# IMAGE BASED SECURITY SYSTEM

**071330**

**HARSHUL SINGHAL**

**DR. RAJESH SIDDAVATAM**


**May – 2011**

**Submitted in partial fulfillment of the Degree of Bachelor of Technology**

**DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY**
**JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY**
**WAKNAGHAT**

# TABLE OF CONTENTS

# CERTIFICATE

This is to certify that the work titled **IMAGE BASED SECURITY SYSTEM** submitted by **HARSHUL SINGHAL** in partial fulfillment for the award of degree of Bachelor of Technology (C.S.E.) of **JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT** has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature in full of Supervisor: _____

Name in Capital block letters: **Dr. RAJESH SIDDAVATAM**

Designation: **Associate Professor, Department of Computer Science and IT, JUIT**

Date: **19th May, 2011**

# ACKNOWLEDGEMENT

No venture can be completed without the blessing of Almighty. I consider it my bounded duty to bow to Almighty whose kind blessings always inspire us on the right path of life This project is our combined effort and realizes the potential of team and gives me a chance to work in co-ordination.

Science has caused many frontiers so has human efforts towards human research. My revered guide **Dr. Rajesh Siddavatam,** Associate Professor, Department of Computer Science and IT, JUIT, has indeed acted as a light house showing me the need of sustained effort in the field of Image Processing to learn more and more. So i take this apportunity to thank him, for lending me stimulating suggestions, innovative quality guidance and creative thinking. He provides me the kind of strategies required for the completion of a task. I am grateful to him for the support , he provided me in doing thinks at my pace and for being patient for my mistakes.

Signature of the student _____

Name of Student ___Harshul Singhal__

Date ___19th May, 2011___

# SUMMARY

The project entitled as **IMAGE BASED SECURITY SYSTEM** is a system developed for detecting and recognizing human faces from the images degraded with noise, face recognition algorithms .This security system is one proposed system which requires no human interaction with any sort of high-tech equipment as in the case of fingerprint, signature, thumb impression etc. .

Principal Component Analysis, Linear Discriminant Analysis and Successive Mean Quantization Transform have demonstrated their success in face detection, recognition, and tracking. These subspace methods are based on second order statistics of the image set.We have implemented these algorithms and compared the performance to find the algorithm best suited for our application.

The basic idea behind this project is to provide a good, efficient method for detecting human faces and recognizing them with the database provided.

| | |
|---|---|
| _____ | _____ |
| Signature of Student | Signature of Supervisor |
| Name: Harshul Singhal | Name: Dr. Rajesh Siddavatam |
| Date: 19th May, 2011 | Date: 19th May, 2011 |

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF ABBREVIATIONS

| | | |
|---|---|---|
| The successive mean Quantization transform | – | **SMQT** |
| Mean quantization units | – | **MQUs** |
| Mean of Least Variance | – | **MLV** |
| Mean of Coefficient of Variation | – | **MCV** |

# Chapter 1

# 1. INTRODUCTION

IMAGE BASED SECURITY SYSTEM is a system that removes noise from a noisy image using Median Filter and successively performs Face Detection , to isolate and recognize human faces from that image .The research work on face detection started and face recognition started in 1960 and is still going on. Many algorithms have been implemented for the same but these vary widely in accuracy and efficiency.

The first tedious task is to understand all the relevant algorithms avalaible and to choose the best out of those algorithms because the performance of the system depends on the complexity of these algorithms.

The system includes:

- The subject standing in front of the camera.
- The image captured is first cleaned using a denoising algorithm.
- Persons face being detected from the scene depending upon the intensity and background of the scene.
- The face detected is then centered and consequently matched with the existing images in the database to recognize the object.
- If the subject gets recognized his access is authenticated else not.

## 1.1 SCOPE

The system is mainly useful for higher officials and faculty members to access a particular location that is confidential and has a restricted access.

