FACE RECOGNITION USING HYBRID SIFT-SVM

DISSERTATION/THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE
OF

MASTER OF TECHNOLOGY
IN
CONTROL AND INSTRUMENTATION

Submitted by:

Kirti Bagla

Roll No. 2K14/C&I/05

Under the supervision of

Dr. Bharat Bhushan

DEPARTMENT OF ELECTRICAL ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042
2016

DEPARTMENT OF ELECTRICAL ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

CERTIFICATE

I, Kirti Bagla, Roll No. 2K14/C&I/05 student of M. Tech. (Control and Instrumentation), hereby declare that the dissertation/project titled "Face Recognition Using Hybrid SIFT-SVM" under the supervision of Dr. Bharat Bhushan of Electrical Engineering Department, Delhi Technological University in partial fulfillment of the requirement for the award of the degree of Master of Technology has not been submitted elsewhere for the award of any Degree.

Place: Delhi **KIRTI BAGLA**
Date: 18.07.2016

Dr. Bharat Bhushan
Assosiate Professor
EED, DTU

ACKNOWLEDGEMENT

I take this opportunity to express my sincere gratitude to all those who have been instrumental in the successful completion of this dissertation.

Dr. Bharat Bhushan, Assosiate Professor, Dept. Of Electrical Engineering, Delhi Technological University, my project guide, has guided me for the successful completion of this dissertation. It is worth mentioning that he always provided the necessary guidance and support. I sincerely thank him for his wholehearted guidance.

I am grateful for the help and cooperation of **Prof. Madhusudan Singh**, Head of the Department of Electrical Engineering, Delhi Technological University, for providing the necessary lab facilities and cooperation and I wish to thank all faculty members whoever helped to finish my project in all aspects.

I would also like to thank my beloved parents, who always give me strong inspirations, moral supports, and helpful suggestions. Without them, my study career would never have begun. It is only because of them, my life has always been full of abundant blessing. To all the named and many unnamed, my sincere thanks. Surely it is Almighty's grace to get things done fruitfully.

Kirti Bagla

2k14/C&I/05

ABSTRACT

Face recognition provides a challenging issue in the domain of analyzing images. In this dissertation, a face recognition model using hybrid SIFT-SVM is presented and a comparative analysis between SVM and hybrid SIFT-SVM has been studied.

The current database is divided into two various parts, training and testing database. The SIFT feature will be created for each training images and the key points are computed, then the SVM is applied for the matching process for test images. Results are obtained for three cases child, adult and old age which are made on the basis of age. The recognition rate has been computed by False Acceptance Rate (FAR) and False Rejection Rate (FRR) on these cases and then the results of hybrid SIFT-SVM is compared with SVM. It has been studied that the recognized result provides robust performance under various conditions like different pose, lighting conditions and facial expressions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

GENERAL

Initially, Woody Bledsoe, along with Helen Chan and Charles Bisson during 1964 and 1965 worked to recognize human faces by using computers. He was proud of what they have done, but an unnamed intelligence agency provided the funding and it did not allow much publicity, hence, little or no work was published. When the database of images is large, to select a small set of records in which one of them would match the photograph was a problem. The ratio of the answer list to the number of records in the database is used to measure the success. Bledsoe (1966) described that the recognition is variant to head rotation, lightening conditions, facial expressions, aging etc. Chances of failure in pattern matching are there when the variations are large because the correlation is very low between the two pictures of same person. They labelled the project as man-machine because the coordinates of a set of features from the photographs were extracted manually, after that these features were used by computer for recognition. To extract the coordinates of features such as peak points etc a graphics tablet was used by the operator. Using these coordinates, a list of 20 distances, such as width of mouth and width of eyes, pupil to pupil, were measured. In the database, the list of computed distances and the photograph was associated with the person's name and it is stored in the computer. In the recognition phase, the set of distances was compared with the corresponding distance for each photograph, yielding a distance between the photograph and the database record. The closest records are returned.

Because it is unlikely that any two pictures would match if there are variations subjected to rotation, lean, tilt, and scale (distance from the camera), every set of distances is normalized to represent the face in a frontal orientation. To get this normalization, the program first tries to determine the extent of tilt, the lean, and the rotation. Then, it uses these angles and undoes the effect of these transformations on the computed distances. To compute these angles, the computer should know the 3D geometry of the head. Because of the unavailability of actual heads, a standard head which was derived from measurements on seven heads was used by Bledsoe.

After Bledsoe left PRI in 1966, this work was continued primarily by Peter Hart at the Stanford Research Institute. The experiments were performed on a database of over 2000 photographs, and the computer consistently worked better than humans when presented with the same recognition tasks.

A system was developed by Christoph von der Malsburg and graduate students of the University of Bochum in 1997. This Bochum system was funded by the United States Army Research Laboratory. It was sold and used for the first time by customers such as Deutsche Bank and operators of airports and other busy locations.

During 2007, an idea came up to build a database by asking users to input the names of people to recognize in photographs online. A company out of Minnesota, Identix developed the software, FaceIt. This software was capable to pick a face from the crowd and it could compare it to the worldwide database for recogniton and hence to put a name on that human face. It was written to detect multiple features on the human faces. It was capable of detecting the distances between the eyes, width of the nose, shape of cheekbones, length of jawlines and many more facial features. By putting the image of the face on a faceprint, the software was able to represent the human face through a numerical code. Facial recognition software used to work on a 2D image of the person when he was almost facing the camera directly. But with FaceIt, a 3D image can be compared to a 2D image by converting a 3D image into a 2D image with the use of a special algorithm that can be scanned through almost all databases. Latest face recognition algorithms' performances were evaluated in the Face Recognition Grand Challenge (FRGC). Results showed that the new algorithms had 10 times more accuracy than the face recognition algorithms of 2002 and 100 times more accuracy than those of 1995. Some algorithms were able to perform better than human participants in recognizing faces.

In Moore's law terms, in every two years, the error was decreased by one-half. Further insufficient resolution issues have been resolved in the last few years because of the improvements in high resolution, megapixel cameras.

1.2 FACE RECOGNITION

In face recognition, Software matches the face with its database. Advantage of this is that We do not require the consent of the person. Otherwise in case of fingerprint, iris scans and sound recognition we need the consent of the concerned person. But just like any other technique it also has many flaws in it. It is not perfect and under certain conditions it might give unsatisfactory results. Face recognition gives pretty good results at full frontal faces and 20 degrees off, but as you go towards profile, there are problems. Other conditions where face recognition does not work well include poor lighting, sunglasses, long hair or other objects partially covering the subject's face, and low resolution images. Another serious disadvantage is that many systems give varying results when facial expressions change. Even a big smile can affect the system. For example Canada is allowing only neutral facial expressions in passport size photographs because it will help in getting more reliable results. Advantage of using hybrid SIFT-SVM is that it is invariant to these conditions.

1.3 MOTIVATION

Face recognition has become a hot topic in recent decades as it doesn't require the consent of the person for its operation but it has any other problems. The motivation is to present a model which provides better results than earlier ones. Idea is to merge the two techniques to get better results.

1.4 PROBLEM FORMULATION

Based on the motivation and utility of the problem, the following objectives have been targeted in this thesis:

i) Study of SIFT algorithm and extraction of features of various test images using this.

ii) Study of SVM and classification of test images.

iii) Merge the above two techniques to make hybrid SIFT-SVM model.

iv) Comparison of the results obtained from SVM as well as Hybrid SIFT-SVM.

1.5 DISSECTION OF THESIS

The whole work is divided into 9 chapters.

Chapter 1 presents the basic introduction of face recognition, its advantages and disadvantages and objective of the thesis.

Chapter 2 contains the literature review on various face recognition techniques.

Chapter 3 explains the face detection.

Chapter 4 explains feature extraction technique i.e. the scale invariant feature transform.

Chapter 5 explains the classification and pattern recognition tool i.e. support vector machine and explains the hybrid SIFT-SVM model.

Chapter 6 gives tools and technology used.

Chapter 7 contains results and discussions.

Chapter 8 gives the comparison of results obtained from SVM and Hybrid SIFT-SVM.

Chapter 9 gives the conclusion and future scope of work.

Appendix can be referred for coding.

1.6 CONCLUSION

In this chapter, a brief introduction has been presented. Also, the motivation of the work and objective of the thesis has been presented. Further, the overview of each chapter is given.

CHAPTER 2
LITERATURE REVIEW

GENERAL

For humans, Face recognition is day to day job. People do not even notice how many times they do this in a day. Although, research has been going on in this field from last few decades but recently it has caught attention and has become one of the hot topics [10, 51]. So, it has undergone noticeable development in past few years [48]. D. Zhang has given advanced pattern recognition technologies with application to biometrics [14]. Locality preserving projection (LPP) is a manifold learning method widely used in pattern recognition and computer vision. Three novel approaches has been proposed by Y. Xu [38]. Y. Xu also proposed a framework to perform multibiometrics by comprehensively combining the left and right palmprint images [34]. A supervised sparse representation method for face recognition is discussed by Y. Xu [46]. M. A. Akhloufi introduced non linear dimensionality reduction approaches for multispectral face recognition [17]. Based on a sparse representation computed by ‘1-minimization, M. K. Hsu [47] proposed a general classification algorithm for (image-based) object recognition. H. F. Wang provided an up-to-date survey of video-based face recognition research [35]. Current 3D face recognition approaches are too slow for person identification, but D. Colbry reported several experiments that extract a sparse feature representation from the canonical 3D face surface and then performed recognition of a probe face based on the sparse features [15]. There was an expected trade off between algorithm speed and recognition performance. W. Feng [4] research showed that cross-talk noise is significantly reduced with wavelet filtering preprocessing. D. Y. Huang gave face detection model based on skin color and Adaboost [37]. K. Vaishanavi [40] proposed a face recognition technique using beck propogation neural network. Face recognition has various applications in today’s world. It is being used for face recognition of criminals by various security agencies around the world. Unique characteristics are measured by facial recognition and matched for identification and authentication. Facial recognition software detects a face [16], often through connected digital camera, extract its features and match them against stored database images [27, 56, 57]. The basic objective of Facial recognition software is to automatically identify individuals from digital images. Facial recognition software uses algorithms [53] that extract specific facial features, such as the relative distance between a person's nose, eyes, jaw and cheekbones. Unlike voice recognition & fingerprint recognition, facial recognition software yields nearly instant results because in facial recognition image is used and hence consent is not required. Facial recognition software can be used as a security measure and for verifying activities [33] of personnel, such as attendance, computer access, etc.

Many face analysis and modelling techniques have been proposed in the last decade but the reliability of such schemes is a great challenge to the scientific community. Some algorithms for face recognition such as PCA (Principal Component analysis), ICA [12] (Independent Component analysis), LDA (Linear Discriminant analysis), Fisher [38], Eigen face [1], SIFT are there [58]. Out of these SIFT [7, 52, 55] is most common because PCA, ICA, LDA are face based technique (Global features extraction) whereas rest of the two are feature extraction based (Local features extraction [6]). Y. Xu proposes matrix-based complex PCA (MCPCA), a feature level fusion method for bimodal biometrics that uses a complex matrix to denote two biometric traits from one subject. C. Y. Chang [36] proposed a modified LDA (called block LDA) to divide the gradient image into several non-overlapping subimages of the same size, in order to increase the quantity of samples and reduce the dimensions of the sample space. In Faced based technique whole face is considered as a feature so it provides good efficiency but the problem is that it is not robust to pose and expression changes. To minimise these above mentioned problems face image is divided into smaller blocks and then global feature extraction algorithms are applied but the problem still persists. This problem is solved by SIFT [11, 45]. SIFT simply transforms data present in the face image into the keypoint descriptor which is going to be used as a local feature of that particular image. Feature detection algorithms repeatedly detect the same point of interest in each image, regardless of the scale of the image and orientation of the subject and match these points of one image with corresponding point in another image. Luo et al. [13] has shown the ability of these features by combining a person's feautures and a matching stratergy. There has been lots of improvements in SIFT features [54]. Mikolajcyk ans Schmid said that in image deformation cases, SIFT is most resistant [9]. Acomparision is also made with other techniques and it was seen that SIFT is a powerful matching tool [21]. A test has also been made by taking all theinitial keypoints as features, which performed well on ORL and AR face databases [22]. Sift features are also used on vedio based techniques [41]. Although, SIFT uses local feature extraction but it concentrates too much on local extraction and the overall information of an image is ignored.

The linear representation methods [23, 29, 39] in which entire image is used for representation, has been realised as a powerful tool for classification. Subspace clustering refers to the task of finding a multi-subspace representation that best fits a collection of points taken from a high-dimensional space. E. Elhamifar and R. Vidal introduced an algorithm inspired by sparse subspace clustering (SSC) to cluster noisy data, and develops some novel theory demonstrating its correctness [24]. J. Wright considered the problem of automatically recognizing human faces from frontal views with varying expression and illumination, as well as occlusion and disguise. He casted the recognition problem as one of classifying among multiple linear regression models and argue that new theory from sparse signal representation offers the key to addressing this problem. Based on a sparse representation computed by minimization, he proposed a general classification algorithm for (image-based) object recognition [25]. Y. Xu, D. Zhang, and J. Y. Yang experiments show that the proposed two-step feature extraction scheme can achieve a higher classification accuracy than the 2DPCA and PCA techniques [30]. A label consistent K-SVD (LC-KSVD) algorithm to learn a discriminative dictionary for sparse coding is presented by J. Wright [31]. Two step test method, proposed by Y. Xu is able to reduce the side-effect of the other training samples that are very "far" from the test sample on the recognition decision of the test sample, the high recognition rates can be obtained [32]. Y. Xu, D. Zhang, J. Yang, and J. Y. Yang proposed a two-phase test sample 2 representation method for face recognition [42]. Y. Xu, Z. Fan, and Q. Zhu proposed to exploit the symmetry of the face to generate new samples and devise a representation based method to perform face recognition [43]. Y. Xu, and Q. Zhu proposed a very simple and fast face recognition method and present its potential rationale [49]. Along with SIFT, SVM is used in dissertation, which will classify the extracted feature and that provides better results. The idea is to merge the two techniques to get better.

2.2 SIFT AND SVM

Scale Invariant Feature Transform (SIFT) was given by David Lowe (2004) which is an image descriptor and it is used for image based matching and recognition. These descriptors are used in computer vision for various purposes which are related to object recognition. Under real conditions, SIFT descriptor is very useful for image matching and object recognition and this is proved experimentally. These features are invariant to scale, light variations, orientation. SIFT algorithm [27, 50] provides a set of features of an object which are not affected by clutter, occlusion and unwanted noise in the image. Also, these features are very distinctive in nature which helps in accomplishing correct matching on various pair of feature points with high probability between a large database and a test sample [44, 52]. Hybrid PCA-SIFT algorithm is proposed by Y. Ke [8]. Volume-SIFT (VSIFT) and Partial-Descriptor-SIFT (PDSIFT) for face recognition based on the original SIFT algorithm is proposed by C. Geng [19]. A. Majumdar gave discriminative SIFT features [20]. The keypoint detector represents the main source of errors in face recognition systems relying on SIFT features, to overcome the presented shortcoming of SIFT-based methods, J. Krizaj [28] presented a technique that computes the SIFT descriptors at predefined (fixed) locations learned during the training stage.

Support vector machine (SVM) is a supervised machine learning method which can be used for classification and pattern recognition. This algorithm is basically used for binary classification but it can be used for multi-class classification by using different methods. Support Vector Machine is based on the concept of decision plane that defines decision boundaries [2]. Firstly, the training data is mapped into a higher dimensional space which gives a hyperplane which separates one class of objects from another. If this hyperplane is mapped back into original dimensional space it may give a non linear classifier. To construct an optimal hyperplane, SVM performs an iterative training algorithm, which minimizes the error function. The maximum margin hyperplane separating the two classes need to be found, that is why it is known as maximum margin classifier [3, 5, 17].

2.3 CONCLUSION

This chapter presents the literature review of the face recognition techniques. Latest advancements and research in the field of image processing are discussed. It helps in enhancing the knowledge of the system and provides guidance in the thesis work.

CHAPTER 3

FACE DETECTION

GENERAL

In the presented model, Face detection has been done by using Viola and Jones algorithm. This algorithm is the first algorithm which was proposed for detection of an object by Viola and Michael Jones in 2001. It can be used to detect variety of object classes but main concentration was on the problem of face detection.

The Viola/Jones Face Detector features are as follows:

1) It can be used for real time object detection.

2) Though, it is time consuming to train but it detection is fast.

This algorithm has three main steps that is Integral image, Boosting and Cascade.

The basic benefits of this algorithm are robustness, real time usage and easy to implement and understand by sing computer vision toolbox in matlab. It is fast and efficient and invariant to scale and location. It scales the features, not the image.

3.2 ALGORITHM FOR FACE DETECTION

3.2.1 HAAR FEATURE SELECTION

Human faces have almost similar properties. Haar features are used to match these regularities which are shown below:

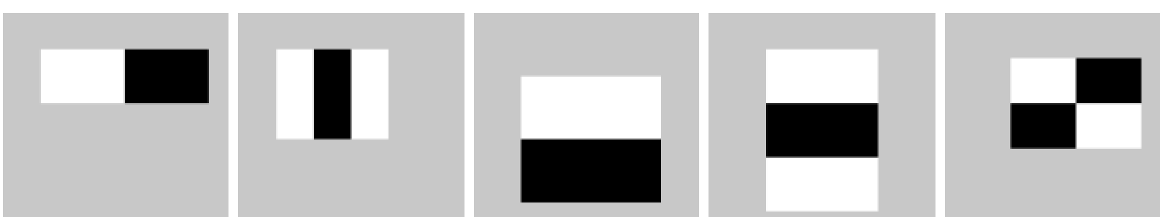


Fig. 3.1 Haar features

These are the five haar features (shown in fig. 3.1) which can be used for detectiong nose, eyes, mouth etc. The net value is sum of the pixels in white area minus the sum of the pixels in black area. Now, summing the pixels and subtracting them can be quite tedius. So, to make the job easier we go for integral image concept.

