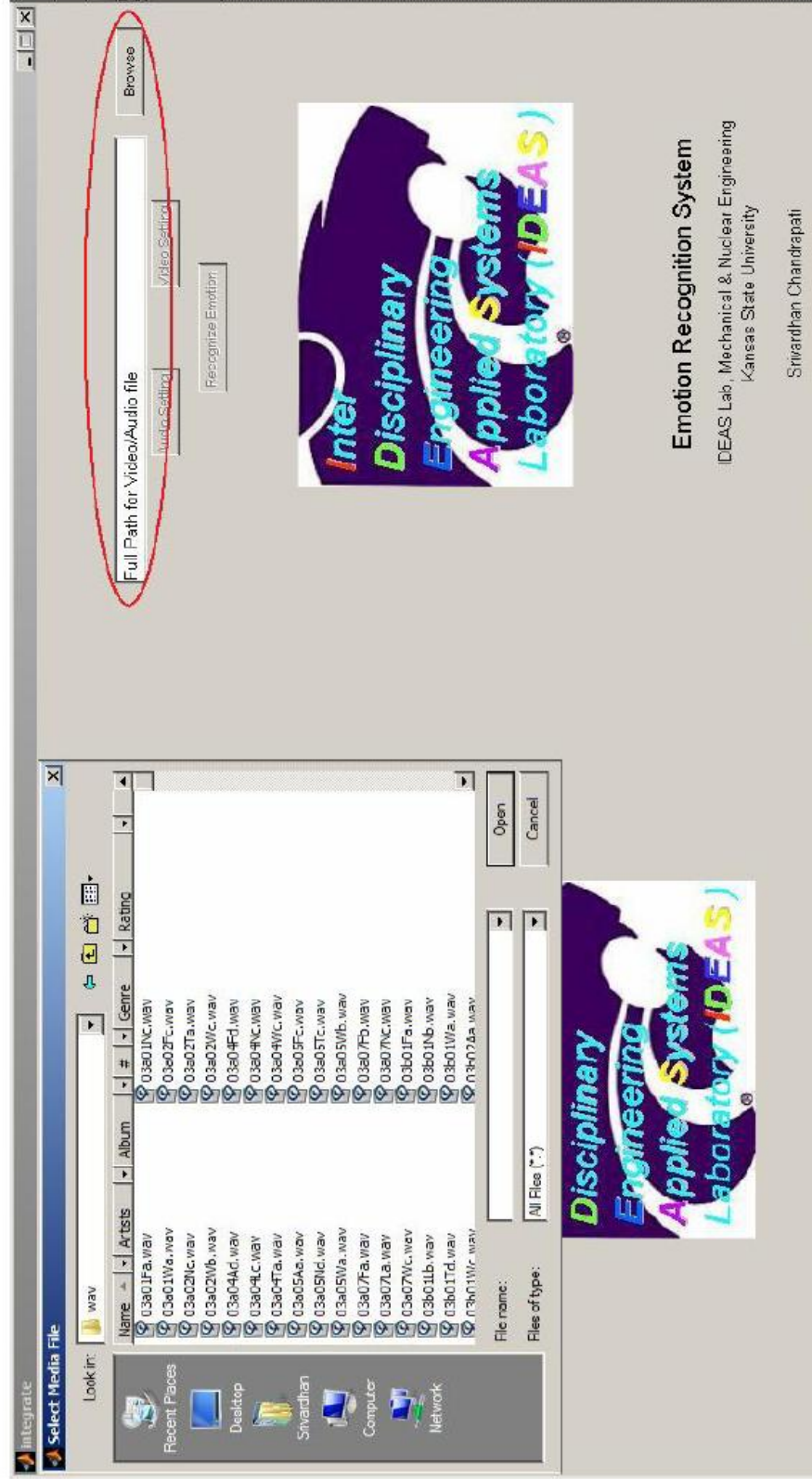
Multimodal Fusion

- **Signal level**
 - Similar kinds of signals are fused before feature extraction
- **Feature level**
 - Features from each modality are extracted separately
 - Used together in one input vector for classification
- **Decision level**
 - Each modality is classified individually
 - Final decision based upon the independent results