## 1.2 PROBLEM STATEMENT

To develop a system that removes noise from a noisy image using filters and successively performs face detection and recognize human faces from that image.The system can thus be used in face authenticated systems wherein problems of noise in images are common due to malfunctioning of camera or due to hardware limitations.

## 1.3 METHODOLOGY

For Denoising we have used the following algorithm

- Adaptive Median Filter

For Face Detection we have used the following algorithm.

- Face Detection using local SMQT (Successive Mean Quantization Transform) features.

The face detection algorithm is the combined with the best face recognition algorithm to achieve the desired objective. All these algorithms have different complexities and different efficiencies when tested with the standard databases avalaible.

# Chapter 2

# 2. HISTORY

Many of the techniques of digital image processing, or digital picture processing as it often was called, were developed in the 1960s at the Jet Propulsion Laboratory, Massachusetts Institute of Technology, Bell Laboratories, University of Maryland, and a few other research facilities, with application to satellite imagery, wire-photo standards conversion, medical imaging, videophone, character recognition and photograph enhancement. The cost of processing was fairly high, however, with the computing equipment of that era. That changed in the 1970s, when digital image processing proliferated as cheaper computers and dedicated hardware became available. Images then could be processed in real time, for some dedicated problems such as television standards conversion. As general-purpose computers became faster, they started to take over the role of dedicated hardware for all but the most specialized and compute-intensive operations.

With the fast computers and signal processors available in the 2000s, digital image processing has become the most common form of image processing and generally, is used because it is not only the most versatile method, but also the cheapest. Digital image processing technology for medical applications was inducted into the Space Foundation Space Technology Hall of Fame in 1994. Image processing allows the use of much more complex algorithms, and hence, can offer both more sophisticated performance at simple tasks, and the implementation of methods which would be impossible by analog means.

# Chapter 3

# 3. EXPLANATION OF EACH MODULE

- Image Denoising using Adaptive Median Filter.
- Face Detection using loacal SMQT (Successive Mean Quantization Transform) features.

# 3.1 IMAGE NOISE

Image noise is the random variation of brightness or color information in images produced by the sensor and circuitry of a scanner or digital camera. There are numerous types of noises as stated below.

## 3.1.1 NOISE TYPES

### 3.1.1.1 Amplifier Noise (Gaussian noise) :

The standard model of amplifier noise is additive, Gaussian, independent at each pixel and independent of the signal intensity, caused primarily by thermal noise. In color cameras where more amplification is used in the blue color channel than in the green or red channel, there can be more noise in the blue channel.
Amplifier noise is a major part of the “read noise” of an image sensor.

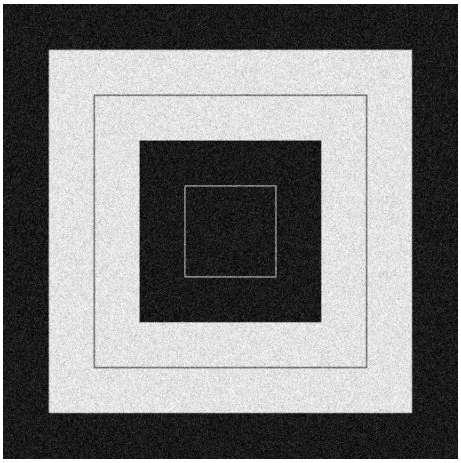

**Fig. 1:** 512x512-Gaussian-Noise

**Fig. 2:** Original Image

**Fig. 3:** Image with Gaussian noise

### 3.1.1.2 SALT AND PEPPER NOISE :

Impulsive noise is sometimes called salt and pepper noise or spike noise.

An image containing salt and pepper noise will have dark pixels in bright regions and bright pixels in dark regions. This type of noise can be caused by dead pixels, analog-to-digital converter errors, bit errors in transmission, etc.

**Fig. 4:** Noise salt and pepper


**Fig. 5:** Salt and Pepper noise

### 3.1.1.3 Shot Noise :

The dominant noise in the lighter parts of an image from an image sensor is typically that caused by statistical quantum fluctuations, that is, variation in the number of photons sensed at a given exposure level; this noise is known as photon shot noise. Shot noise has a root-mean-square value proportional to the square root of the image intensity, and the noises at different pixels are independent of one another. Shot noise follows a Poisson distribution, which is usually not very different from Gaussian.