3.2.2 CREATING AN INTEGRAL IMAGE

Integral image calculation gives speed advantage over other alternatives. With the help of this, the net value can be calculated by just 3 additions. Interal image at any location (x,y) is sum of pixels to its above and to its left and including (x,y). Integral image evaluation is invariant to time also. One of the examples is shown below.

Original Integral Original Integral

5	2	3	4	1
1	5	4	2	3
2	2	1	3	4
3	5	6	4	5
4	1	3	2	6

5	7	10	14	15
6	13	20	26	30
8	17	25	34	42
11	25	39	52	65
15	30	47	62	81

5	2	3	4	1
1	5	4	2	3
2	2	1	3	4
3	5	6	4	5
4	1	3	2	6

5	7	10	14	15
6	13	20	26	30
8	17	25	34	42
11	25	39	52	65
15	30	47	62	81

5+2+3+1+5+4=20 5+4+2+2+1+3=17 34-14-8+5=17

Fig.3.2 Integral image

Fig. 3.2 describes conversion technique from original image into integral image and vice versa with an example.

3.2.3 ADABOOST TRAINING

Adaptive Boosting was given by Yoav Freund and Robert Schapire who got Godel Prize in 2003 for this work. To improve the performance of various other learning algorithms, adaboost was merged with them. The output of learning algorithms also known as weak learners is added to form a strong and final output of the boosted classifier. Adaboost algorithm gives importance to the ones which are misclassified by previous classifiers. This algorithm is affected by noisy data. Individually, the output is weak but as long as the output of weak learners is better than the random guessing, the final model can give a strong learner.

This is also referred as best out of the box classifier. Th below figures are given for better understanding. There are two classes one with plus and other with minus sign. One weak learner shown beside tries to separate the two classes. Some are classified correctly and some are misclassified. Now, more weight or importance is given to the ones which are misclassified as shown next. Again the same is repeated and process continues.

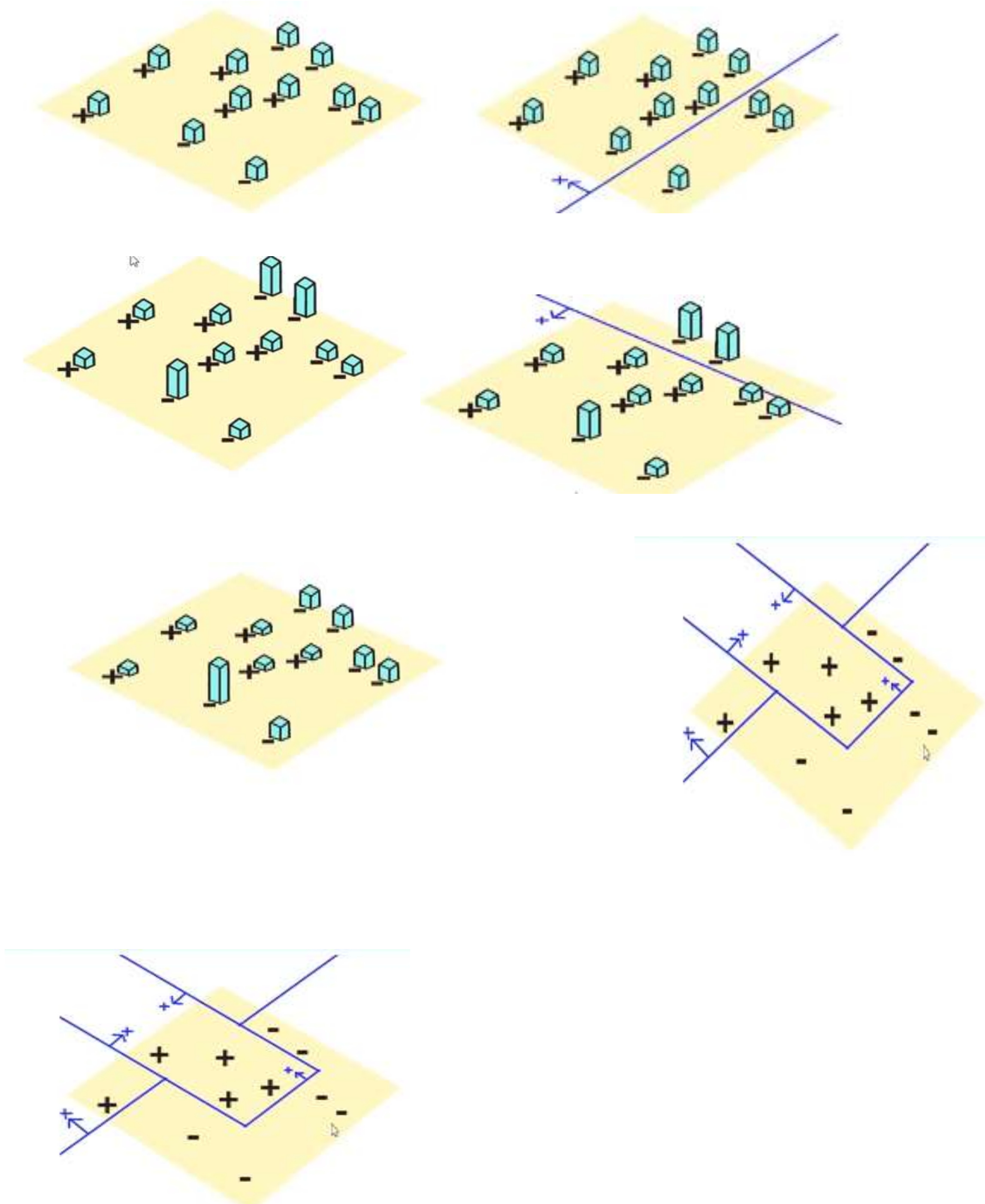


Fig. 3.3 Boosting

Fig. 3.3 describes boosting technique and it can be seen from the figure that the strong learner is the sum of weak learners and the resultant learner is successfully classifying the two classes. Classification was not possible by a single weak learner because the two classes could not be separated by a straight line, but by taking multiple weak learner, problem was solved.

3.2.4 CASCADING CLASSIFIERS

Cascading is used to minimize the false acceptance and false rejection rate and hence, it is used to increase the efficiency. In this, a photo is divided into different blocks. If first classifier is sure that the input area is not a face then that area is not verified further and it is discarded. If first classifier is not sure, then it is passed on to next classifier for further processing. This also helps in increasing the speed of detection.

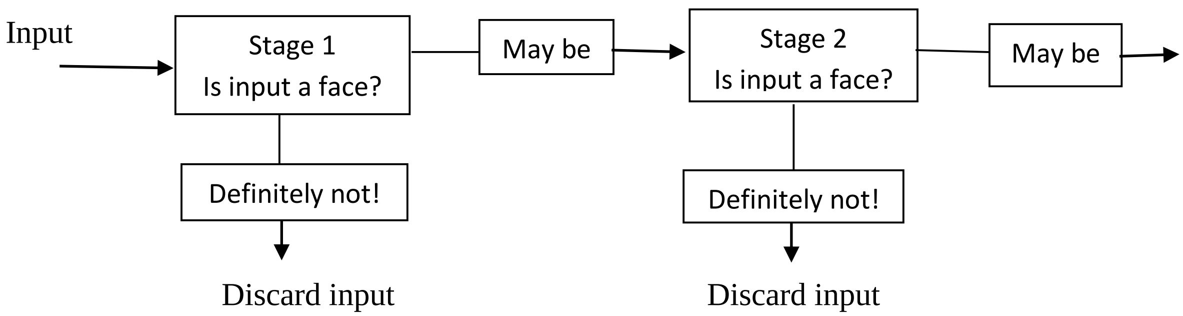


Fig. 3.4 Cascading

Fig. 3.4 describes cascading technique. Two stages are cascaded in the figure. Firstly, input area goes to stage one and chances are there for it to be a face, it goes to second stage otherwise it is discarded and not evaluated further.

CHAPTER 4

SCALE INVARIANT FEATURE TRANSFORM

GENERAL

Scale Invariant Feature Transform (SIFT) was given by David Lowe (2004) which is an image descriptor and it is used for image based matching and recognition. These descriptors are used in computer vision for various purposes which are related to object recognition. Under real conditions, SIFT descriptor is very useful for image matching and object recognition and this is proved experimentally. The greatest advantage is that these features are invariant to scale, light variations, orientation etc. SIFT algorithm provides a set of features of an object which are not affected by clutter, occlusion and unwanted noise in the image. Also, these features are very distinctive in nature which helps in accomplishing correct matching on various pair of feature points with high probability between a large database and a test sample.

SIFT is a local feature extraction technique. A point, edge, or small image patch found in an image which has a pattern or distinct structure refers to a local feature. It is generally associated with a patch is that is different from its immediate surroundings by texture, color, or intensity. What really matters is its distinctiveness from the surrounding. It does not matter what a feature represents in reality. Various examples of local features are blobs, corners, and edge pixels. These features help to find image correspondences regardless of occlusion, changes in viewing conditions, or the presence of clutter. These features help in representing image contents in compact manner which is then used for detection or classification. Good local features exhibit properties like repeatable detections, distinctive, localizable.

Feature detection is a technique of selecting region in an image which has unique content. The points which are useful for further processing are detected by using this. It is not necessary that these points represent a physical structure for example corners of a table. The major concern is to find out points which are locally invariant to rotation or scale change. These detected points are then used for feature extraction which is done with the help of a descriptor. Descriptors use image processing tools to change a local pixel neighbourhood into a compact vector representation. Because of this representation comparison becomes invariant to scale and orientation. Local gradient computations are performed by some descriptors, such as SIFT or SURF. Some work on local intensity variation, BRISK or FREAK, which are then encoded into a binary vector.

4.2 ALGORITHM FOR FEATURE EXTRACTION OF DETECTED FACE

First is the detection of extreme value in space scale. This step is to determine keypoints (face's eyes, nose, mouth etc). At different scales, Image is convolved with the Gaussian filters and the scale image is the result of this convolution. L(x,y,σ) is the description of input image I(x,y).

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y) \quad (4.1)$$

Where $G(x,y,\sigma) = \frac{1}{2\Pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$

Now, the extreme points (key points) are determined from these scale images which are the face keypoints mainly. This step generates too many candidate keypoints and all are not stable. So, filtering is required to be done which uses taylor's series expension, DoG(x,y,σ). This is used to delete the points which have low contrast and this also reduces the edge effect.

$$DoG = L(x,y,k\sigma)-L(x,y,\sigma) \quad (4.2)$$

After filtering, direction of keypoint is needed which will make it invariant to rotation. For the scaled image the magnitude and phase is computed as follows:

$$M(x,y)=\sqrt{((L(x+1,y)-L(x-1,y))^2+(L(x,y+1)-L(x,y-1))^2} \quad (4.3)$$

$$\theta = \arctan\frac{L(x,y+1)-L(x,y-1)}{L(x+1,y)-L(x-1,y)} \quad (4.4)$$

Magnitude and phase is calculated for every pixel around the keypoint. The neighbouring samples are added to the histogram bin. The direction is assigned to the keypoint which corresponds to the maximal component of the histogram. The final step is feature description of keypoints. This step makes it invariant to light variations and 3D viewpoint.

4.3 CONCLUSION

In this chapter, Scale invariant feature transform has been studied and algorithm of the same is discussed.

CHAPTER 5
SUPPORT VECTOR MACHINE

GENERAL

SVM was introduced in 1995 by Vladimir Vapnik and it marked a beginning of a new era. Because of its theoretical and computational merits, SVM gained attention from the pattern recognition community. Various advantages are simple interpretation of the margin, uniqueness of the solution, robustness, effectiveness in high dimensional space, versatility. Support vector machine (SVM) is a supervised machine learning method which can be used for classification and pattern recognition. This algorithm is basically used for binary classification but it can be used for multi-class classification by using different methods. Support Vector Machine is based on the concept of decision plane that defines decision boundaries. Firstly, the training data is mapped into a higher dimensional space which gives a hyperplane which separates one class of objects from another. If this hyperplane is mapped back into original dimensional space it may give a non linear classifier. To construct an optimal hyperplane, SVM performs an iterative training algorithm, which minimizes the error function. The maximum margin hyperplane separating the two classes need to be found, that is why it is known as maximum margin classifier.

Support Vector Machines are based on the concept of decision planes that define decision boundaries. A decision plane is one that separates between a set of objects having different class memberships. Let us take one simple example. In this example, there are two classes in which an object can belong to i.e. GREEN or RED. A boundary line is defined which separates two classes. It has RED to its left and GREEN to its right. Any object which comes to the right is classified as GREEN and if it falls to its left then it is classified as RED.

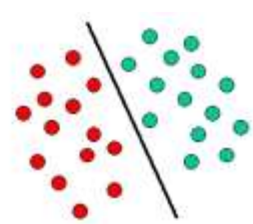


Fig. 5.1 Linear classifier

In fig 5.1, two classes are separated with the help of a line. So, it is an example of linear classifier. Generally classification tasks are not linear and hence not simple and usually more complex structures are required in order to make an optimal separation, i.e., correctly classify new objects (test cases) on the basis of the examples that are available (train cases). One example for non linear classification is shown below in fig. 5.2. As compared to earlier example to classify RED or GREEN objects a curve is required which is certainly more complex than a straight line. Tasks which require to make separating lines which are used to distinguish between different classes are executed by hyperplane classifiers. Support Vector Machines are used to handle such tasks.

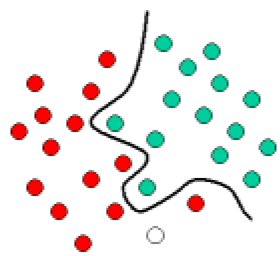


Fig. 5.2 Non-linear classifier

The basic idea behind Support Vector Machines is shown by the figure 5.3 below. By using mathematical functions, also known as kernels, the original objects are mapped on to higher dimension where the problem becomes linear. This mapping process is also known as transformation. The advantage of mapping is the new setting is linearly separable and hence problem becomes easy because we do not require a curve now. After that, it is required to find an optimal line which can separate the two different objects.

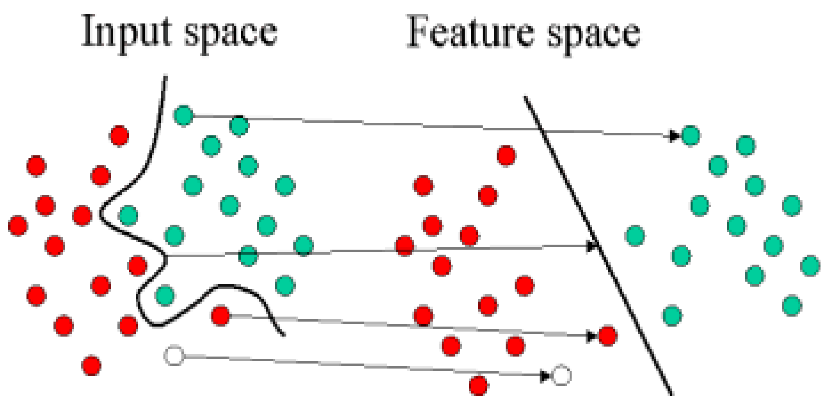


Fig. 5.3 Mapping to higher dimension

SVM can handle both regression and classification tasks and also supports multiple continuous and categorical variables. To construct an optimal hyperplane, SVM employs an iterative training algorithm, which is used to minimize an error function.

5.2 ALGORITHM FOR CLASSIFICATION ON THE BASIS OF FEATURE

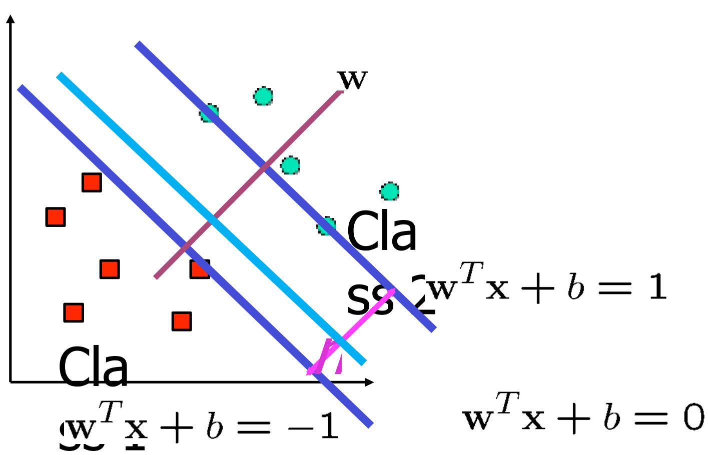


Fig. 5.4 SVM classifier

Let the separating hyperplane is;

$w^T.x+b=0$ (5.1)

In fig. 5.4, two hyperplanes are selected such that they separate the two categories with no data points in between.

$$w^T.x+b=1 \quad (5.2)$$

$$w^T.x+b=-1 \quad (5.3)$$

The above two equations can be summarized as below;

$$y_i(w^T x_i+b)\geq 1 \text{ for all } i \quad (5.4)$$

where $y_i = 1$ is the positive class and $y_i= -1$ is the negative class.