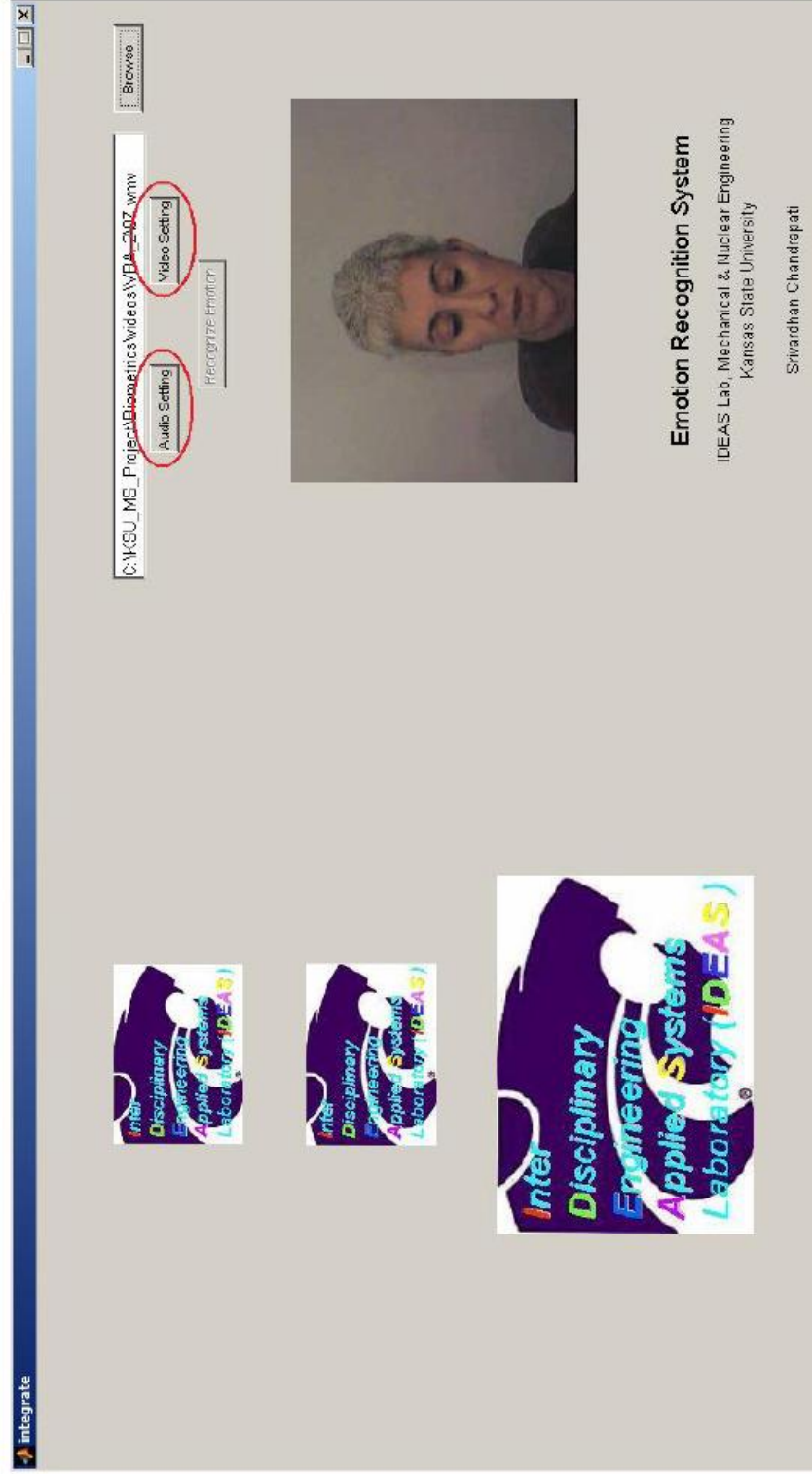


User Interface : File Selection



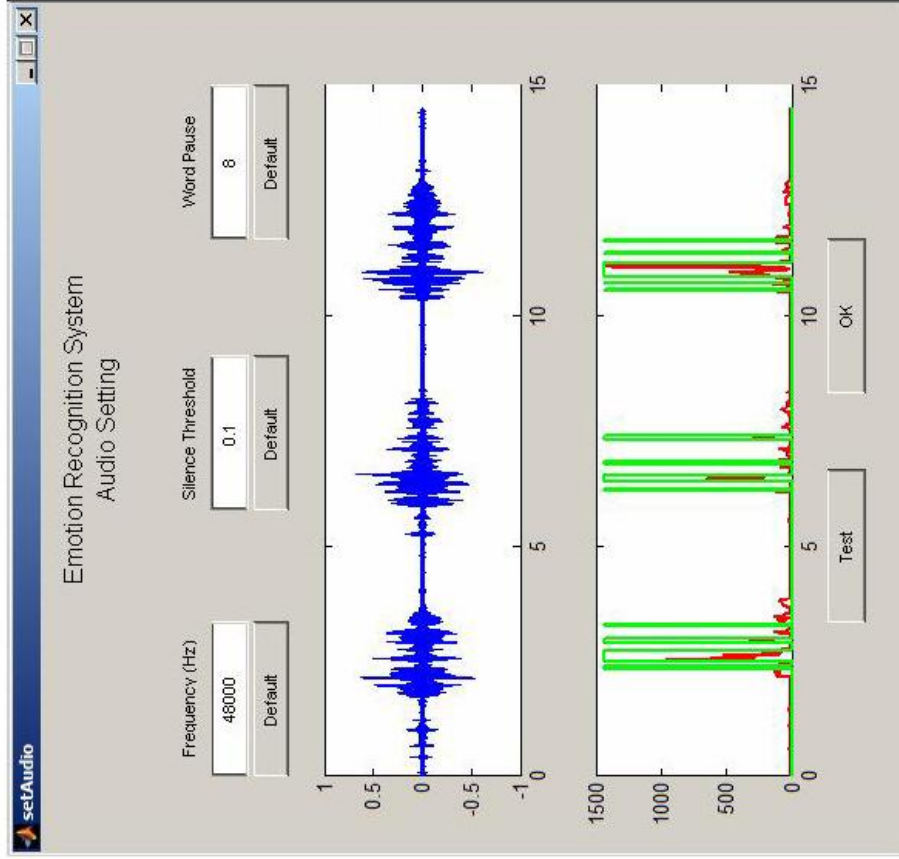


User Interface : Read File





User Interface : Audio Toolbox

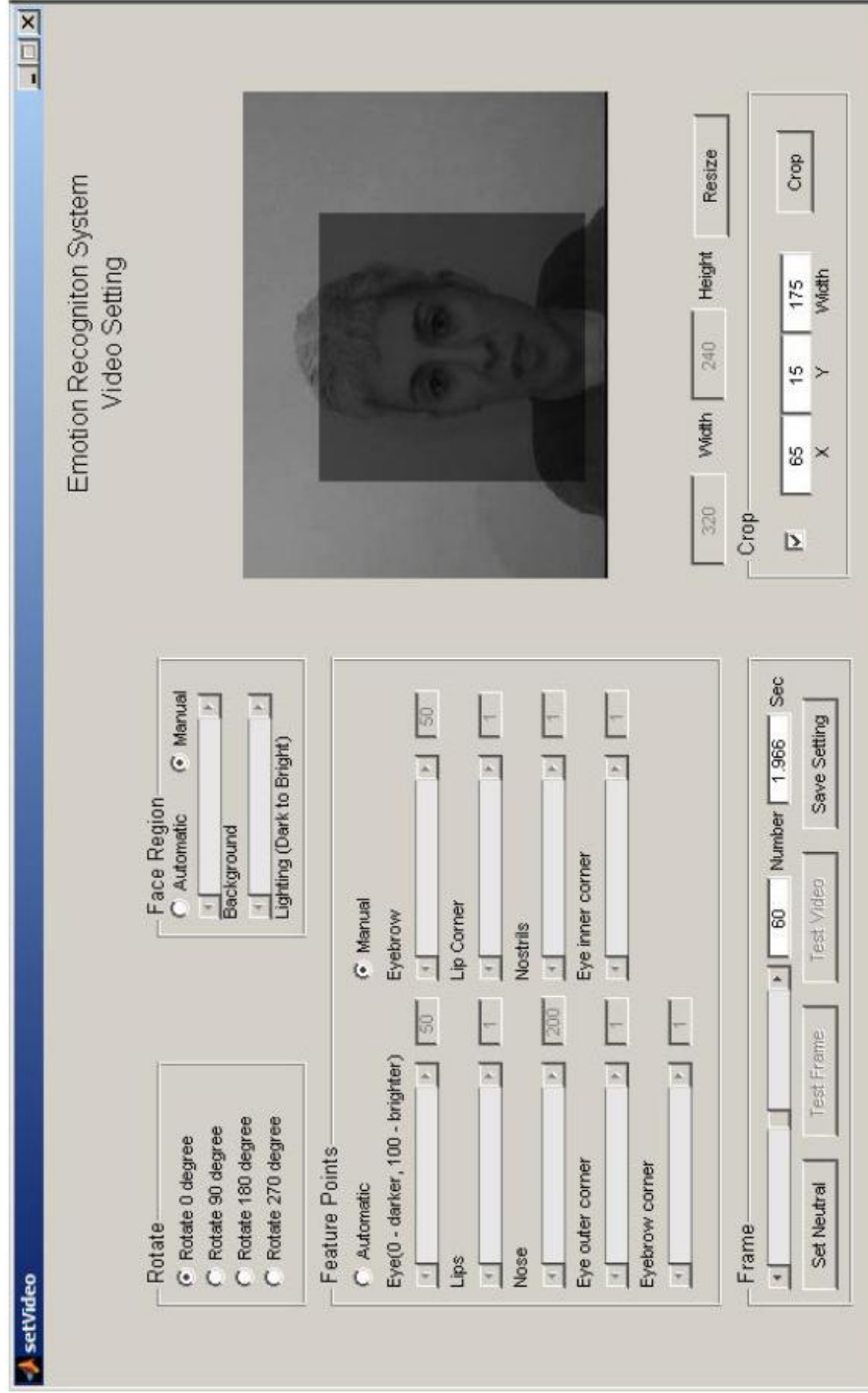


Srivardhan Chandrapati
Master of Science Defense

Multi-Modal Expression Recognition
2nd May 2008, 8:30AM

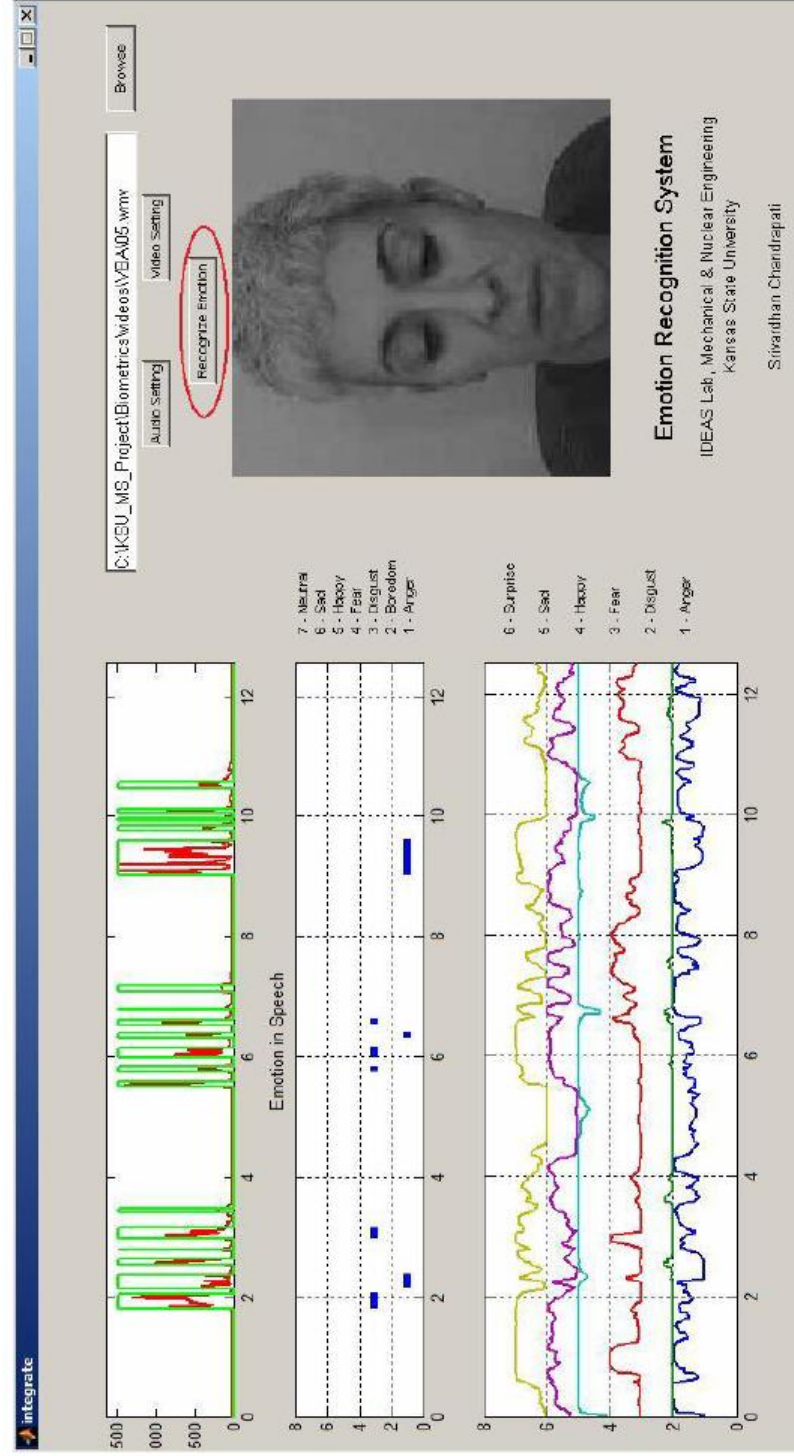


User Interface : Video Toolbox





User Interface : Emotion Recognition