### 3.1.1.4 Quantization Noise :

The noise caused by quantizing the pixels of a sensed image to a number of discrete levels is known as quantization noise; it has an approximately uniform distribution, and can be signal dependent, though it will be signal independent if other noise sources are big enough to cause dithering, or if dithering is explicitly applied.

### 3.1.2 Noise problems with digital cameras :

In low light, correct exposure requires the use of long shutter speeds, higher gain (ISO sensitivity), or both. On most cameras, longer shutter speeds lead to increased salt-and-pepper noise due to photodiode leakage currents. At the cost of a doubling of read noise variance (41% increase in read noise standard deviation), this salt-and-pepper noise can be mostly eliminated by dark frame subtraction. Banding noise, similar to shadow noise, can be introduced through brightening shadows or through color-balance processing.

The relative effect of both read noise and shot noise increase as the exposure is reduced, corresponding to increased ISO sensitivity, since fewer photons are counted (shot noise) and since more amplification of the signal is necessary.

**Fig. 6:** Image on the left has exposure time of >10 seconds in low light. The image on the right has adequate lighting and 0.1 second exposure.

## 3.2 Filters

### 3.2.1 Mean Filter

The mean filter is a simple sliding-window spatial filter that replaces the center value in the window with the average (mean) of all the pixel values in the window. The window, or kernel, is usually square but can be any shape. An example of mean filtering of a single 3x3 window of values is shown below.

| unfiltered values | | |
|---|---|---|
| 5 | 3 | 6 |
| 2 | 1 | 9 |
| 8 | 4 | 7 |

**Table 1:** Unfiltered values

| mean filtered | | |
|---|---|---|
| * | * | * |
| * | 5 | * |
| * | * | * |

**Table 2:** mean filtered

5 + 3 + 6 + 2 + 1 + 9 + 8 + 4 + 7 = 45

45 / 9 = 5.

Center value (previously 1) is replaced by the mean of all nine values (5).

### 3.2.2 Median filter

The median filter is also a sliding-window spatial filter, but it replaces the center value in the window with the median of all the pixel values in the window. As for the mean filter, the kernel is usually square but can be any shape. An example of median filtering of a single 3x3 window of values is shown below.

| unfiltered values | | |
|---|---|---|
| 6 | 2 | 0 |
| 3 | 97 | 4 |
| 19 | 3 | 10 |

**Table 3:** Unfiltered values

in order: 0, 2, 3, 3, 4, 6, 10, 15, 97

.

| median filtered | | |
|---|---|---|
| * | * | * |
| * | 4 | * |
| * | * | * |

**Table 4:** median filtered

Center value (previously 97) is replaced by the median of all nine values (4).

Note that for the first (top) example, the median filter would also return a value of 5, since the ordered values are 1, 2, 3, 4, 5, 6, 7, 8, 9. For the second (bottom) example, though, the mean filter returns the value 16 since the sum of the nine values in the window is 144 and 144 / 9 = 16. This illustrates one of the celebrated features of the median filter: its ability to remove 'impulse' noise (outlying values, either high or low). The median filter is also widely claimed to be 'edge-preserving' since it theoretically preserves step edges without blurring. However, in the presence of noise it does blur edges in images slightly.

### 3.2.3 MLV and MCV filters

The Mean of Least Variance (MLV) filter is an edge-preserving (actually edge-enhancing) noise smoothing filter based on the concepts of mathematical morphology. The idea is to smooth an image only over areas that have a low variance - that is, areas that are most nearly homogeneous and do not contain edges or other features. The MCV filter is similar, except that it uses the Coefficient of Variation rather than the variance to perform better on multiplicative noise.