It can be easily seen that the distance between the two boundary hyperplanes is $\frac{2}{||w||}$

The classification problem is converted into optimization one where the minimization of

$$\frac{1}{2}||w||^2 \text{ is required} \quad (5.5)$$

Subject to $y_i(w^T x_i+b)\geq 1$

Vapnik proposed a soft margin classifier that finds the best hyperplane separating the two categories. ζ_i

is slack variable. The optimization problem can be given as

$$Min\frac{1}{2}||w||^2+C\sum\zeta_i \quad (5.6)$$

Subject to $y_i(w^T x_i+b)\geq 1- \zeta_i$, $\zeta_i \geq 0$

5.3 HYBRID SIFT-SVM

In this dissertation, a novel method for face recognition is presented. The idea is to make a hybrid algorithm from two pre-existing algorithms i.e. Scale Invariant Feature Transform (SIFT) and Support Vector Machine (SVM) to enhance the performance. SIFT features of face images are extracted and matched using SVM classifier. In this approach, Few training images are taken to train SVM classifier. A database is created having facial SIFT features of training images of all the individuals. For a new image (person), first of all, the facial region will be detected (if it is not a cropped face image). For detecting face region, Viola-Jones algorithm is used. Then the SIFT algorithm is used for feature extraction which makes it invariant to posture and illumination. Extracted feature vectors, also known as descriptors, and their corresponding locations are obtained from SIFT. Now this feature vector, will be classified using SVM classifier.

To recognize a test sample, the presented method has the three main steps: Firstly, it identifies certain number of training samples on the basis of Euclidean distance between test & training samples. Then the matched pairs of SIFT features between each sample are counted and samples with large number of matched pair are chosen. In third step, the similarity between the test samples and the training samples is calculated and the class with the maximum similarity is chosen as the result. The presented method whose flowchart is shown in fig. 5.5 for better understanding, greatly reduces the complexity in computation and also reduces error recognition due to interference. It also enhances the robustness. Large number of experiments on different public faces images confirm high recognition accuracy. For coding, appendix can be referred.

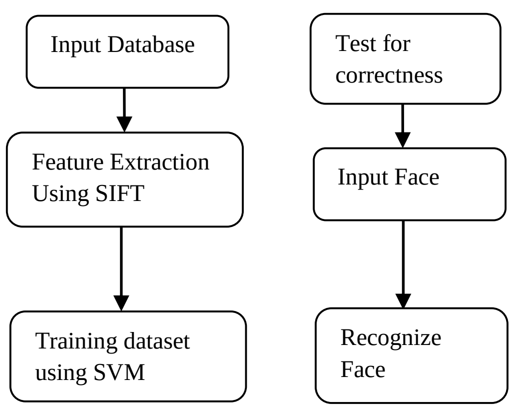


Fig. 5.5 Flow chart of presented model

5.4 CONCLUSION

In this chapter, Support vector machine has been studied and algorithm of the same is discussed. Also, hybrid SIFT-SVM is discussed with the help of a flow chart.

CHAPTER 6
TOOLS AND TECHNOLOGY USED

GENERAL

MATLAB was invented by Cleve Moler. He was a specialist in matrix computation. He worked to make a reliable library on FORTRAN for calculating eigen values of matrices and to solve systems of linear equations. To make the job easier he created MATLAB for his students which actually meant “MATRIX LABORATORY”. So, originally MATLAB was a FORTRAN program. Initially it contained 1 data type and fixed 80 functions. He then used his software to teach numerical analysis at Stanford in 1979. Jack Little, who came to know about this program, tried to use it in the domain of signal processing and control.

Little and Steve Bangert made PC MATLAB by transforming Moler's code from FORTRAN to C, they also added new user-defined functions, improved graphics, libraries, toolboxes. In 1984, These three collectively formed Mathworks which had PC MATLAB as its first product. Their first sale was just 10 copies.

In the today’s world, there is a requirement to quantify estimation, formulation and graphics. To achieve this, a high level language is required which has fourth generation technology. Matlab is developed and updated by Mathwork. Matrix handling is allowed in matlab, algorithms can be implemented, various functions can be plotted, graphical user interface can be designed, various programs in different languages can be merged, data can be analyzed, different applications and models can be created. Built in commands are there which the job easier. Hence for computation of mathematical programs, this is very useful. Because of its various advantages, it is been heavily used in many technical fields for analysis of data, solving problems, and for experimentation and development of algorithm.

The MATLAB system consists of five main parts which are as follows:

Development Environment: These are the set of tools and facilities which allows us to use Matlab files and functions effectively. A lot many tools are GUI which is therefore user friendly. It basically contains Command Window, a command history, an editor and debugger, and help window, the workspace and the search path.

The MATLAB Mathematical Function Library: It contains wide collection of algorithms which range from elementary functions like sinusoidal functions etc, to more complicated functions like FFT, inverse, eigenvalues etc.The MATLAB Language: It is a high level language which allows flow control statements, various functions and it has object oriented programming features also.

Graphics: It has facility to display matrices and vectors in graphs and also it allows annotation and printing of those graphs. Various high level functions are there which helps in two-dimensional and three-dimensional visualization. It also has image processing toolbox and functions for animation and presentations. Not only high level functions, it contains low level functions as well which help in customizing the appearance of graphics and help to make GUI for Matlab applications.

The MATLAB Application Program Interface (API): With the help of this library, various programs can be written in C and Fortran and then these programs can interact with Matlab. It has facilities through which m-files can be read or written and routines can be called from Matlab. Matlab has its application in various domains. For example control systems, image and video processing, signal processing and communication, test and measurement, computational finance, computational biology. To visualize a data in matlab, it has commands matlab. For this, just write the command in the command window. Some common commands which are used by the users are given in table 6.1:

Table 6.1 Basic commands in Matlab

Command	**Purpose**
Clc	Clear command window
Clear	Removes variable from memory
Exist	Check for existence of file or variable
Global	Declares variable to be global
Help	Searches for a help topic
Lookfor	Searches help entries for a keyboard

Quit	Stops Matlab
Who	Lists current variable
Whos	List current variables(long display)

6.1 M files

Previously we have discussed that how to write a command in command window. Let us now see, how to write multiple commands in a single line. How to execute mutilple commands in one go. The M files can be of two types:

Scripts- The program files with .m extension are the example of script file. In these files, various commands can be written and matlab runs these commands in a one go. These files have limitations for example input is not accepted etc.

Functions – The program files which have .m extension in another kind of files are called function file. These files accept the input and return some output. Variables defined in these files are locally defined that is they exist only in fuction file.

To create .m file, matlab editor is used. Script files can call multiple functions. To run a script file, type its name on command window. To open a matlab editor, two ways are there, by using the command prompt or by using IDE.

6.2 Data Types

It is not required to mention data type with the statement. As soon as a new variable is declared, matlab allocates proper space to it and the variable is created. If the variable exceeds the earlier allocated space then it is replaced with a new one and a new space is also allocated to it. Matlab offers 15 different types of data types. Each data type has some functionality. The basic advantage of these data types is any length of array or matrix can be stored with the help of these. The table 6.2 mentions the common data types:

Table 6.2 Data types and their description

Data type	Description
Uint64	64 bit unsigned integer
Int64	64 bit signed integer
Uint 32	32 bit unsigned integer
Int 32	32 bit signed integer
Uint 16	16 bit unsigned integer
Int 16	16 bit signed integer
Uint 8	8 bit unsigned integer
Int 8	8 bit signed integer

6.3 Operators

It is used to perform some operations and this can be guessed by its name only. It is used for logical and mathematical operations. Primarily matlab works with matrices or arrays, but both scalar and non scaler data can be operated by using them. There are many operators for example Relational operators, Logical operators, Bitwise operators, Set operators, Arithmatic operators.

6.4 Vectors

There can be two types of vectors in matlab i.e. Row vectors, Column vectors.

Row Vectors: This type of vector is created when the set of data or element is entered by using comma or space and it is bounded by square brackets.

Column Vectors: This type of vector is created when the set of data or element is entered by using semicolon and is bounded square brackets.

6.5 Plotting

To create a graph in matlab, some steps are needed to be followed which are given below:

1. Define x variable and its range.
2. Define function y.

3. Use plot command to plot y vs x i.e. plot(x,y).

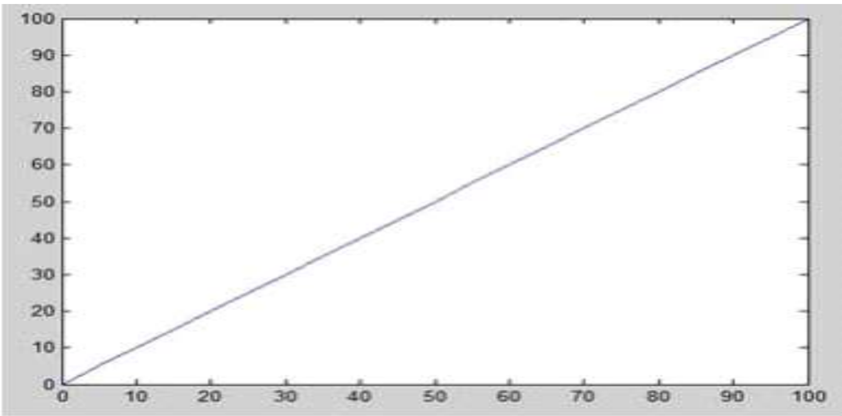


Fig 6.1: Graph plot on Matlab

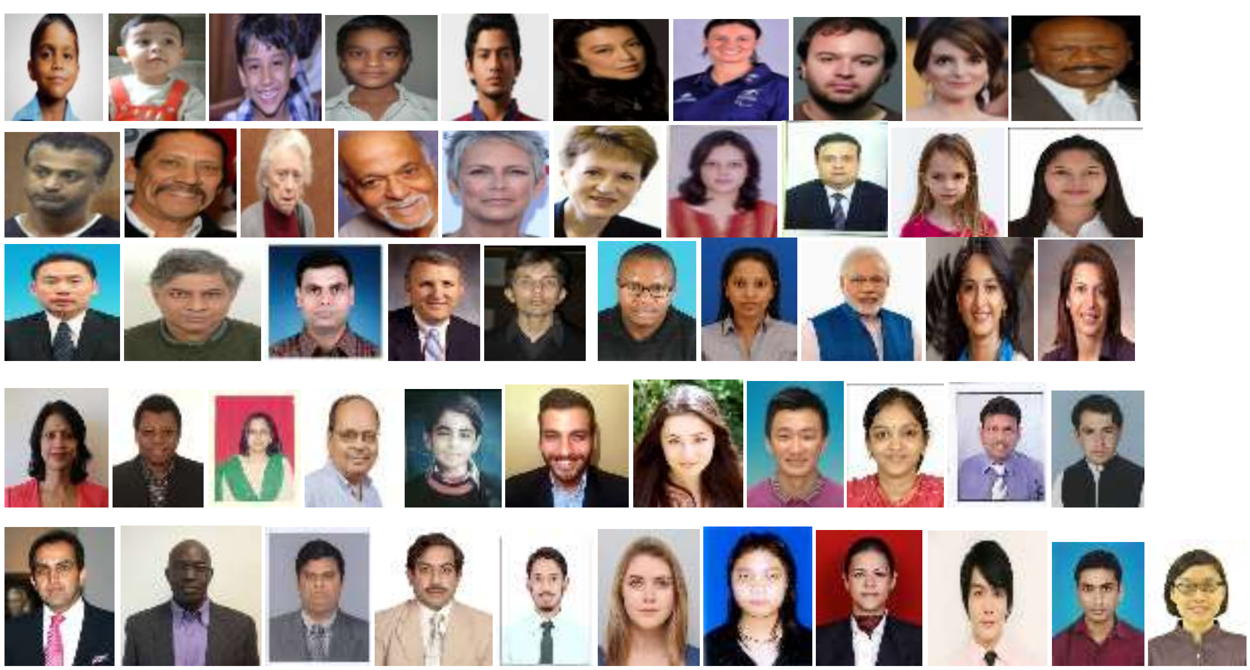
In fig. 6.1, a basic plot of a straight line is shown. There are many other options to it for example adding title, giving name to x-axis and y-axis, adjusting the axes of the graph.

CHAPTER 7
RESULTS AND DISCUSSION

GENERAL

Multi-class classification is needed as in any case there would be many classes (the face images of different people). Simulation has been done on Matlab. Libsvm is used, a library for Support Vector Machines for performing multi-class SVM. For testing the method, standard face databases has been used. The Face Database consists of 75 images. The images of each individual cover a range of poses from frontal towards side view and illumination changes as well. This database is suitable to test perspective change and illumination invariance**.** The below figures show the results of proposed method in which firstly, it detects a face, then it extracts the features using SIFT and finally, it classifies using SVM and with that it recognizes the face. On the basis of age, Analysis is divided into three cases. In every case, some samples are taken and then they are compared with different algorithms in the end and then in the second section it has been tested that the presented model is invariant to posture and expression.

The input database chosen is given in fig. 7.1



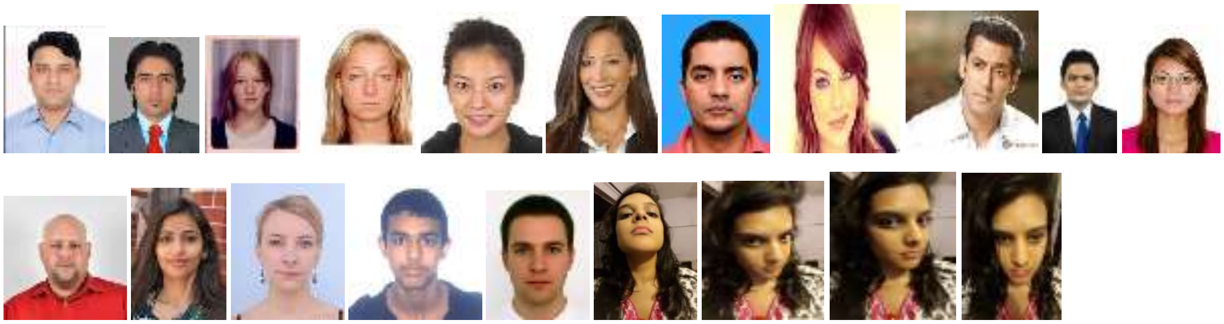


Fig. 7.1 “Candidates” chosen in the first step (Input database)

7.2 STUDY ON DIFFERENT CLASSES MADE ON THE BASIS OF AGE

Three categories are made on the basis of age. Every case is discussed below.

CASE 1. Child

Fig. 7.2 The test samples of children

Fig. 7.3 Extraction of features using SIFT for test images (shown beside) with different pose and illumination

Fig. 7.2 shows the samples of children taken to test the model and results of SIFT algorithm on the sample images is shown in fig. 7.3. SIFT is invariant to pose and illumination variations. So, expected results are obtained for case one. Now, taking these extracted features as an input for SVM which is used for classification, the results are shown in fig. 7.4.

Fig. 7.4 Results of SVM on the extracted SIFT features

Same operation (first SIFT and then SVM) is performed on next section of age i.e. adults.

CASE 2. Adult

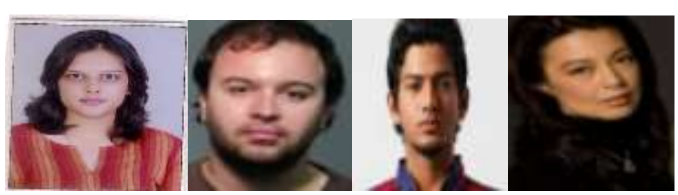


Fig. 7.5 The test samples of adults

Fig. 7.6 Extraction of features using SIFT for test images

Fig. 7.7 Results of SVM on the extracted SIFT features

Fig. 7.5 shows the samples of adults taken to test the model and results of SIFT algorithm on the sample images is shown in fig. 7.6 and fig. 7.7 shows the corresponding SVM results. Third section is old age. Hybrid SIFT-SVM is performed on some old age samples.

CASE 3. Old Age

Fig. 7.8 The test samples of old age people

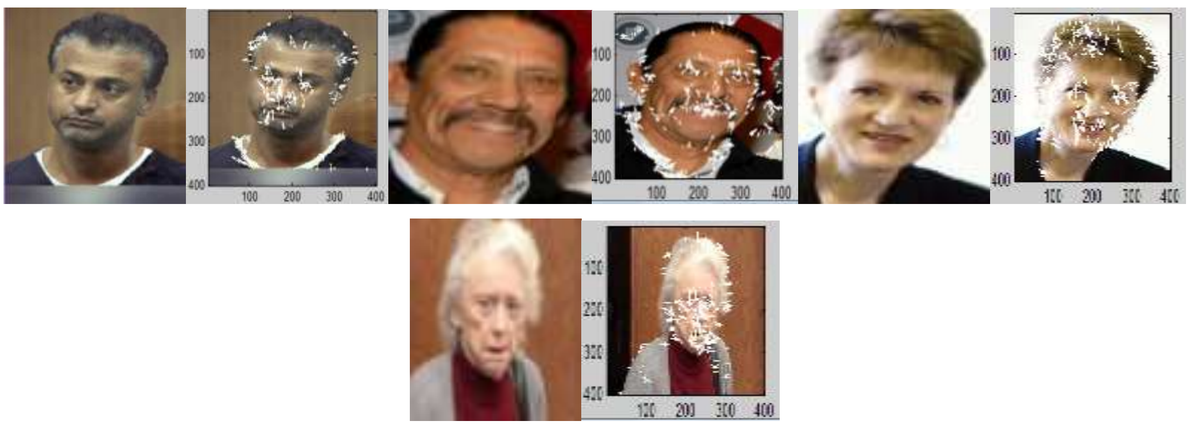


Fig. 7.9 Extraction of features using SIFT for test images

Fig. 7.10 Results of SVM on the extracted SIFT features

Fig. 7.8 shows the samples of old age people taken to test the model and results of SIFT algorithm on the sample images is shown in fig. 7.9 and fig. 7.10 shows the corresponding SVM results. It has been studied that on all the three sections of age, results have good recognizing rate. For comparision with the pre-existing algorithms, some performance measures are taken.