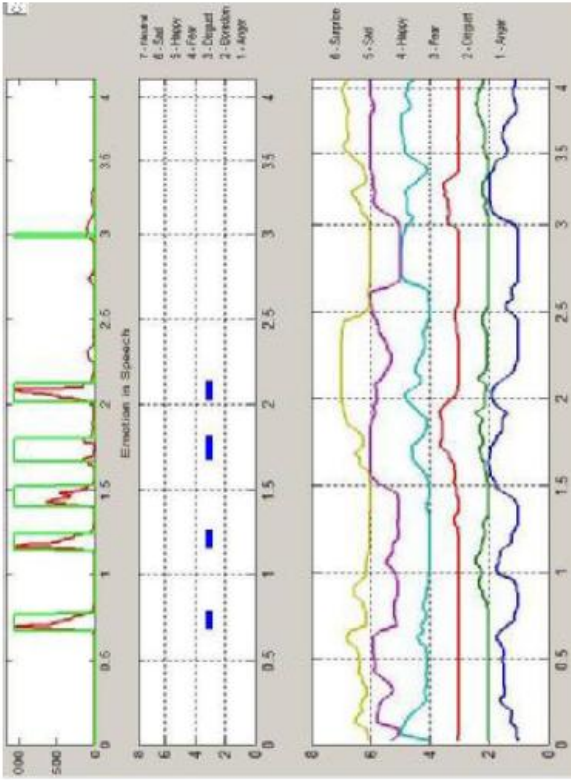
Testing

- We had a volunteer fluent in German who voiced the phrases with the noted emotions
- Most of the voiced phrases were classified as either anger, disgust or boredom
- We used the data from the angry and disgust expressions to study if the voiced emotions had any cues that could solve the anger-disgust expression confusion in the face
- Further testing was not possible for because of lack of a freely available multimodal database