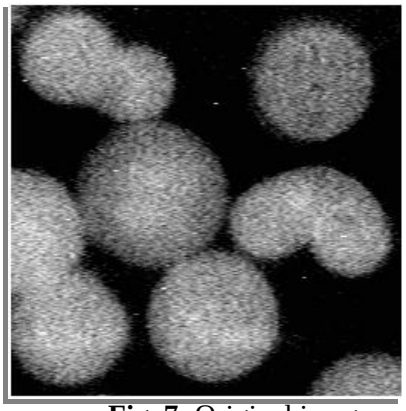

**Fig. 7:** Original image

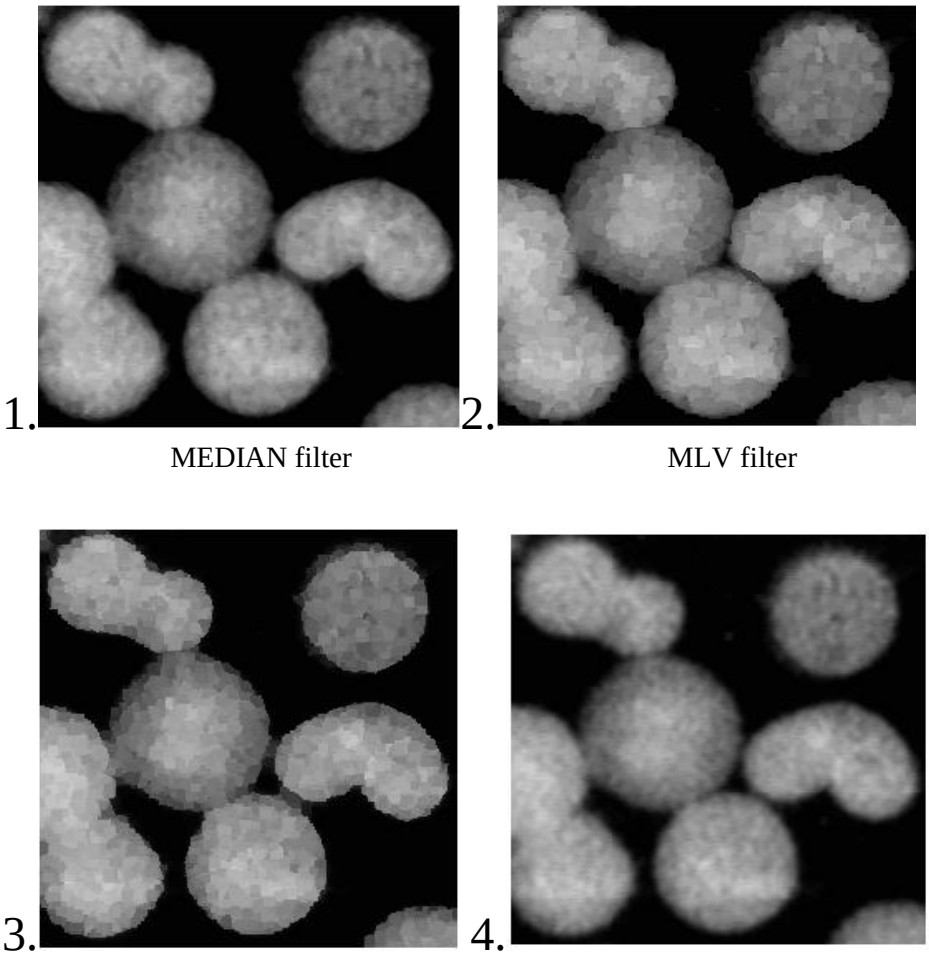


**Fig. 8:** An applet implementing several spatial image filters
Filter Kernel size :5*5
Image of cells viewed through a fluorescence microscope(256*256)