There can be two types of errors in a recognition system: FR (False Rejection) in which system refuses a true face and FA (False Acceptance) in which system accepts a false image. The performance is measured in terms of FRR (False Rejection Rate) and FAR (False Acceptance Rate). By taking these parameters into consideration and varying the number of test and training database images the presented method is compared with PCA, FISHER and SIFT algorithms. Comparision tables are made in which the recognition rate of presented method is compared with some other techniques as shown.

TABLE 7.1 Rate of recognition in different algorithms for Child Case

No of Test Set	50	80	100	150
No of Training Set	50	80	100	150
PCA(%)	46.3	46.3	40.7	53.1
FISHER(%)	74.7	65.7	64.7	61
SIFT(%)	76.7	70.4	68.4	64.3
Hybrid SIFT-SVM(%)	79.9	76.6	69.9	63.8

TABLE 7.2 Rate of recognition in different algorithms for Adult Case

No of Test Set	60	90	120	180
No of Training Set	60	90	120	180
PCA(%)	46.9	47.1	39.8	52.9
FISHER(%)	74.8	64.2	63.7	62.6
SIFT(%)	75.8	69.8	67.9	64.1
Hybrid SIFT-SVM(%)	79.8	74.3	69.2	62.6

TABLE 7.3 Rate of recognition in different algorithms for Old Age Case

No of Test Set	55	70	130	190
No of Training Set	55	70	130	190
PCA(%)	47.2	46.1	37.9	52.6
FISHER(%)	71.3	63.8	61.7	63.2
SIFT(%)	73.2	69.2	67.3	63.9
Hybrid SIFT-SVM(%)	81.2	72.8	68.8	62.7

Table 7.1, 7.2 and 7.3 show that the recognition rate has improved when the proposed method is used which implies that the method is robust and reliable.

7.3 STUDY OF INVARIANCE TO POSTURE AND EXPRESSION

In this section, it has been studied that how the presented model behaves with varying expressions and posture. It has been analysed that the method is invariant to posture and expressions.

The input database is same as mentioned in the starting of the chapter (fig. 7.1).

Test samples are given in fig. 7.11 below.

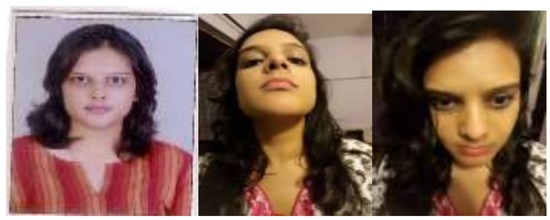


Fig. 7.11 The test samples

Fig 7.12 Extraction of features using SIFT for test images

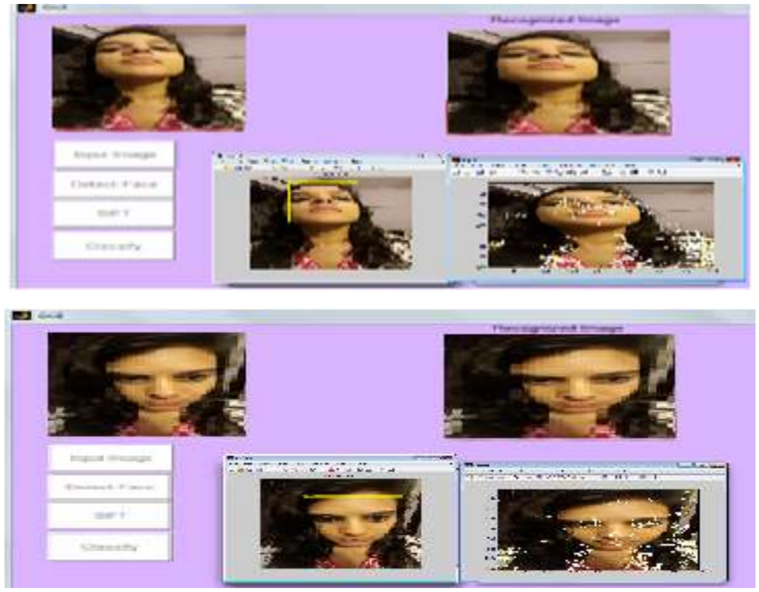


Fig. 7.13 Results of SVM on the extracted SIFT features

Fig. 7.11 shows the samples of different poses of a single person taken to test the model and results of SIFT algorithm on the sample images is shown in fig. 7.12 and fig. 7.13 shows the corresponding SVM results. It can be seen from the results that presented model is giving good accuracy rate, not only for frontal images or faces but also for the different poses. Some of the selfies of a same person is taken here and then results are observed.

CHAPTER 8

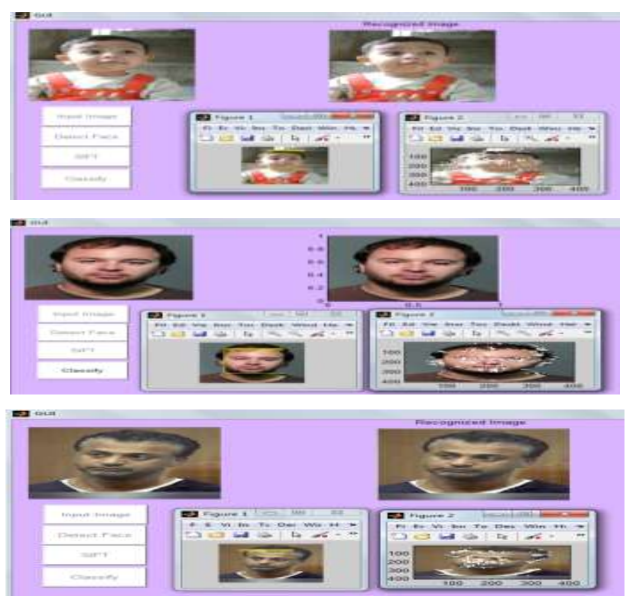
COMPARISION OF HYBRID SIFT-SVM AND SVM

GENERAL

It has been studied and observed that the results for hybrid SIFT-SVM are better than SVM. The input database is same as mentioned in previous chapter i.e. fig. 7.1. Test samples are given in fig. 8.1 below.

Fig. 8.1 The test samples

Fig. 8.2 Extraction of features using SIFT for test images



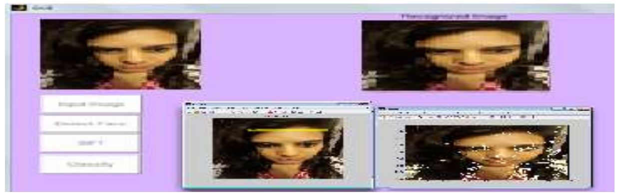


Fig. 8.3 Results of SVM on the extracted SIFT features

Fig. 8.4 Results of SVM algorithm

Fig. 8.1 shows the samples of all age categories taken to test the model and results of SIFT algorithm on the sample images is shown in fig. 8.2 and fig. 8.3 shows the corresponding SVM results on the extracted SIFT features. Fig 8.4 shows the results of only SVM algorithm with no feature extraction. It can be seen that the recognition rate of hybrid SIFT-SVM is better than SVM from the above figures as hybrid SIFT-SVM is giving more accuracy rate in comparision to only SVM. Robustness is more in the proposed model. For coding, appendix can be referred.

CHAPTER 9
CONCLUSION AND FUTURE SCOPE OF WORK

9.1 CONCLUSION

Features of input image is extracted by using SIFT which is a local feature extraction technique and then, SVM is used for classification which takes SIFT output as its input. Hybrid SIFT-SVM is applied on three cases child, adult and old age. In each case, test and training database is varied and proposed model is compared with other algorithms. Tables on three cases show that the recognition accuracy has been increased in comparison to other algorithms and hybrid model outperforms other examined methods. It is more reliable and robust than PCA, ICA, Fisher and SIFT algorithms. Also, it has been studied that the presented model is invariant to posture and expression.

In this dissertation, a comparative analysis has been done in between hybrid SIFT-SVM and SVM and the results shows that presented model is better in terms of accuracy.

9.2 FUTURE SCOPE OF WORK

There are several points which may further be investigated but couldn't be covered in this work due to limited time frame. The main points are described below.

1) The thesis work carried out here is useful for face recognition purpose, if it can be used on real time data then it would be very helpful for security agencies. It can be incorporated through high resolution cameras and then it can be used in real world.

2) The studies done in this work are based upon SIFT and SVM, and it has been compared with different feature extraction technique now instead of SVM, we could try other classifiers and instead of SIFT we could use SURF to see how things turn up.

APPENDIX I

ADJUST ARROW HEAD SIZE

This program is for adjusting the size of arrowhead i.e. to make it bigger or smaller.

```
function adjust_quiver_arrowhead_size(quivergroup_handle, scaling_factor)
% Make quiver arrowheads bigger or smaller.
%
% adjust_quiver_arrowhead_size(quivergroup_handle, scaling_factor)
%
% Example:
%   h = quiver(1:100, 1:100, randn(100, 100), randn(100, 100));
%   adjust_quiver_arrowhead_size(h, 1.5);   % Makes all arrowheads 50%
bigger.
%
% Inputs:
%   quivergroup_handle      Handle returned by "quiver" command.
%   scaling_factor          Factor by which to shrink/grow arrowheads.
%
% Output: none
if ~exist('quivergroup_handle', 'var')
    help(mfilename);
    return
end
if isempty(quivergroup_handle) || any(~ishandle(quivergroup_handle))
    errordlg('Input "quivergroup_handle" is empty or contains invalid
handles.', ...
             mfilename);
    return
end
if length(quivergroup_handle) > 1
    errordlg('Expected "quivergroup_handle" to be a single handle.',
mfilename);
    return
end
if ~strcmpi(get(quivergroup_handle, 'Type'), 'hggroup')
    errrodlg('Input "quivergroup_handle" is not of type "hggroup".',
mfilename);
    return
end
if ~exist('scaling_factor', 'var') || ...
   isempty(scaling_factor) || ...
   ~isnumeric(scaling_factor)
    errordlg('Input "scaling_factor" is missing, empty or non-numeric.', ...
             mfilename);
    return
end
if length(scaling_factor) > 1
    errordlg('Expected "scaling_factor" to be a scalar.', mfilename);
    return
end
if scaling_factor <= 0
    errordlg('"Scaling_factor" should be > 0.', mfilename);
    return
end
line_handles = get(quivergroup_handle, 'Children');
```

```
if isempty(line_handles) || (length(line_handles) < 3) || ...
   ~ishandle(line_handles(2)) || ~strcmpi(get(line_handles(2), 'Type'),
'line')
    errordlg('Unable to adjust arrowheads.', mfilename);
    return
end
arrowhead_line = line_handles(2);
XData = get(arrowhead_line, 'XData');
YData = get(arrowhead_line, 'YData');
if isempty(XData) || isempty(YData)
    return
end
%   Break up XData, YData into triplets separated by NaNs.
first_nan_index = find(~isnan(XData), 1, 'first');
last_nan_index  = find(~isnan(XData), 1, 'last');
for index = first_nan_index : 4 : last_nan_index
    these_indices = index + (0:2);
    if these_indices(end) > length(XData)
        break
    end
    x_triplet = XData(these_indices);
    y_triplet = YData(these_indices);
    if any(isnan(x_triplet)) || any(isnan(y_triplet))
        continue
    end
    %   First pair.
    delta_x = diff(x_triplet(1:2));
    delta_y = diff(y_triplet(1:2));
    x_triplet(1) = x_triplet(2) - (delta_x * scaling_factor);
    y_triplet(1) = y_triplet(2) - (delta_y * scaling_factor);
    %   Second pair.
    x_triplet(3) = x_triplet(2) + (delta_x * scaling_factor);
    y_triplet(3) = y_triplet(2) + (delta_y * scaling_factor);
    XData(these_indices) = x_triplet;
    YData(these_indices) = y_triplet;
end
set(arrowhead_line, 'XData', XData);
```

APPENDIX II

FACE DETECT

To detect a face from an image following code has been used.

```
hsvIm = rgb2hsv(Image);
Im1 = MyConv(hsvIm,MyGauss(5,5));
EdgeIm = rgb2gray(hsvIm-Im1);
SkinImage = Skindetect(hsvIm,EdgeIm);
[m,n]=size(SkinImage);
sumRows=sum(SkinImage,2)/(n*255);
idx=find(sumRows<0.15);
SkinImage(idx,:)=[];
Image(idx,:,:)=[];
[m,n]=size(SkinImage);
sumCols=sum(SkinImage,1)/(m*255);
idx=find(sumCols<0.15);
SkinImage(:,idx)=[];
Image();
[P,clusters] = bwlabel(SkinImage,8);
for i=1:clusters
    [r,c]=find(P==i);
    rMin=min(r);
    rMax=max(r);
    cMin=min(c);
    cMax=max(c);
    t(i,:)=[rMin,cMin,rMax-rMin,cMax-cMin];
end
    area=t(:,3).*t(:,4);
    minIdx=find(area==max(area));
    t=t(minIdx,:);
Face=Image(t(1):t(3)+t(1),t(2):t(2)+t(4),:);
end
```

APPENDIX III

BUILD DETECTOR

To detect a face, a detector is made by following commands.

```
function detector = buildDetector( thresholdFace, thresholdParts, stdsize )
if( nargin < 1 )
 thresholdFace = 1;
end
if( nargin < 2 )
 thresholdParts = 1;
end
if( nargin < 3 )
 stdsize = 176;
end
nameDetector = {'LeftEye'; 'RightEye'; 'Mouth'; 'Nose'; };
mins = [[12 18]; [12 18]; [15 25]; [15 18]; ];
detector.stdsize = stdsize;
detector.detector = cell(5,1);
for k=1:4
 minSize = int32([stdsize/5 stdsize/5]);
 minSize = [max(minSize(1),mins(k,1)), max(minSize(2),mins(k,2))];
 detector.detector{k} = vision.CascadeObjectDetector(char(nameDetector(k)),
'MergeThreshold', thresholdParts, 'MinSize', minSize);
end
detector.detector{5} = vision.CascadeObjectDetector('FrontalFaceCART',
'MergeThreshold', thresholdFace);
function detector = buildDetector( thresholdFace, thresholdParts, stdsize )
 if( nargin < 1 )
 thresholdFace = 1;
end
 if( nargin < 2 )
 thresholdParts = 1;
end
 if( nargin < 3 )
 stdsize = 176;
end
nameDetector = {'LeftEye'; 'RightEye'; 'Mouth'; 'Nose'; };
mins = [[12 18]; [12 18]; [15 25]; [15 18]; ];
detector.stdsize = stdsize;
detector.detector = cell(5,1);
for k=1:4
 minSize = int32([stdsize/5 stdsize/5]);
 minSize = [max(minSize(1),mins(k,1)), max(minSize(2),mins(k,2))];
 detector.detector{k} = vision.CascadeObjectDetector(char(nameDetector(k)),
'MergeThreshold', thresholdParts, 'MinSize', minSize);
end
 detector.detector{5} = vision.CascadeObjectDetector('FrontalFaceCART',
'MergeThreshold', thresholdFace);
```

APPENDIX IV

CALCULATE DOG

In SIFT algorithm, keypoints can be obtained by DOG. The following code is to calculate Difference of Guassian (DOG) images.

```
function calculateDog = calculateDog(octaveStack)
    cellDOG = cell(size(octaveStack,1),1);
    octaves = size(octaveStack,1);
 for i = 1:octaves
        %each octave do the substraction of gaussians
        cellDOG{i} = zeros (size(octaveStack{i},1), size(octaveStack{i},2),
size(octaveStack{i},3), size(octaveStack{i},4)-1);
        cant = size(octaveStack{i},4);
        for j = 2:cant
            %substraction of the previous from the current
            cellDOG{i}(:,:,:,j-1) = octaveStack{i}(:,:,:,j) -
octaveStack{i}(:,:,:,j-1);
%             cant
        end
    end
calculateDog = cellDOG;
end
```