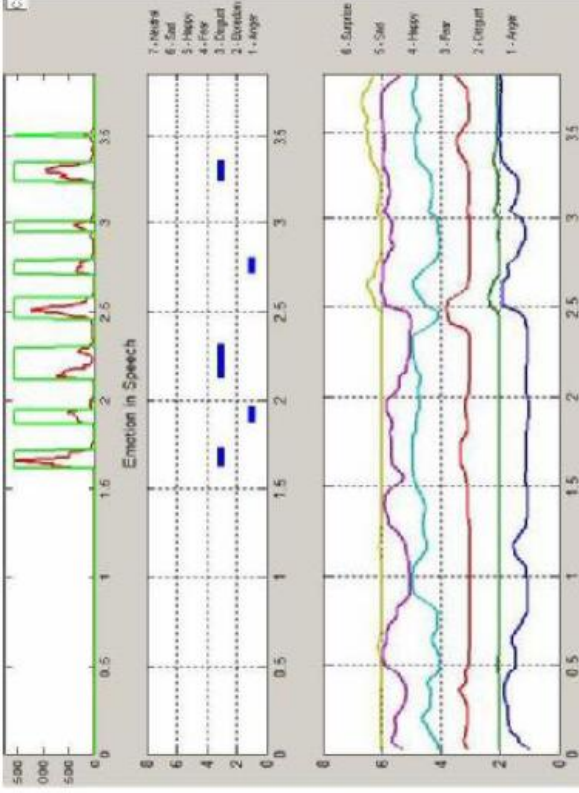


Analysis

Disgust



Anger





Conclusions : Major

- Happy and Surprise are the most accurately recognized emotions in the face, while distress and fear are the least accurately recognized.
- The “angry-disgust” and “fear-surprise” FE pairs are evidently linked by misclassification across different FE databases; that is, there is some evidence that these are ethnicity/culture specific.
- Anger, Disgust, Boredom and Sadness are more accurately recognized in the voice, while Happy and Fear are hard to recognize.
- Addition of voiced expression recognition can clear the confusion in expression recognition in face.



Conclusions : Minor

- Time dependent machine learning techniques are better suited for emotion recognition in speech.
- They allow for complete multimodal fusion.
- Standard emotionally labeled speech and multimodal databases are required for further research in the direction.



Questions