### 3.2.4 Adaptive Median Filter

The Adaptive Median Filter is designed to eliminate the problems faced with the standard median filter. The basic difference between the two filters is that, in the Adaptive Median Filter, the size of the window surrounding each pixel is variable. This variation depends on the median of the pixels in the present window. If the median value is an impulse, then the size of the window is expanded. Otherwise, further processing is done on the part of the image within the current window specifications. 'Processing' the image basically entails the following: The center pixel of the window is evaluated to verify whether it is an impulse or not. If it is an impulse, then the new value of that pixel in the filtered image

will be the median value of the pixels in that window. If, however, the center pixel is not an impulse, then the value of the center pixel is retained in the filtered image.
Thus, unless the pixel being considered is an impulse, the gray-scale value of the pixel in the filtered image is the same as that of the input image. Thus, the Adaptive Median Filter solves the dual purpose of removing the impulse noise from the image and reducing distortion in the image. Adaptive Median Filtering can handle the filtering operation of an image corrupted with impulse noise of probability greater than 0.2.
This filter also smoothens out other types of noise, thus, giving a much better output image than the standard median filter.

**Fig. 9:** Results of filtering with a 5*X*5 median and conditional median filter. From left to right, first row : original Image, noisy image.
second row : standard median filter, Adaptive median filter.

## 3.3 LOCAL SMQT FEATURES

## SMQT

The Successive Mean Quantization Transform (SMQT) reveals the organization or structure of the data and removes properties such as gain and bais. The transform is described and applied in image processing. The SMQT is considered as an extra processing step for mel frequency cepstral coefficients commonly used in speech recognition. In image processing the transform is applied in automatic image enhancement and dynamic range compression.

The aim with the SMQT is to remove the disparity between the sensors due to gain and bais. Additionallly, the extraction of the structure of the data should be done in an efficient manner. This structure extraction problem can be seen as the problem of dynamic range compression. The finding of structures in data has been proposed before which extracts a binary structure from an image. More recently the modified Census Transform emerged; this transform has similarities to a first level SMQT. However, these techniques reveal only one bit structures or structure kernels. The SMQT can be used to extend the structure representation to an arbitrary dimensional data. This will be shown by applying the SMQT in both speech and image processing.

The SMQT uses an approach that performs an automatic structural breakdown of information. This operation can be seen as a progressive focus on the details in an image.

SMQT in image processing involves transforming a whole image at different levels gives an illustrative description of the operation for the SMQT.

The level L in the SMQTL denotes the number of bits use to describe the transformed image. Hence, SMQT1 of the image has a 1 bit representation(0,1) and SMQT2 of the image has 2 bits(0,1,2,3).Choosing a level of the transform lower than the number of bits in the original image yield a dynamic range compressed image. A SMQT8 of an image, which has dynamic range represented by 8 bits, will yield an uncompressed image with

enhanced details. A comparison with histogram equalization is conducted. The histogram equalization has some problems with oversaturation and artifacts in several area in the images.

Notice how the histogram equalized images have a tendency to get washed out or unnatural. These effects do not occur, or are very limited, in the SMQT enhanced images. The SMQT also has less computational complexity and fewer adjustments compared to more advanced enhancement techniques.