APPENDIX V

CALCULATE KEYPOINTS

Function that calculates the keypoints and filters out the ones on edges and with low contrast.

```
function calculateKeypoints=calculateKeypoints(dogDescriptors, originalImage)
    contrastLimit = 0.03;
    isKeypoint = false;
%   keypointsMap = cell(size(dogDescriptors,1), size(dogDescriptors,1)-2);
    keypointsMap = cell(size(dogDescriptors,1), size(dogDescriptors{1},4)-2);
    cant = 0;
%    size(dogDescriptors{1},3)
    for octave = 1:size(dogDescriptors,1)
        for layer = 2:(size(dogDescriptors{octave},4)-1)
%             keypointsMap{octave}{layer-1} =
zeros(size(dogDescriptors{octave},1), size(dogDescriptors{octave},2));
            keypointsMap{octave,layer-1} =
zeros(size(dogDescriptors{octave},1), size(dogDescriptors{octave},2));
            %each of the points are to be compared - if it takes too long, do
it in C++
            for row = 2:size(dogDescriptors{octave},1)-1
                for column = 2:size(dogDescriptors{octave},2)-1
                    %checks if it is maxima
                    isKeypoint = false;
                    if dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row-1,column,1,layer) && ...
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row-1,column-1,1,layer) && ...
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row,column-1,1,layer) && ...
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row+1,column-1,1,layer) && ...
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row+1,column,1,layer) && ...
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row,column+1,1,layer) && ...
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row-1,column+1,1,layer) && ...
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row-1,column,1,layer-1) && ...
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row-1,column-1,1,layer-1) && ...
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row,column-1,1,layer-1) && ...
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row+1,column-1,1,layer-1) && ...
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row+1,column,1,layer-1) && ...
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row+1,column+1,1,layer-1) && ...
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row,column+1,1,layer-1) && ...
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row-1,column+1,1,layer-1) && ...
```

```
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row,column,1,layer-1) && ...
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row-1,column,1,layer+1) && ...
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row-1,column-1,1,layer+1) && ...
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row,column-1,1,layer+1) && ...
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row+1,column-1,1,layer+1) && ...
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row+1,column,1,layer+1) && ...
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row+1,column+1,1,layer+1) && ...
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row,column+1,1,layer+1) && ...
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row-1,column+1,1,layer+1) && ...
                        dogDescriptors{octave}(row,column,1,layer) >
dogDescriptors{octave}(row,column,1,layer+1)
                        if(keypointsMap{octave,layer-1}(row,column) == 0)
                            cant = cant + 1;
                            isKeypoint = true;
                        end
                    end
                    %checks if it is minima
                    if dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row-1,column,1,layer) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row-1,column-1,1,layer) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row,column-1,1,layer) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row+1,column-1,1,layer) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row+1,column,1,layer) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row+1,column+1,1,layer) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row,column+1,1,layer) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row-1,column+1,1,layer) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row-1,column,1,layer-1) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row-1,column-1,1,layer-1) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row,column-1,1,layer-1) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row+1,column-1,1,layer-1) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row+1,column,1,layer-1) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row+1,column+1,1,layer-1) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row,column+1,1,layer-1) && ...
```

```
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row-1,column+1,1,layer-1) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row,column,1,layer-1) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row-1,column,1,layer+1) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row-1,column-1,1,layer+1) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row,column-1,1,layer+1) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row+1,column-1,1,layer+1) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row+1,column,1,layer+1) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row+1,column+1,1,layer+1) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row,column+1,1,layer+1) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row-1,column+1,1,layer+1) && ...
                        dogDescriptors{octave}(row,column,1,layer) <
dogDescriptors{octave}(row,column,1,layer+1)
                        if(keypointsMap{octave,layer-1}(row,column) == 0)
                            keypointsMap{octave,layer-1}(row,column) = 1;
                            cant = cant + 1;
                            isKeypoint = true;
                        end
                    end
                    %checks the contrast - for now just the simplest way,
                    %without doing tailor expansion
                    if(isKeypoint==true)
if(abs(dogDescriptors{octave}(row,column,1,layer))<contrastLimit)
                            keypointsMap{octave,layer-1}(row,column) = 0;
                            isKeypoint = false;
                            cant = cant - 1;
                        end
                    end
                    %checks the points on the ridges
                    if(isKeypoint==true)
                        DerivativeYY = (dogDescriptors{octave}(row-
1,column,1,layer) + ...
                               dogDescriptors{octave}(row+1, column, 1,
layer) - ...
                               2.0*dogDescriptors{octave}(row, column, 1,
layer));
                        DerivativeXX = (dogDescriptors{octave}(row,column-
1,1,layer) + ...
                               dogDescriptors{octave}(row, column+1, 1,
layer) - ...
                               2.0*dogDescriptors{octave}(row, column, 1,
layer));
                        DerivativeXY = (dogDescriptors{octave}(row-1,column-
1,1,layer) + ...
                               dogDescriptors{octave}(row+1,column+1,1,layer)
- ...
                               dogDescriptors{octave}(row+1, column-1, 1,
layer) - ...
```

```
                                dogDescriptors{octave}(row-1, column+1, 1,
layer))/4;
                        trTerm = DerivativeXX + DerivativeYY;
                        DeterminantH = DerivativeXX * DerivativeYY -
DerivativeXY*DerivativeXY;
                        if(DeterminantH<0)
                           % DeterminantH
                        end
                        ratio = (trTerm*trTerm)/DeterminantH;

                        %r=10 is the value proposed in section 4.1 of Lowe
                        %paper, however experimentally 5 seems to be better
                        %ratio
                        threshold = ((5+1)^2)/5;
                        if(ratio>=threshold || DeterminantH<0)
                            keypointsMap{octave,layer-1}(row,column) = 0;
                            isKeypoint = false;
                            cant = cant -1;
                        end
                    end
                end
            end
        end
    end
    withPointsImage = originalImage;
    returnData = cell(4,1);
    returnData{1} = keypointsMap;
    returnData{2} = withPointsImage;
    returnData{3} = dogDescriptors;
    %number of keypoints
    returnData{4} = cant;
%     qtyDep
    cant;
    calculateKeypoints = returnData;
end
```

APPENDIX VI

CHECK KERNEL

To construct a kernel, following code is used.

```
function K = constructKernel(fea_a,fea_b,options)
% function K = constructKernel(fea_a,fea_b,options)
%   Usage:
%   K = constructKernel(fea_a,[],options)
%
%   K = constructKernel(fea_a,fea_b,options)
%
%   fea_a, fea_b  : Rows of vectors of data points.
%
%   options       : Struct value in Matlab. The fields in options that can
%                   be set:
%           KernelType  -  Choices are:
%               'Gaussian'      - e^{-(|x-y|^2)/2t^2}
%               'Polynomial'    - (x'*y)^d
%               'PolyPlus'      - (x'*y+1)^d
%               'Linear'        -  x'*y
%
%               t       -  parameter for Gaussian
%               d       -  parameter for Poly
%
%   version 1.0 --Sep/2006
%
%   Written by Deng Cai (dengcai2 AT cs.uiuc.edu)
%
if (~exist('options','var'))
   options = [];
else
   if ~isstruct(options)
       error('parameter error!');
   end
end
%=================================================
if ~isfield(options,'KernelType')
    options.KernelType = 'Gaussian';
end
switch lower(options.KernelType)
    case {lower('Gaussian')}        %  e^{-(|x-y|^2)/2t^2}
        if ~isfield(options,'t')
            options.t = 1;
        end
    case {lower('Polynomial')}      % (x'*y)^d
        if ~isfield(options,'d')
            options.d = 2;
        end
    case {lower('PolyPlus')}      % (x'*y+1)^d
        if ~isfield(options,'d')
            options.d = 2;
        end
    case {lower('Linear')}      % x'*y
    otherwise
        error('KernelType does not exist!');
```

```
end
%==================================================
switch lower(options.KernelType)
    case {lower('Gaussian')}
        if isempty(fea_b)
            D = EuDist2(fea_a,[],0);
        else
            D = EuDist2(fea_a,fea_b,0);
        end
        K = exp(-D/(2*options.t^2));
    case {lower('Polynomial')}
        if isempty(fea_b)
            D = full(fea_a * fea_a');
        else
            D = full(fea_a * fea_b');
        end
        K = D.^options.d;
    case {lower('PolyPlus')}
        if isempty(fea_b)
            D = full(fea_a * fea_a');
        else
            D = full(fea_a * fea_b');
        end
        K = (D+1).^options.d;
    case {lower('Linear')}
        if isempty(fea_b)
            K = full(fea_a * fea_a');
        else
            K = full(fea_a * fea_b');
        end
    otherwise
        error('KernelType does not exist!');
end
if isempty(fea_b)
    K = max(K,K');
end
```

APPENDIX VII

DEFINE ORIENTATION

SIFT features not only have magnitude but also a direction. So, following code is to define the orientation.

```
function defineOrientation=defineOrientation(genDescriptor, dogDescriptor,
...
                                octaveDescriptor, originalImage, accumSigmas)
    %First, the gradient magnitudes and orientations are calculated for
    %each pixel in each of L scaled images, such as indicated in the
    %paragraph 2 of section 5 of 1, later these magnitudes/orientations
    keypointDescriptor = genDescriptor{1};
    orientMagn =
cell(size(octaveDescriptor,1),size(octaveDescriptor{1},4),2);
    for octaveId = 1:size(octaveDescriptor,1)
        for scaleId = 1:size(octaveDescriptor{octaveId},4)
            %in order not to iterate over each pixel, I will try to calculate
the gradient using
            %filter and matrix operations:
            %filter for calculating diffX:
            filterDiffX = [0 0 0; -1 0 1; 0 0 0];
            diffXMat = imfilter(octaveDescriptor{octaveId}(:,:,1,scaleId),
filterDiffX);
            %filter for calculating diffY:
            filterDiffY = [0 1 0; 0 0 0; 0 -1 0];
            diffYMat = imfilter(octaveDescriptor{octaveId}(:,:,1,scaleId),
filterDiffY);
            %get the magnitude operating directly on matrixes:
            magnMat = sqrt(diffXMat.*diffXMat + diffYMat.*diffYMat);
            %do similar thing for orientation
            orientMat = atan2(diffYMat, diffXMat);
            %however, the atan function only gives the orientation respect
            %to a particular quadrant, atan2 function must be used
            %store magnitude and orientation, so they can be used later on
            orientMagn{octaveId}{scaleId}{1} = magnMat;
            orientMagn{octaveId}{scaleId}{2} = orientMat;
        end
    end
    %four elements for each layer in the octave: coordinates, histograms,
    %position of best histograms
    orientationDescriptor = cell(size(keypointDescriptor,
1),size(keypointDescriptor,2));
    %36 buckets in histogram
    hist = zeros(36,1);
    cant = 0;
    %for each keypoints octave
    for octave = 1:size(keypointDescriptor, 1)
        %for each keypoints layer
        for kptLayer = 1:size(keypointDescriptor,2)
            %Once the magnitudes and orientations have been calculated, it is
            %necessary to calculate the orientation histogram explained in
slides
            %36 and 37 of [2] seen in class. There is one histogram for each
            %keypoing, each one can have one or more orientations.
```

```
            %gets the indices of all the elements that are keypoints in a
            %particular keypoint level.
            [rowKpt colKpt] = find(keypointDescriptor{octave,kptLayer} == 1);

            %gaussian kernel with sigma 1.5 times of the sigma
            %corresponding to the scale of the keypoint
            %TODO: kptLayer or kptLayer+1?
            accumSigma = accumSigmas(octave, kptLayer)*1.5;
            weightKernel = fspecial('gaussian',[round(accumSigma*6-1)
round(accumSigma*6-1)], accumSigma);
            knlHeight = size(weightKernel,1);
            knlWidth = size(weightKernel,2);
            winHeight = size(orientMagn{octave}{kptLayer}{1},1);
            winWidth = size(orientMagn{octave}{kptLayer}{1},2);
               %-------new part to see if improves
                totWeighted = orientMagn{octave}{kptLayer}{1};
%orientMagn{octave}{kptLayer}{1};
%conv2(orientMagn{octave}{kptLayer}{1},weightKernel);
                %-------end new part
            for keypoint = 1:size([rowKpt colKpt],1)
                xfrom = round(colKpt(keypoint)-knlWidth/2);
                xto = round(colKpt(keypoint)+knlWidth/2-1);
                yfrom = round(rowKpt(keypoint)-knlHeight/2);
                yto = round(rowKpt(keypoint)+knlHeight/2-1);
                truncXKnlLeft = 0;
                truncXKnlRight = 0;
                truncYKnlTop = 0;
                truncYKnlBottom = 0;
                if(xfrom<1)
                    xfrom = 1;
                    truncXKnlLeft = knlWidth-(xto-xfrom)-1;
                end
                if(yfrom<1)
                    yfrom = 1;
                    truncYKnlTop = knlHeight-(yto-yfrom)-1;
                end
                if(xto>winWidth)
                    xto = winWidth;
                    truncXKnlRight = knlWidth-truncXKnlLeft-(xto-xfrom+1);
                end
                if(yto>winHeight)
                    yto=winHeight;
                    truncYKnlBottom = knlHeight - truncYKnlTop-(yto-yfrom+1);
                end
                %truncates kernel if necessary
                weightKernelEval =
                %gets the matrix of magnitude values
%                magnitudes =
orientMagn{octave}{kptLayer}{1}(yfrom:yto,xfrom:xto);
                magnitudes = totWeighted(yfrom:yto,xfrom:xto);
                cant=cant+1;
                %applies the weight of the kernel to matrix, getting
                %weighted magnitudes
                magnitudes = weightKernelEval.*magnitudes;
                %gets the matrix of orientations
                orientations =
orientMagn{octave}{kptLayer}{2}(yfrom:yto,xfrom:xto);
```

```
                %transforms orientations to degrees in order to distribute
                %them into buckets
                orientations = (orientations.*180)./pi; % + 180;
                %for each bucket get the magnitudes
                for bucket=1:36
                    bucketRangeFrom = (bucket-19)*10;
                    bucketRangeTo = (bucket-18)*10;
                    [rowOr, colOr] = find(orientations<bucketRangeTo &
orientations>=bucketRangeFrom);
%                    indexes = sub2ind(size(weightedMagnitudes),rowOr,colOr);
%                    hist(bucket) = sum(weightedMagnitudes(indexes));
                    indexes = sub2ind(size(magnitudes),rowOr,colOr);
                    hist(bucket) = sum(magnitudes(indexes));
                end
                %finds the position of highest peak of the histogram
                posMaxHist = find(hist==max(hist));
                %finds those that are within 80% of the highest peak
                posOtherHist = find(hist>(max(hist)-
max(hist)*0.2)&hist~=hist(posMaxHist(1)));
                posAllHist = zeros(1,1);
                if(size(posOtherHist,1)>0)
                    posAllHist = cat(2,posMaxHist,posOtherHist.');
                else
                    posAllHist = posMaxHist;
                end
                interpolatedOrientations = zeros(size(posAllHist,1),1);
                %in section 5 (par 4) of [1] says: "Finally, a parabola is
fit to the 3 histogram values
                %closest to each peak to interpolate the peak position for
                %better accuracy".
                for currentBestHist = 1:size(posAllHist,2)
                    posHist = posAllHist(currentBestHist);
                    x1 = posHist-1;
                    x2 = posHist;
                    x3 = posHist+1;
                    y1 = 0;
                    y2 = hist(x2);
                    y3 = 0;
                    %in order not to lose the topology
                    if(x1<1)
                        y1 = hist(36);
                    else
                        y1 = hist(x1);
                    end
                    if(x3>36)
                        y3 = hist(1);
                    else
                        y3 = hist(x3);
                    end
                    valsX = [x1-0.5 x2-0.5 x3-0.5];
%                    valsX = [x1 x2 x3];
                    valsY = [y1 y2 y3];
                    pars = polyfit(valsX,valsY,2);
                    %result of derivative = 0 to see where is the parabolic
maxima
                    xMax = (pars(2)*(-1))/(2*pars(1));
```

```
                    xMax = xMax;
                    if(xMax<0)
                        xMax = 36+xMax;
                    end
                    if(xMax>36)
                        xMax = xMax-36;
                    end
                    %now, convert to degrees
                    xMax = xMax * 10;
                    interpolatedOrientations(currentBestHist) = xMax;
                end
                %creates the structure with the data
                histDescriptor = struct('octave', octave, ...
                                        'layer', kptLayer, ...
                                        'position',[rowKpt(keypoint)
colKpt(keypoint)], ...
                                        'histogram', hist, ...
                                        'bestHist', posAllHist.', ...
                                        'interpOrien',
interpolatedOrientations.', ...
                                        'theBestHist', posMaxHist);
orientationDescriptor{octave}{kptLayer}(rowKpt(keypoint),colKpt(keypoint)) =
histDescriptor;
            end
        end
    end
    %returns orientation descriptor along with magnitudes and orientations
    retCell = cell(2);
    retCell{1} = orientationDescriptor;
    retCell{2} = orientMagn;
    defineOrientation = retCell;
end
```

APPENDIX VIII

DETECT FACE PARTS

Computer vision toolbox in matlab is used to detect the face parts.

```
function [bbox,bbX,faces,bbfaces] = detectFaceParts(detector,X,thick)
if( nargin < 3 )
 thick = 1;
end
%%%%%%%%%%%%%%%%%%%%%% detect face %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Detect faces
bbox = step(detector.detector{5}, X);
bbsize = size(bbox);
partsNum = zeros(size(bbox,1),1);
%%%%%%%%%%%%%%%%%%%%%% detect parts %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nameDetector = {'LeftEye'; 'RightEye'; 'Mouth'; 'Nose'; };
mins = [[12 18]; [12 18]; [15 25]; [15 18]; ];
stdsize = detector.stdsize;
for k=1:4
 if( k == 1 )
  region = [1,int32(stdsize*2/3); 1, int32(stdsize*2/3)];
 elseif( k == 2 )
  region = [int32(stdsize/3),stdsize; 1, int32(stdsize*2/3)];
 elseif( k == 3 )
  region = [1,stdsize; int32(stdsize/3), stdsize];
 elseif( k == 4 )
  region = [int32(stdsize/5),int32(stdsize*4/5); int32(stdsize/3),stdsize];
 else
  region = [1,stdsize;1,stdsize];
 end
 bb = zeros(bbsize);
 for i=1:size(bbox,1)
  XX = X(bbox(i,2):bbox(i,2)+bbox(i,4)-1,bbox(i,1):bbox(i,1)+bbox(i,3)-1,:);
  XX = imresize(XX,[stdsize, stdsize]);
  XX = XX(region(2,1):region(2,2),region(1,1):region(1,2),:);
  b = step(detector.detector{k},XX);
  if( size(b,1) > 0 )
   partsNum(i) = partsNum(i) + 1;
   if( k == 1 )
    b = sortrows(b,1);
   elseif( k == 2 )
    b = flipud(sortrows(b,1));
   elseif( k == 3 )
    b = flipud(sortrows(b,2));
   elseif( k == 4 )
    b = flipud(sortrows(b,3));
   end
   ratio = double(bbox(i,3)) / double(stdsize);
   b(1,1) = int32( ( b(1,1)-1 + region(1,1)-1 ) * ratio + 0.5 ) +
bbox(i,1);
   b(1,2) = int32( ( b(1,2)-1 + region(2,1)-1 ) * ratio + 0.5 ) + bbox(i,2);
   b(1,3) = int32( b(1,3) * ratio + 0.5 );
   b(1,4) = int32( b(1,4) * ratio + 0.5 );
   bb(i,:) = b(1,:);
  end
 end
```

```
 bbox = [bbox,bb];
 p = ( sum(bb') == 0 );
 bb(p,:) = [];
end



%%%%%%%%%%%%%%%%%%%%%%% draw faces %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
bbox = [bbox,partsNum];
bbox(partsNum<=2,:)=[];
if( thick >= 0 )
 t = (thick-1)/2;
 t0 = -int32(ceil(t));
 t1 = int32(floor(t));
else
 t0 = 0;
 t1 = 0;
end
bbX = X;
boxColor = [[0,255,0]; [255,0,255]; [255,0,255]; [0,255,255]; [255,255,0]; ];
for k=5:-1:1
 shapeInserter =
vision.ShapeInserter('BorderColor','Custom','CustomBorderColor',boxColor(k,:)
);
 for i=t0:t1
  bb = int32(bbox(:,(k-1)*4+1:k*4));
  bb(:,1:2) = bb(:,1:2)-i;
  bb(:,3:4) = bb(:,3:4)+i*2;
  bbX = step(shapeInserter, bbX, bb);
 end
end
%%%%%%%%%%%%%%%%%%%%%%% faces %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if( nargout > 2 )
 faces = cell(size(bbox,1),1);
 bbfaces = cell(size(bbox,1),1);
 for i=1:size(bbox,1)
  faces{i,1} = X(bbox(i,2):bbox(i,2)+bbox(i,4)-
1,bbox(i,1):bbox(i,1)+bbox(i,3)-1,:);
  bbfaces{i,1} = bbX(bbox(i,2):bbox(i,2)+bbox(i,4)-
1,bbox(i,1):bbox(i,1)+bbox(i,3)-1,:);
 end
end
```

APPENDIX IX

DETECT ROATATION IN FACE PARTS

Code is to detect how much rotation is there in face as compared to a frontal image.

```
function [fourpoints,bbX,faces,bbfaces] =
detectRotFaceParts(detector,X,thick,rotate)
if( nargin < 4 )
 rotate = 15;
end
if( nargin < 3 )
 thick = 1;
end
rotate = [0:rotate:360-rotate/2];
srcOrg = [size(X,2);size(X,1)]/2+0.5;
fourpoints = [];
k = 1;
for deg = rotate
 R = imrotate(X,deg,'bicubic');
 bbox = detectFaceParts(detector,R);
 if( size(bbox,1) >= 1 )
  dstOrg = [size(R,2);size(R,1)]/2+0.5;
  fourpoints = vertcat(fourpoints,bbox2fourpoint(bbox,srcOrg,dstOrg,deg));
 end
end
fourpoints = mergeFourPoints(fourpoints);
if( nargout >= 2 )
 bbX = drawFourPoints(X,fourpoints,thick);
 if( nargout >= 3 )
  faces = cell(size(fourpoints,1),1);
  bbfaces = cell(size(fourpoints,1),1);
  leng = round(sqrt( three2area( fourpoints(:,1:2), fourpoints(:,3:4),
fourpoints(:,5:6) ) + three2area( fourpoints(:,1:2), fourpoints(:,7:8),
fourpoints(:,5:6) ) ));
  for i=1:size(fourpoints,1)
   U = [1,1;leng(i,1)-1,1;leng(i,1)-1,leng(i,1)-1;1,leng(i,1)-1];
   V = [fourpoints(i,1:2); fourpoints(i,3:4); fourpoints(i,5:6);
fourpoints(i,7:8)];
   T = maketform('projective',V,U);
   faces{i,1} =
imtransform(X,T,'bicubic','XData',[1,leng(i,1)],'YData',[1,leng(i,1)]);
   bbfaces{i,1} =
imtransform(bbX,T,'bicubic','XData',[1,leng(i,1)],'YData',[1,leng(i,1)]);
  end
 end
end
function fourpoint = bbox2fourpoint( bbox, srcOrg, dstOrg, deg )
T = [cos(deg*pi/180), -sin(deg*pi/180); sin(deg*pi/180), cos(deg*pi/180)];
fourpoint = zeros(size(bbox,1), 2*4*5+1);
for i=1:size(bbox,1)
 for j=0:4
  if( bbox(i,j*4+1) > 0 && bbox(i,j*4+2) > 0 )
   x = bbox(i,j*4+1:j*4+2)' - dstOrg;
   y = T * x + srcOrg;
```

```
   fourpoint(i,j*8+1:j*8+2) = y';
   x = bbox(i,j*4+1:j*4+2)' + [bbox(i,j*4+3);0] - dstOrg;
   y = T * x + srcOrg;
   fourpoint(i,j*8+3:j*8+4) = y';
   x = bbox(i,j*4+1:j*4+2)' + [bbox(i,j*4+3);bbox(i,j*4+4)] - dstOrg;
   y = T * x + srcOrg;
   fourpoint(i,j*8+5:j*8+6) = y';
   x = bbox(i,j*4+1:j*4+2)' + [0;bbox(i,j*4+4)] - dstOrg;
   y = T * x + srcOrg;
   fourpoint(i,j*8+7:j*8+8) = y';
  end
 end
 fourpoint(i,2*4*5+1) = deg;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function area = three2area(xy1, xy2, xy3)
xy1 = xy1 - xy3;
xy2 = xy2 - xy3;
area = abs( xy1(:,1) .* xy2(:,2) - xy1(:,2) .* xy2(:,1) ) / 2;
```

APPENDIX X

MATCHING OF DESCRIPTORS OF TWO IMAGES

Code is to compare two descriptors or SIFT features.

```
%get matches between descriptors of two different images
function getMatches = getMatches(descriptorImage1, descriptorImage2)
    %in [1] it is recommended also to take into account the second nearest
    %neighbour and ignore it if the distance is more than 0.8 between these
    %two neighbours
    cant = 0;
    matches = repmat(struct('descriptorIm1',descriptorImage1(1), ...
        'descriptorIm2',descriptorImage2(1)), size(descriptorImage1,1),1);
    indexMatches = 1;
    for keypointIm1 = 1:size(descriptorImage1,1)
        bestL2Diff=9999999999;
        bestL2Index = -1;
        secondBestL2Diff=9999999999;
        for keypointIm2 = 1:size(descriptorImage2,1)
            l2Difference = getL2Difference(descriptorImage1(keypointIm1), ...
                                            descriptorImage2(keypointIm2));
            if(l2Difference<bestL2Diff)
                secondBestL2Diff = bestL2Diff;
                bestL2Diff = l2Difference;
                bestL2Index = keypointIm2;
                if(secondBestL2Diff==9999999999)
                    secondBestL2Diff = l2Difference;
                end
            end
        end
        diffBestSecond = secondBestL2Diff-bestL2Diff;
        ratioBestSecond = double(bestL2Diff)/double(diffBestSecond);
        if(diffBestSecond~=0 && bestL2Diff~=0 && ratioBestSecond>1.3)
            %ignore the keypoint
            %'ignore'
            ratioBestSecond;
        else
            %add the keypoint to matches
            matchStruct = struct('descriptorIm1',
descriptorImage1(keypointIm1), ...
                                    'descriptorIm2',
descriptorImage2(bestL2Index));

            matches(indexMatches) = matchStruct;
            indexMatches = indexMatches + 1;
        end
    end
    getMatches = matches;
    indexMatches
    function getL2Difference = getL2Difference(descriptor1, descriptor2)
        cant = cant+1;
        l2Diff = [];
        if(cant==40641)
            l2Diff = sqrt(sum((descriptor1.kptDescriptor-
descriptor2.kptDescriptor).^2));
        end
```

```
        l2Diff = sqrt(sum((descriptor1.kptDescriptor-
descriptor2.kptDescriptor).^2));
        getL2Difference = l2Diff;
```

APPENDIX XI

SIFT DESCRIPTOR

Code is to calculate the SIFT descriptor of an input image and to plot it.

```
function siftDescriptor = siftDescriptor()
    image1 = imread('siftface.jpg');
    %image2 = imread('model2.png');
    %define the scale space
    retScaleSpace = scaleSpace(image1,4,3);
    octaveStack = retScaleSpace{1};
    accumSigmas = retScaleSpace{2};
    octaveDOGStack = calculateDog(octaveStack);
    keypoints = calculateKeypoints(octaveDOGStack, image1);
    orientationDef = defineOrientation(keypoints, octaveDOGStack, ...
               octaveStack, image1, accumSigmas);
    descriptor = localDescriptor_v3(orientationDef, keypoints, ...
               accumSigmas, size(image1,1)*2, size(image1,2)*2);
%to plot the descriptor, uncomment and comment the rest of the code below
figure,
    plotDescriptor(descriptor, image1, orientationDef, keypoints);
    retScaleSpace2 = scaleSpace(image2,4,3);
    octaveStack2 = retScaleSpace2{1};
    accumSigmas2 = retScaleSpace2{2};
    octaveDOGStack2 = calculateDog(octaveStack2);
    keypoints2 = calculateKeypoints(octaveDOGStack2, image2);
    orientationDef2 = defineOrientation(keypoints2, octaveDOGStack2, ...
                octaveStack2, image2, accumSigmas2);
    descriptor2 = localDescriptor_v3(orientationDef2, keypoints2, ...
                accumSigmas2, size(image2,1)*2, size(image2,2)*2);
   plotDescriptor(descriptor2, image2, orientationDef2, keypoints2);
    matches = getMatches(descriptor, descriptor2);
    plotMatches(image1,image2,matches);
    siftDescriptor = keypoints;
end
```

APPENDIX XII

SVM CLASSIFY

To classify images, SVM code is as follows.

```
% Load Datasets
Dataset = 'TrainDataBase';
Testset  = 'TestDataBase';
% we need to process the images first.
% Convert your images into grayscale
% Resize the images
width=100; height=100;
DataSet      = cell([], 1);
 for i=1:length(dir(fullfile(Dataset,'*.jpg')))
     % Training set process
     k = dir(fullfile(Dataset,'*.jpg'));
     k = {k(~[k.isdir]).name};
     for j=1:length(k)
        tempImage       = imread(horzcat(Dataset,filesep,k{j}));
        imgInfo         = imfinfo(horzcat(Dataset,filesep,k{j}));
         % Image transformation
         if strcmp(imgInfo.ColorType,'grayscale')
            DataSet{j}   = double(imresize(tempImage,[width height])); %
array of images
         else
            DataSet{j}   = double(imresize(rgb2gray(tempImage),[width
height])); % array of images
         end
     end
 end
TestSet =  cell([], 1);
  for i=1:length(dir(fullfile(Testset,'*.jpg')))
     % Training set process
     k = dir(fullfile(Testset,'*.jpg'));
     k = {k(~[k.isdir]).name};
     for j=1:length(k)
        tempImage       = imread(horzcat(Testset,filesep,k{j}));
        imgInfo         = imfinfo(horzcat(Testset,filesep,k{j}));
         % Image transformation
         if strcmp(imgInfo.ColorType,'grayscale')
            TestSet{j}   = double(imresize(tempImage,[width height])); %
array of images
         else
            TestSet{j}   = double(imresize(rgb2gray(tempImage),[width
height])); % array of images
         end
     end
  end
train_label               =
zeros(size(length(dir(fullfile(Dataset,'*.jpg'))),1),1);
train_label(1:8,1)   = 1;
train_label(8:length(dir(fullfile(Dataset,'*.jpg'))),1)   = 2;
% Prepare numeric matrix for svmtrain
Training_Set=[];
for i=1:length(DataSet)
```

```
    Training_Set_tmp   = reshape(DataSet{i},1, 100*100);
    Training_Set=[Training_Set;Training_Set_tmp];
end
Test_Set=[];
for j=1:length(TestSet)
    Test_set_tmp   = reshape(TestSet{j},1, 100*100);
    Test_Set=[Test_Set;Test_set_tmp];
end
% Perform first run of svm
SVMStruct = svmtrain(Training_Set , train_label, 'kernel_function',
'linear');
Group       = svmclassify(SVMStruct, Test_Set);
```

APPENDIX XIII

SIFT

This function reads an image and returns its SIFT keypoints.

```
% [image, descriptors, locs] = sift(imageFile)
%
%
%   Input parameters:
%     imageFile: the file name for the image.
%
%   Returned:
%     image: the image array in double format
%     descriptors: a K-by-128 matrix, where each row gives an invariant
%         descriptor for one of the K keypoints.  The descriptor is a vector
%         of 128 values normalized to unit length.
%     locs: K-by-4 matrix, in which each row has the 4 values for a
%         keypoint location (row, column, scale, orientation).  The
%         orientation is in the range [-PI, PI] radians.
%
function [image, descriptors, locs] = sift(imageFile)
% Load image
%imagefile='model1.png';
image = imread(imageFile);
% If you have the Image Processing Toolbox, you can uncomment the following
%   lines to allow input of color images, which will be converted to
grayscale.
 if isrgb1(image)
    image = rgb2gray(image);
 end
[rows, cols] = size(image);
% Convert into PGM imagefile, readable by "keypoints" executable
f = fopen('tmp.pgm', 'w');
if f == -1
    error('Could not create file tmp.pgm.');
end
fprintf(f, 'P5\n%d\n%d\n255\n', cols, rows);
fwrite(f, image', 'uint8');
fclose(f);
% Call keypoints executable
if isunix
    command = '!./sift ';
else
    command = '!siftWin32 ';
end
command = [command ' <tmp.pgm >tmp.key'];
eval(command);
% Open tmp.key and check its header
g = fopen('tmp.key', 'r');
if g == -1
    error('Could not open file tmp.key.');
end
[header, count] = fscanf(g, '%d %d', [1 2]);
if count ~= 2
    error('Invalid keypoint file beginning.');
end
```

```matlab
num = header(1);
len = header(2);
if len ~= 128
    error('Keypoint descriptor length invalid (should be 128).');
end
% Creates the two output matrices (use known size for efficiency)
locs = double(zeros(num, 4));
descriptors = double(zeros(num, 128));
% Parse tmp.key
for i = 1:num
    [vector, count] = fscanf(g, '%f %f %f %f', [1 4]); %row col scale ori
    if count ~= 4
        error('Invalid keypoint file format');
    end
    locs(i, :) = vector(1, :);
    [descrip, count] = fscanf(g, '%d', [1 len]);
    if (count ~= 128)
        error('Invalid keypoint file value.');
    end
    % Normalize each input vector to unit length
    descrip = descrip / sqrt(sum(descrip.^2));
    descriptors(i, :) = descrip(1, :);
end
fclose(g);
%eval('!rm -f tmp.pgm');
%eval('!rm -f tmp.key');
```

APPENDIX XIV

SCALE SPACE

After constructing DOG function, Keypoints are taken at the scale space extreme Dog function convolved with the image. Below code is for the same.

```
function scaleSpace=scaleSpace(image, octaves, scales)
     grayScaleIm = rgb2gray(image);
    grayScaleIm = double(grayScaleIm)/double(255.0);
    firstBlurSigma = 0.5;
    kernelSize = 15;
    %step 1: double the image size prior to building the first level of the 
pyramid
    %this must be done after bluring the original image with gaussian of 
sigma = 0.5. This is suggested in the section
    %3.3 in paper [2].
    initialBluredImage = gaussianBlur(grayScaleIm,firstBlurSigma,kernelSize);
    inDSI = imresize(grayScaleIm, 2, 'bilinear'); %grayScaleIm; 
%imresize(grayScaleIm, 2, 'bilinear');
    initialDoubleSizeImage = inDSI;
%   initialDoubleSizeImage = 
gaussianBlur(initialDoubleSizeImage,1,kernelSize);
    %in section 3.3 of [2] is suggested to use sigma = 1.6
    initialSigma = sqrt(2); %1.6; %sqrt(2);
%    initialSigma = 1.6;
    currentSigma = initialSigma;
    totScales = scales + 3;
    cellOctaves = cell(octaves,1);
    previousDoubleSizeImage = initialDoubleSizeImage ;
    %this matrix will contain the values of accumulated sigmas and will be
    %used to calculate orientation histogram weight later on
    accumSigmas = zeros(octaves, totScales);
    for octave = 1:octaves
        sigma = zeros(size(initialDoubleSizeImage,1), 
size(initialDoubleSizeImage,2), size(initialDoubleSizeImage,3), totScales);
        cellOctaves{octave} = sigma;
        %it is done for 5 blur levels
        for blur_level = 1:totScales
%           %in case of the first blur, in section 3.3 of [2] it states that 
since the original image was pre-smoothed with sigma = 0.5,
%           %"This means that little additional smoothing is needed prior to 
creation of the first octave os scale space". Basically, we know that the 
image is already blurred with
%           %sigma = 1 (0.5 * 2 since it was upscaled) , we have to complete 
the rest of the blur until reaching sigma = 1.6 (initialSigma), which can be 
calculated using the following equation:
%           %sqrt(initialSigma^2 - (2*0.5)^2), this is what I do next in the 
code
%
%           if(octave==1 && blur_level == 1)
%               currentSigma = sqrt(initialSigma^2 - (2*firstBlurSigma)^2);
%           end
            %method used to calculate accum sigmas was taken from 
http://mathworld.wolfram.com/Convolution.html
            if (octave==1 && blur_level == 1)
```

```
                accumSigmas(octave,blur_level) = sqrt(((0.5*2)^2)
+(currentSigma^2));
            elseif (blur_level == 1)
                %TODO: the 3 must be parametrized as round(totScales/2)
%                accumSigmas(octave,blur_level) = sqrt(((accumSigmas(octave-
1,3))^2) ...
%                    +(currentSigma^2));
                accumSigmas(octave,blur_level) = sqrt(((accumSigmas(octave-
1,3)/2)^2) ...
                    +(currentSigma^2));
            else
                accumSigmas(octave,blur_level) =
sqrt((accumSigmas(octave,blur_level-1)^2) ...
                    +(currentSigma^2));
            end
            k = (2^((blur_level)/scales));
%           k = (2^((blur_level)/scales));
%           bluredImage =
gaussianBlur(initialDoubleSizeImage,currentSigma,kernelSize);
            bluredImage =
gaussianBlur(previousDoubleSizeImage,currentSigma,kernelSize);
            previousDoubleSizeImage = bluredImage;
            %disp(['Octave ' num2str(octave) ' blur level '
num2str(blur_level) '  sigma ' num2str(currentSigma)]);
            cellOctaves{octave}(:, :, :, blur_level) = bluredImage;
            currentSigma  = initialSigma * k;
        end
%       cellOctaves{octave} = uint8(cellOctaves{octave});
        currentSigma = initialSigma;
        %in [2] it states to resample two images from the top (totScales-3)
        initialDoubleSizeImage =
reduceInHalf(cellOctaves{octave}(:,:,:,totScales-3));
%imresize(initialDoubleSizeImage, 0.5, 'bilinear');
%reduceInHalf(cellOctaves{octave}(:,:,:,3));
%imresize(initialDoubleSizeImage, 0.5, 'bilinear');
        previousDoubleSizeImage = initialDoubleSizeImage;
    end
    returnData = cell(2,1);
    %code just to check images
%    subplot(1,2,1);
%    imagesc(cellOctaves{4}(:,:,:,2));
%    sigmaknl = accumSigmas(4,2);
%    knl = fspecial('gaussian',[(round(6*sigmaknl)-1) (round(6*sigmaknl)-
1)],sigmaknl);
%    for i = 1:4-1
%        inDSI = reduceInHalf(inDSI);
%    end
%    inDSI = imfilter(inDSI,knl);
%    subplot(1,2,2);
%    imagesc(inDSI);
    %end code to check images
    returnData{1} = cellOctaves;
    returnData{2} = accumSigmas;
    scaleSpace = returnData;
    %As suggested in section 3 of paper [2], the reduction is done by taking
every second pixel
    function reduceInHalf = reduceInHalf(image)
```

```
        reduceInHalf=image(1:2:end,1:2:end) ;
    end
end
```

APPENDIX XV

LOCAL DESCRIPTOR

Code to create a local descriptor i.e. SIFT features

```
function localDescriptor_v3=localDescriptor_v3(orientationDef, genDescriptor,
accumSigmas, maxHeight, maxWidth)
%define some constants
%descriptor width recommended for each of the subregions
DESC_WIDTH = 4;
%number of bins in the histogram in descriptor array
DESC_HIST_BINS = 8;
%descriptor window size
DESC_WIN_SIZE = 16;
    keypoints = 0;
    keypointDescriptor = genDescriptor{1};
    qtyKeypoints = genDescriptor{4};
    keypointDescriptors = cell(size(keypointDescriptor,1),
size(keypointDescriptor,2), maxHeight, maxWidth);
    kptDescriptors =
repmat(struct('octave',0,'kptLayer',0,'kptDescriptor',zeros(4,4,8), ...
                                    'kptX',0,'kptY',0),qtyKeypoints,1)
    %for each of the keypoints, I calculate the orientation, then I rotate
    %the keypoint descriptor accordingly. Finally, calculate the keypoint
    %descriptor.
    cont = 0;
    for octave = 1:size(keypointDescriptor, 1)
        %for each keypoints layer
        for kptLayer = 1:size(keypointDescriptor,2)
            [rowKpt colKpt] = find(keypointDescriptor{octave,kptLayer} == 1);
            if(size(rowKpt,1)==0)
                continue;
            end
            keypointData = orientationDef{1}{octave}{kptLayer};
            magnitudes = orientationDef{2}{octave}{kptLayer}{1};
            orientations = orientationDef{2}{octave}{kptLayer}{2};
            %for each keypoint
            for keypoint = 1:size([rowKpt colKpt],1)
                keypointDetail =
keypointData(rowKpt(keypoint),colKpt(keypoint));
                %for each of the main orientations of the keypoint
                for orient = 1:size(keypointDetail.bestHist,1)
                    kptDescriptor = zeros(128,1);
                    cont = cont+1;
                    keypoints = keypoints+1;
                    degreeInd = orient;
                    %get the degree to rotate
                    degrees = keypointDetail.interpOrien(degreeInd);
                    %the gaussian weights for the window
                    gaussWeight = getGaussWeights(DESC_WIN_SIZE,
DESC_WIN_SIZE/2);
                    %%%%%%%%%gets the coordinates of rotated imate and
rotates the image (of magnitudes)%%%%%%
                    row = rowKpt(keypoint);
                    col = colKpt(keypoint);
```

```
                    v=[row, col]';
                     c=[size(magnitudes,1)/2, size(magnitudes,2)/2]' ;
                    %c=[304.5, 282.5]' ;
                    rotAngle=degrees;
                    rotAngle = 360 - rotAngle;
                    rotMagnitudes= imrotate(magnitudes,rotAngle);
                    %the rotation is also performed for orientations
                    rotOrientations= imrotate(orientations,rotAngle);
                    %this is rotation matrix such as explained by Erik
                    RM=[cosd(rotAngle) -sind(rotAngle)
                           sind(rotAngle) cosd(rotAngle)];
                    temp_v=RM*(v-c);
                    rot_v = temp_v+c;
                    difmat = [(size(rotMagnitudes,1) - size(magnitudes,1))/2,
(size(rotMagnitudes,2) - size(magnitudes,2))/2]';
                    rot_v2 = rot_v + difmat;
                    rotRow = rot_v2(1);
                    rotCol = rot_v2(2);
                    %%%%%%%%%END: gets the coordinates of rotated imate and
rotates the image%%%%%%
                    %the window is 16 x 16 pixels in the keypoint level
                    for x = 0:DESC_WIN_SIZE-1
                        for y = 0:DESC_WIN_SIZE-1
                            %first identify subregion I am in
                            subregAxisX = floor(x/4);
                            subregAxisY = floor(y/4);
                            yCoord = rotRow + y - DESC_WIN_SIZE/2;
                            xCoord = rotCol + x - DESC_WIN_SIZE/2;
                            yCoord = round(yCoord);
                            xCoord = round(xCoord);
                            %get the magnitude
if(yCoord>0&&xCoord>0&&yCoord<=size(rotMagnitudes,1) &&
xCoord<=size(rotMagnitudes,2))
                                magn = rotMagnitudes(yCoord,xCoord);
                                %multiply the magnitude by gaussian weight
                                magn = magn*gaussWeight(y+1,x+1);
                                orientation = rotOrientations(yCoord,xCoord);
                                orientation = orientation + pi;
                                %calculate the respective bucket
                                bucket = (orientation)*(180/pi);
                                bucket = ceil(bucket/45);
                                kptDescriptor((subregAxisY*4+subregAxisX)*8 +
bucket) = ...
kptDescriptor((subregAxisY*4+subregAxisX)*8 + bucket) + magn;
                            end
                        end
                    end
                    %normalize the vector
                    sqKptDescriptor = kptDescriptor.^2;
                    sumSqKptDescriptor = sum(sqKptDescriptor);
                    dem = sqrt(sumSqKptDescriptor);
                    kptDescriptor = kptDescriptor./dem;
                    %threshold
                    kptDescriptor(find(kptDescriptor>0.2))=0.2;
                    %Renormalizing again, as stated in 6.1 of [1]
                    sqKptDescriptor = kptDescriptor.^2;
                    sumSqKptDescriptor = sum(sqKptDescriptor);
```

```
                    dem = sqrt(sumSqKptDescriptor);
                    kptDescriptor = kptDescriptor./dem;
%
keypointDescriptors{octave}{kptLayer}{rowKpt(keypoint)}{rowKpt(keypoint)} =
kptDescriptor;
                    kptDescriptors(cont) =
struct('octave',octave,'kptLayer',kptLayer, ...
                                    'kptDescriptor',kptDescriptor, ...
'kptX',colKpt(keypoint),'kptY',rowKpt(keypoint));
                end
            end
        end
    end
    %keypoints
    %return the keypoint descriptor
    %localDescriptor = keypointDescriptors;
    localDescriptor_v3 = kptDescriptors;
    %function that gets the gaussian weighted window
    function getGaussWeights = getGaussWeights(windowSize, sigma)
        k = fspecial('Gaussian', [windowSize windowSize], sigma);
        k = k.*(1/max(max(k)));
        getGaussWeights = k;
    end
end
```

APPENDIX XVI

PLOT DESCRIPTOR

Plots a particular descriptor, indicating keypoints and orientations.

```
function plotDescriptor = plotDescriptor(descriptor, image, orientationDef,
genDescriptor)
    clf;
%    imagesc(image);
    keypointDescriptor = genDescriptor{1};
    orientations = orientationDef{1};
    %for each keypoints in octave plots the dots
    for octave = 1:size(keypointDescriptor, 1)
        %for each keypoints layer
        for kptLayer = 1:size(keypointDescriptor,2)
            [rowKpt colKpt] = find(keypointDescriptor{octave,kptLayer} == 1);
             for keypoint = 1:size([rowKpt colKpt],1)
                 %plots the dot
                 image = plotDot(image, rowKpt(keypoint), colKpt(keypoint),
octave);
             end
        end
    end
    imagesc(image);
    hold on;
    for octave = 1:size(keypointDescriptor, 1)
        %for each keypoints layer
        for kptLayer = 1:size(keypointDescriptor,2)
            [rowKpt colKpt] = find(keypointDescriptor{octave,kptLayer} == 1);
             for keypoint = 1:size([rowKpt colKpt],1)
                 %plot the arrow with orientation and magnitude starting on
                 %the dot, the length of the line also depends on the
                 %octave in which the keypoint is located
                 plotArrow(rowKpt(keypoint), colKpt(keypoint), octave, ...
        orientations{octave}{kptLayer}(rowKpt(keypoint),colKpt(keypoint)));
             end
        end
    end
    hold off;
    function plotArrow = plotArrow(row,col,octave, keypointDetail)
        %iterates on all the orientations in a particular keypoint and
        %draws them
        for orient = 1:size(keypointDetail.bestHist,1)
            %get the degree to rotate
            degrees = keypointDetail.interpOrien(orient);
            radians = (pi/180)*degrees ;
            magnitude =
keypointDetail.histogram(keypointDetail.bestHist(orient));
            %to see better magnitudes, the small ones are thresholded
            if(magnitude<6)
                magnitude = 6;
            end
            %proportional to the octave in which it was found
            magnitude = magnitude*octave;
            relRow = row;
            relCol = col;
```

```
            if(octave==1)
                relRow = round(row/2);
                relCol = round(col/2);
            end
            if(octave>2)
                relRow = row * (2^(octave-2));
                relCol = col * (2^(octave-2));
            end
            relCol ;
            relRow ;
            colTo = round(relCol + magnitude*cos(radians));
            rowTo = rowTo - relRow;
            h = quiver(relCol,relRow,colTo,rowTo, 'Color','w');
            adjust_quiver_arrowhead_size(h, 7.0);
        end
    end
    function plotDot = plotDot(image, row, col, octave)
        relRow = row;
        relCol = col;
        if(octave==1)
            relRow = round(row/2);
            relCol = round(col/2);
        end
        if(octave>2)
            relRow = row * (2^(octave-2));
            relCol = col * (2^(octave-2));
        end
        if(relRow==1)
            relRow = 2;
        end
        if(relCol==1)
            relCol = 2;
        end
        image(relRow,relCol,1) = 255;
        image(relRow,relCol,2) = 255;
        image(relRow,relCol,3) = 0;
        image(relRow-1,relCol-1,1) = 255;
        image(relRow-1,relCol-1,2) = 255;
        image(relRow-1,relCol-1,3) = 0;
        image(relRow+1,relCol+1,1) = 255;
        image(relRow+1,relCol+1,2) = 255;
        image(relRow+1,relCol+1,3) = 0;
        image(relRow-1,relCol+1,1) = 255;
        image(relRow-1,relCol+1,2) = 255;
        image(relRow-1,relCol+1,3) = 0;
        image(relRow+1,relCol-1,1) = 255;
        image(relRow+1,relCol-1,2) = 255;
        image(relRow+1,relCol-1,3) = 0;
        plotDot = image;
    end
end
```

APPENDIX XVII

PLOT MATCHES

It plots the matches of two images.

```
function plotMatches = plotMatches(image1, image2, matches)
    %first, build image with two images
    heightImage1 = size(image1,1);
    widthImage1 = size(image1,2);
    heightImage2 = size(image2,1);
    widthImage2 = size(image2,2);
    totFrameWidth = widthImage1 + widthImage2;
    if(heightImage1>heightImage2)
        totFrameHeight = heightImage1;
    else
        totFrameHeight = heightImage2;
    end
    combinedImage = ones(totFrameHeight, totFrameWidth,3);
    combinedImage(1:heightImage1, 1:widthImage1,:) = image1;
  combinedImage(1:heightImage2,(widthImage1+1):(widthImage2+widthImage1),:) =
image2;
  imagesc(double(combinedImage)/double(255));
    hold on;
    for match = 1:size(matches,1)
        desc1 = matches(match).descriptorIm1;
        desc2 = matches(match).descriptorIm2;
        octave1 = desc1.octave;
        xPos1 = desc1.kptX;
        yPos1 = desc1.kptY;
        if(octave1==1)
            xPos1 = round(xPos1/2);
            yPos1 = round(yPos1/2);
        end
        if(octave1>2)
            xPos1 = xPos1 * (2^(octave1-2));
            yPos1 = yPos1 * (2^(octave1-2));
        end
        octave2 = desc2.octave;
        xPos2 = desc2.kptX;
        yPos2 = desc2.kptY;
        if(octave2==1)
            xPos2 = round(xPos2/2);
            yPos2 = round(yPos2/2);
        end
        if(octave2>2)
            xPos2 = xPos2 * (2^(octave2-2));
            yPos2 = yPos2 * (2^(octave2-2));
        end
        xPos2 = widthImage1 + xPos2;
        plot([xPos1,xPos2],[yPos1,yPos2]);
    end
    hold off;
end
```

APPENDIX XVIII

GUI

Code to make graphical user interface.

```
function varargout = GUI(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @GUI_OpeningFcn, ...
                   'gui_OutputFcn',  @GUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
% --- Executes just before GUI is made visible.
function GUI_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 GUI (see VARARGIN)
% Choose default command line output for GUI
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
% inimg = imread('inputimage.jpg');
% imshow(inimg,'Parent',handles.axes1);
%h=waitbar(0,'Loading..');
%createbin();
% UIWAIT makes GUI wait for user response (see UIRESUME)
% uiwait(handles.figure1);
% img = imread('1.jpg');
% imshow(img,'Parent',handles.axes1);
%set(handles.pushbutton1,'Visible','off');
drawnow();
%[out_img gender name] = FacialSimilarity(img,Seq,Names,label,d);
set(handles.axes2,'Visible','on');
set(handles.axes3,'Visible','on');
set(handles.axes4,'Visible','on');
% imshow(out_img{1},'Parent',handles.axes2);
% imshow(out_img{2},'Parent',handles.axes3);
% imshow(out_img{3},'Parent',handles.axes4);
% set(handles.text4,'String',gender);
% set(handles.text6,'String',name{1});
% set(handles.text7,'String',name{2});
% set(handles.text8,'String',name{3});
% --- Outputs from this function are returned to the command line.
function varargout = GUI_OutputFcn(hObject, eventdata, handles)
```

```matlab
% 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;
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
 global Seq Names label d
% [file path] = uigetfile('*.jpg');
% if(size(file,2)<=1)
%     return;
% end
s=get(handles.text9,'String');
img = imread(s);
imshow(img,'Parent',handles.axes1);
set(handles.pushbutton1,'Visible','on');
set(handles.text4,'Visible','on');
drawnow();
[out_img gender name] = FacialSimilarity(img,Seq,Names,label,d);
set(handles.axes2,'Visible','on');
set(handles.axes3,'Visible','on');
set(handles.axes4,'Visible','on');
imshow(out_img{1},'Parent',handles.axes2);
imshow(out_img{2},'Parent',handles.axes3);
imshow(out_img{3},'Parent',handles.axes4);
set(handles.text4,'String',gender);
set(handles.text6,'String',name{1});
set(handles.text7,'String',name{2});
set(handles.text8,'String',name{3});
set(handles.pushbutton1,'Visible','on');
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --- Executes during object creation, after setting all properties.
function figure1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Seq Names label d
[file path] = uigetfile('*.jpg');
if(size(file,2)<=1)
    return;
end
img = imread([path file]);
strT = strcat(path,file);
set(handles.text9,'String',strT);
set(handles.text11,'String',file);
imshow(img,'Parent',handles.axes1);
set(handles.pushbutton2,'enable','off');
set(handles.pushbutton3,'enable','on');
% --- Executes on button press in pushbutton3.
```

```
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
faceDetector = vision.CascadeObjectDetector;
s=get(handles.text9,'String');
I = imread(s);
bboxes = step(faceDetector, I);
IFaces = insertObjectAnnotation(I, 'rectangle', bboxes, 'Face');
 figure,   imshow(IFaces), title('Detected faces');
set(handles.pushbutton2,'enable','off');
set(handles.pushbutton3,'enable','off');
set(handles.pushbutton6,'enable','on');
% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
s=get(handles.text9,'String');
I1=imread (s);
figure()
%subplot(3,3,1),imshow(I1),title('Input1'),
[x1,y1,z1]=size(I1);
s1=[reshape(I1,[1,x1*y1*z1])];
S_all=[s1];
S=double(S_all);
Sweight=rand(size(S_all,1));
MixedS=Sweight*S;
ms1=reshape(MixedS(1,:),[x1,y1,z1]);
I1_mixed=uint8(round(ms1));
%subplot(3,3,4),imshow(I1_mixed),title('Mixed1'),
MixedS_bak=MixedS;
MixedS_mean=zeros(3,1);
for i=1:1
    MixedS_mean(i)=mean(MixedS(i,:));
end
for i=1:1
    for j=1:size(MixedS,2)
        MixedS(i,j)=MixedS(i,j)-MixedS_mean(i);
    end
end
MixedS_cov=cov(MixedS');
[E,D]=eig(MixedS_cov);
Q=inv(sqrt(D))*(E)';
MixedS_white=Q*MixedS;
IsI=cov(MixedS_white');
X=MixedS_white;
[VariableNum,SampleNum]=size(X);
numofIC=VariableNum;
B=zeros(numofIC,VariableNum);
for r=1:numofIC
    i=1;maxIterationsNum=150;
    b=2*(rand(numofIC,1)-.5);
    b=b/norm(b);
    while i<=maxIterationsNum+1
       if i == maxIterationsNum
            fprintf('No convergence¡£', r,maxIterationsNum);
```

```
                break;
            end
            bOld=b;
            u=1;
            t=X'*b;
            g=t.^3;
            dg=3*t.^2;
            b=((1-u)*t'*g*b+u*X*g)/SampleNum-mean(dg)*b;
            b=b-B*B'*b;
            b=b/norm(b);
            if abs(abs(b'*bOld)-1)<1e-10
                 B(:,r)=b;
                 break;
             end
            i=i+1;
        end
    end
    ICAedS=B'*Q*MixedS_bak;
    ICAedS_bak=ICAedS;
    ICAedS=abs(55*ICAedS);
    is1=reshape(ICAedS(1,:),[x1,y1,z1]);
    I1_icaed =uint8 (round(is1));
    %subplot(3,3,7),imshow(I1_icaed),title('Restored1');
    reqToolboxes = {'Computer Vision System Toolbox', 'Image Processing
    Toolbox'};
    if( ~checkToolboxes(reqToolboxes) )
     error('detectFaceParts requires: Computer Vision System Toolbox and Image
    Processing Toolbox. Please install these toolboxes.');
    end
    str=get(handles.text9,'String');
    img = imread(str);
    detector = buildDetector();
    [bbox bbimg faces bbfaces] = detectFaceParts(detector,img,2)
    %figure;imshow(bbimg);
    for i=1:size(bbfaces,1)
        set(handles.axes2,'visible','on');
        %figure()
    axes(handles.axes5);
     imshow(bbfaces{i});
    end
    % --- Executes on button press in pushbutton6.
    function pushbutton6_Callback(hObject, eventdata, handles)
    % hObject    handle to pushbutton6 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    s=get(handles.text9,'String');
    image1 = imread(s);
        %image2 = imread('model2.png');
        %define the scale space
        retScaleSpace = scaleSpace(image1,4,3);
        octaveStack = retScaleSpace{1};
        accumSigmas = retScaleSpace{2};
        octaveDOGStack = calculateDog(octaveStack);
        keypoints = calculateKeypoints(octaveDOGStack, image1);
        orientationDef = defineOrientation(keypoints, octaveDOGStack, ...
                   octaveStack, image1, accumSigmas);
        descriptor = localDescriptor_v3(orientationDef, keypoints, ...
```

```
                accumSigmas, size(image1,1)*2, size(image1,2)*2);
            %U=keca(descriptor,1)
%to plot the descriptor, uncomment and comment the rest of the code below
figure,
    plotDescriptor(descriptor, image1, orientationDef, keypoints);
set(handles.pushbutton2,'enable','off');
set(handles.pushbutton3,'enable','off');
set(handles.pushbutton4,'enable','off');
set(handles.pushbutton6,'enable','off');
set(handles.pushbutton7,'enable','on');
% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
s=get(handles.text11,'String');
st=sscanf(s, '%d')
svmclassify2
Dataset = 'TrainDataBase';
m=length(dir(fullfile(Dataset,'*.jpg')));
for m = 1:m
    %img_name = sortedImgs(m);
    img_name = int2str(st);
    str_name = strcat('TrainDataBase\', img_name, '.jpg');
    returnedImage = imread(str_name);
    imshow(returnedImage), title('Recognized Image');
end
set(handles.pushbutton2,'enable','off');
set(handles.pushbutton3,'enable','off');
set(handles.pushbutton4,'enable','off');
set(handles.pushbutton6,'enable','off');
set(handles.pushbutton7,'enable','off');
```

REFERENCES

[1] M. A. Turk, and A. P. Pentland, “Face recognition using eigenfaces,” Journal of cognitive neuroscience, vol. 3, no. 1, pp. 586-591, 1991.

[2] Christopher J.C. Burges, “A Tutorial on Support Vector Machines for Pattern Recognition,” Bell Laboratories, Lucent Technologies, June 1998.

[3] V. Vapnik, “Statistical Learning Theory,” New York: Wiley, Sep. 1998.

[4] W. Feng, Q. He, Y. Yan, G. Jin, and M. Wu, “Real-time human face recognition system with high parallelism,” Proc. of SPIE, vol. 3817, pp. 108-115, 1999.

[5] Issam El-Naqa, Yongyi Yang, Miles N. Wernick, Nikolas P. Galatsanos and Robert M. Nishikawa, “A Support Vector Machine Approach for Detection of Microcalcifications,” IEEE Transactions on Medical Imaging, vol. 21, Dec. 2002.

[6] S. Arca, P. Campadelli, and R. Lanzarotti, “A face recognition system based on local feature analysis,” Proc. of e 4th International Conference on Audio and Video-Based Biometric Person Authentication, pp. 182-189, 2003.

[7] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” Journal of Computer Vision, vol. 60, no. 2, pp. 91-110, 2004.

[8] Y. Ke and R. Sukthankar, “Pca-sift: a more distinctive representation for local image descriptors,”Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp.506-513, 2004.

[9] K. Mikolajczyk, and C. Schmid, “A performance evaluation of local descriptors,” IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 27, no. 10, pp. 1651-1630, 2005.

[10] C. L. Lin, T. C. Chuang, and K. C. Fan, “Palmprint verification using hierarchical decomposition,” Journal of Pattern Recognition, vol. 38, no. 12, pp. 2639-2652, 2005.

[11] M. Bicego, A. Lagorio, E. Grosso, and M. Tistarelli, “On the use of sift features for face authentication,” Proc. of Conference on Computer Vision and Pattern Recognition Workshop, 2006.

[12] L. Zhang, Q. Gao, and D. Zhang, “Block independent component analysis for face recognition,” Journal of The 14th International Conference on Image Analysis and Processing, vol. 27, no. 9, pp. 217-222, 2007.

[13] J. Luo, Y. Ma, E. Takikawa, S. Lao, M. Kawade, and B. L. Lu, "Person-specific sift features for face recognition," Proc. of The IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 593-596, 2007.

[14] D. Zhang, F. Song, Y. Xu, and Z. Liang, Advanced pattern recognition technologies with applications to biometrics, IGI, Hershey, USA, 2008.

[15] D. Colbry, F. Oki, and G. Stockman, "3D face identification: experiments towards a large gallery," Proc. of SPIE, vol. 6944, pp. 694403-1-694403-9, 2008.

[16] Geng Du, Fei Su, Anni Cai, "Face recognition using SURF features," Pattern Recognition and Computer Vision, vol. 7496, Dec. 2009.

[17] Durgesh k. Srivastava, Lekha Bhambhu, "Data Classification Using Support Vector Machine," Journal of Theoretical and Applied Information Technology, Sep. 2009.

[18] M. A. Akhloufi, A. Bendada, and J. C. Batsale, "Multispectral face recognition using non linear dimensionality reduction," Proc. of International Society for Optical Engineering, pp. 73410J-1-73410J-10, 2009.

[19] C. Geng, "Face recognition using sift features," Proc. of the 16th IEEE International Conference on Image Processing, pp. 3313-3316, 2009.

[20] A. Majumdar, and R. K. Ward, "Discriminative sift features for face recognition," Proc. of Canadian Conference on Electrical and Computer Engineering, pp. 27-30, 2009.

[21] B. Dai, D. Zhang, H. Liu, S. Sun, and K. Li, "Evaluation of face recognition techniques," Proc. of SPIE, vol. 7489, pp. 74890M-1-74890M-7, 2009.

[22] C. Geng, and X. Jiang, "Face recognition using sift features," Proc. of the 16th IEEE International Conference on Image Processing, pp. 3277-3280, 2009.

[23] Y. Shi, D. Q. Dai, C. C. Liu, and H. Yan, "Sparse discriminant analysis for breast cancer biomarker identification and classification," Journal of Progress in Natural Science, vol. 19, no. 11, pp. 1635-1641, 2009.

[24] E. Elhamifar, and R. Vidal, "Sparse subspace clustering," Proc. of IEEE International Conference on Computer Vision and Pattern Recognition, pp. 2790-2797, 2009.

[25] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma, "Robust face recognition via sparse representation," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 31, no. 2, pp. 210-227, 2009.

[26] Y. Xu, A. Zhong, J Yang, and David Zhangn, “Lpp solution schemes for use with face recognition,” Journal of Pattern Recognition, vol. 43, no. 12, pp. 4165-4176, 2010.

[27] X. Li, A. Li, and X. Bai, “3D face detection and face recognition: state of the art and trends,” Proc. of International Society for Optical Engineering, pp. 78201E-1-78201E-7, 2010.

[28] J. Kriˇzaj, V. ˇStruc and N. Paveˇsi´c, “Adaptation of sift features for robust face recognition,” Proc. Of The 7th international conference on Image Analysis and Recognition, pp. 394-404, 2010.

[29] Z. Lai, Z. Jin, J. Yang, and W. K. Wong, “Sparse local discriminant projections for feature extraction,” Proc. of the 20th International Conference on Pattern Recognition, pp. 926-929, 2010.

[30] Y. Xu, D. Zhang, and J. Y. Yang, “A feature extraction method for use with bimodal biometrics,” Journal of Pattern Recognition, Vol. 43, no. 3, pp. 1106-1115, 2010.

[31] J. Wright, Y. Ma, J. Mairal, G. Sapiro, T. Huang, and S. Yan, “Sparse representation for computer vision and pattern recognition,” Proc. of IEEE, vol. 98, no. 6, pp. 1031-1044, 2010.

[32] Y. Xu, A. Zhong, J. Yang, and D. Zhanget, “Bimodal biometrics based on a representation and recognition approach,” Journal of Optical Engineering, vol. 50, no. 3, pp. 037202-1-037202-7, 2011.

[33] Wang, H., Yang, Gao, F. and Li, “Normalization Methods if SIFT Vector for Object Recognition,” Tenth International Symposium on Distributed Computing and Applications to Business Engineering and Science, pp. 175-178, Oct. 2011.

[34] Y. Xu, Q. Zhu, and D. Zhang, “Combine crossing matching scores with conventional matching scores for bimodal biometrics and face and palmprint recognition experiments,” Journal of Neurocomputing,vol. 74, no. 18, pp. 3946-3952, 2011.

[35] H. F. Wang, Y. Han, and Z. X. Zhang, “Applying local gabor ternary pattern for video-based illumination variable face recognition,” Proc. of International Conference on Machine Vision, pp.83490W-1-83490W-6, 2011.

[36] C. Y. Chang, C. W. Chang, and C. Y. Hsieh, “Applications of block linear discriminant analysis for face recognition,” Journal of Information Hiding and Multimedia Signal Processing, vol. 2, no. 3, pp.259-269, 2011.

[37] D. Y. Huang, C. J. Lin, and W. C. Hu, “Learning-based face detection by adaptive switching of skin color models and AdaBoost under Varying Illumination,” Journal of Information Hiding and Multimedia Signal Processing, vol. 2, no. 3, pp. 204-216, 2011.

[38] J. Wang, Q. Li, J. You, and Q. Zhao, “Fast kernel fisher discriminant analysis via approximating the kernel principal component analysis,” Journal of Neurocomputing, vol. 74, no. 17, pp. 3313V3322, 2011.

[39] L. Zhang, M. Yang, and X. Feng, “Sparse representation or collaborative representation: which helps face recognition,” Proc. of IEEE International Conference on Computer Vision, pp. 471-478, 2011.

[40] K. Vaishnavi, and G. P. R. Kumar, “Face recognition using passion back propagation neural networks,” Journal of Computing Technology and Information Security, vol. 1, no. 2, pp. 9-15, 2011.

[41] A. Mian, “Online learning from local features for video-based face recognition,” Journal of Pattern Recognition, vol. 44, no. 5, pp. 1068-1075, 2011.

[42] Y. Xu, D. Zhang, J. Yang, and J. Y. Yang, “A two-phase test sample sparse representation method for use with face recognition,” IEEE Trans. Circuits and Systems for Video Technology, vol. 21, no.9, pp. 1255-1262, 2011.

[43] Y. Xu, Z. Fan, and Q. Zhu, “Feature space-based human face image representation and recognition,” Journal of Optical Engineering, vol. 51, no. 1, pp. 017205-1-017205-7, 2012.

[44] Hirdesh Kumar, Padmavati, “Face Recognition using SIFT by varying Distance Calculation Matching Method,” International Journal of Computer Applications, vol. 47, no. 3, June 2012.

[45] Hung-Fu Huang and Shen-Chuan Tai, “Facial Expression Recognition Using New Feature Extraction Algorithm,” Electronic Letters on Computer Vision and Image Analysis, ISSN 1577-5097, Sept. 2012.

[46] Y. Xu, W. Zuo, and Z. Fan, “Supervised sparse representation method with a heuristic strategy and face recognition experiments,” Journal of Neurocomputing, vol. 79, no. 1, pp. 125-131, 2012.

[47] M. K. Hsu, C. Hsu, T. N. Lee, and H. Szu, “Face recognition from a moving platform via sparse representation,” Proc. of SPIE, vol. 8401, pp. 840106-1-840106-6, 2012.

[48] D. Y. Huang, C. H. Chen, W. C. Hu, S. C. Yi, and Y. F. Lin, "Feature-based vehicle flow analysis and measurement for a real-time traffic surveillance system," Journal of Information Hiding and Multimedia Signal Processing, vol. 3, no. 3, pp. 279-294, 2012.

[49] Y. Xu, and Q. Zhu, A simple and fast representation-based face recognition method, Springer, London, UK, 2012.

[50] Yu-Yao Wang, Zheng-Ming Li, Min Wang, Long Wang, "A Scale Invariant Feature Transform Based Method", Journal of Information Hiding and Multimedia Signal Processing, vol. 4, no. 2, April 2013.

[51] Patrik Kamencay, Martina Zachariasova, Robert Hudec, Roman Jarina, Miroslav Benco, Jan Hlubik, "A Novel Approach to Face Recognition using Image Segmentation Based on SPCA-KNN Method," Radio Engineering, vol. 22, no. 1, April 2013.

[52] Mr. Amit Kr. Gautam, Ms. Twisha, "Improved Face Recognition Technique using Sift," Journal of Electrical and Electronics Engineering, pp. 72-76, March 2014.

[53] G. Srividhya, Ms. B. Vijaya Lakshmi, "Face Recognition of Different Modalities Using SIFT and LBP Features," International Journal of Innovative Research in Science, Engineering and Technology, issue 1, ISSN (online) 2319 – 8753, ISSN (print) 2347 – 6710, vol. 3, Feb. 2014.

[54] Isra'a Abdul-Ameer Abdul-Jabbar, Jieqing Tan, and Zhengfeng Hou, "Adaptive PCA-SIFT Matching Approach for Face Recognition Application," International MultiConference of Engineers and Computer Scientists, vol. 1, March 2014.

[55] Trasha Gupta, Lokesh Garg, "Face Recognition Using SIFT," International Journal of Emerging Technology and Advanced Engineering, issue 5, vol. 4, May 2014.

[56] A. Martinez, and R. Benavente, "The ar face database," 1998, http://www2.ece.ohio-state.edu/~aleix/ARdatabase.html.

[57] P. J. Phillips, "The facial recognition technology (feret) database," http://www.itl.nist.gov/iad/humanid/feret/feret_master.html.

[58] Ajitkumar Deshmukh, Pritam Sirpotdar, Juber Sheikh, Prof. Dr. M. A. Joshi, "High accuracy Face Recognition system based on SIFT," International Journal of Advanced Research in Computer Engineering & Technology, issue 6, vol. 4 , May 2015.

LIST OF PUBLICATIONS

[1] Kirti Bagla and Bharat Bhushan, "A novel approach for face recognition using hybrid SIFT-SVM", presented in IEEE 1st Internatinal Conference on Power Electronis, Intelligent control and Energy Systems, ICPEICES-2016, Delhi, to be published on IEEE Explore.