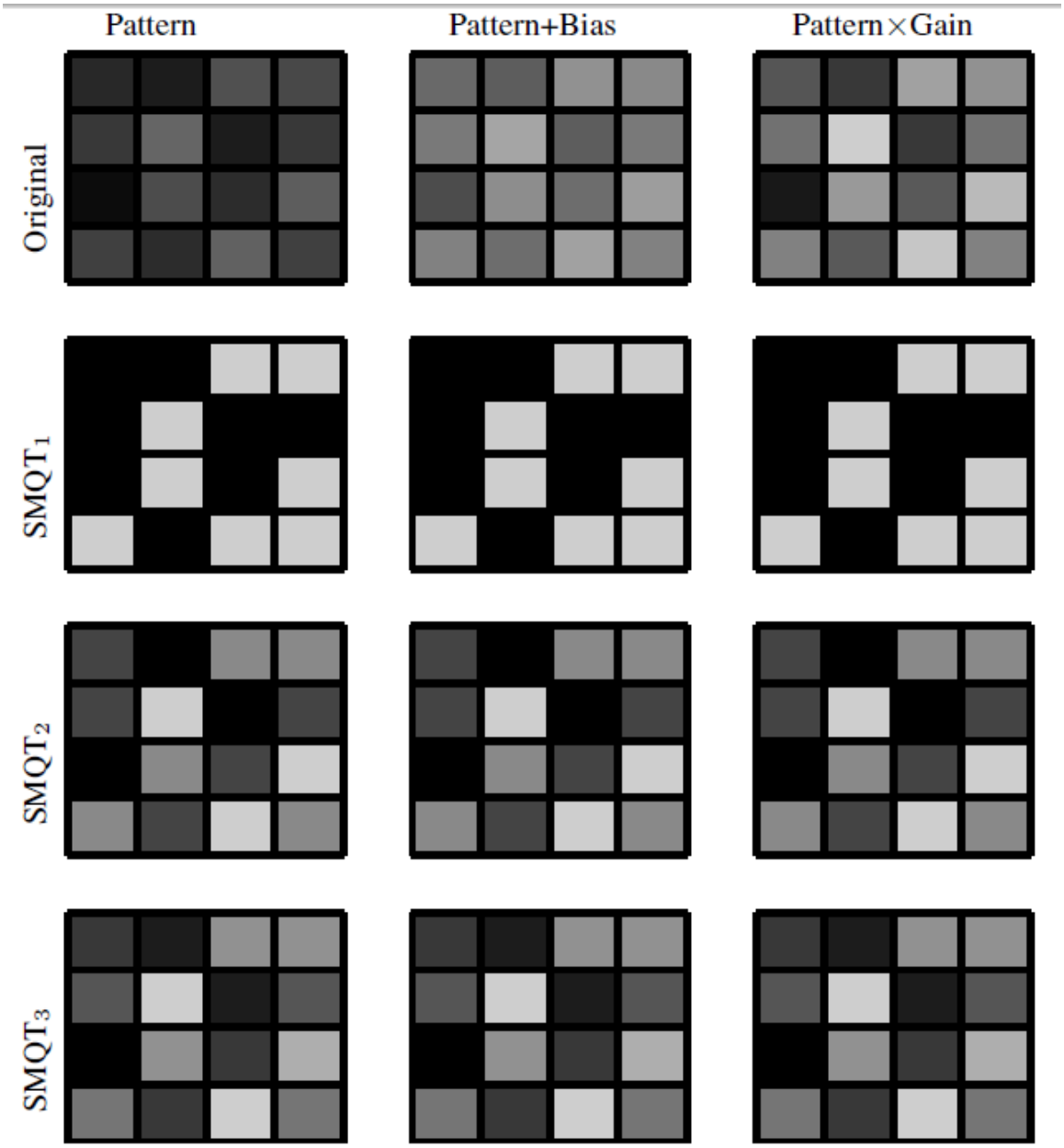


**Fig. 10:** Example 4 × 4 local area patterns and SMQT results.

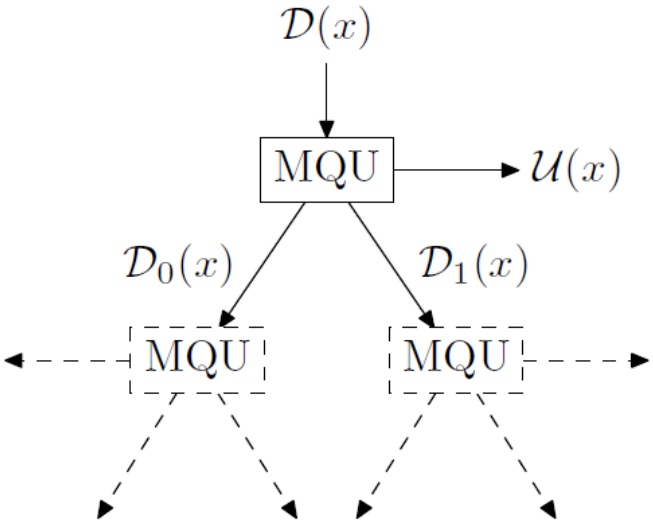


**Fig. 11:** The operation of one Mean Quantization Unit

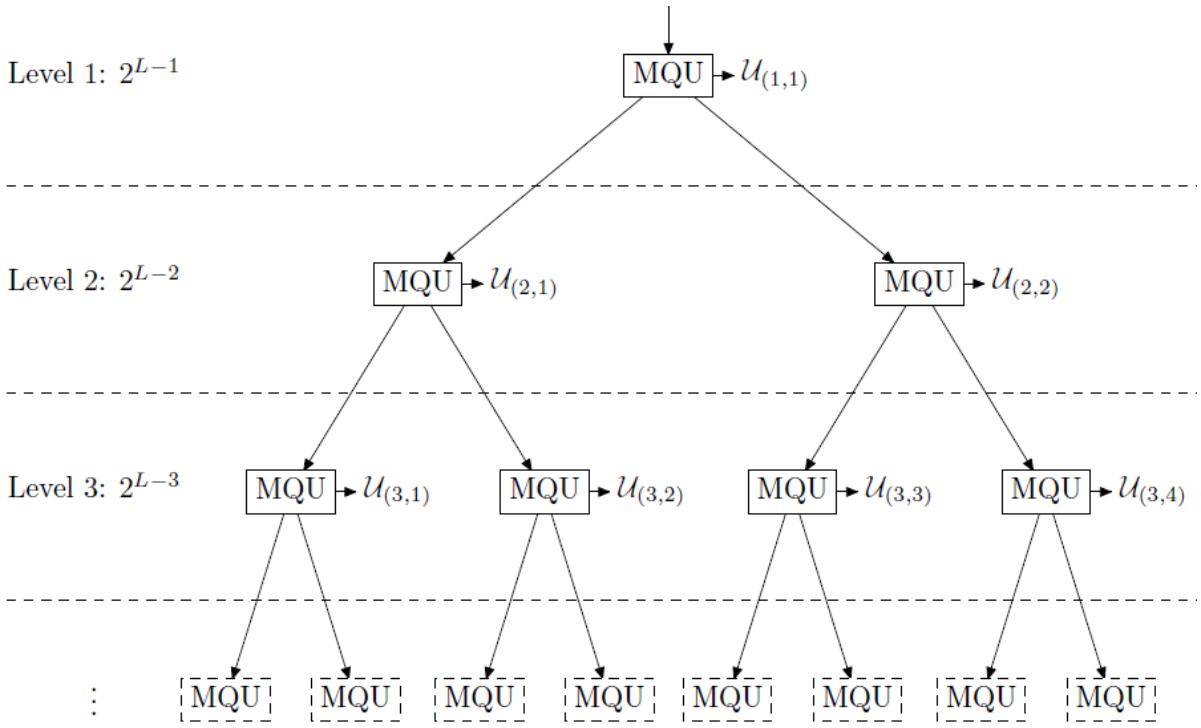


**Fig. 12:** The Successive Mean Quantization Transform (SMQT) as a binary tree of Mean Quantization Units (MQUs).

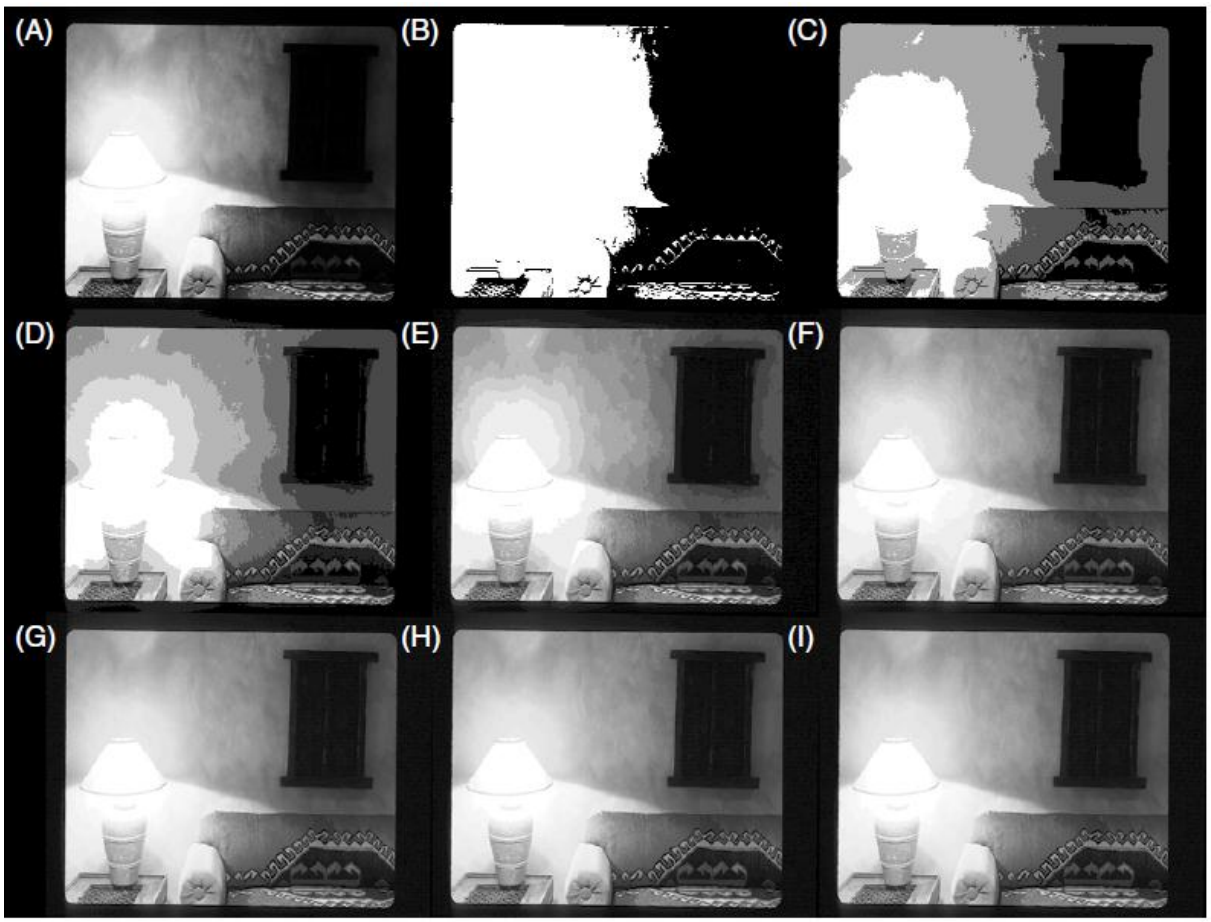


**Fig. 13:** Example of SMQT on whole image. (A) original image, dynamic range 0-255 (8 bit). (B) - (I) corresponds to SMQT1− SMQT8 of image.

# Chapter 4

# 4.1 HOW TO DETECT A FACE USING FACE DETECTOR

As mentioned above, the first stage in Face Recognition is Face Detection. The library makes it fairly easy to detect a frontal face in an image using its Haar Cascade Face Detector (also known as the Viola-Jones method).

The function "cvHaarDetectObjects" performs the actual face detection, but the function is a bit tedious to use directly, so it is easiest to use this wrapper function – Appendix B. Now you can simply call "detectFaceInImage" whenever you want to find a face in an image. You also need to specify the face classifier that to detect the face. For example, detector comes with several different classifiers for frontal face detection, as well as some profile faces (side view), eye detection, nose detection, mouth detection, whole body detection, etc. You can actually use this function with any of these other detectors if you want, or even create your own custom detector such as for car or person detection , but since frontal face detection is the only one that is very reliable.

For frontal face detection, you can chose one of these Haar Cascade Classifiers that come with detector (in the "data\haarcascades\" folder):
"haarcascade_frontalface_default.xml"
"haarcascade_frontalface_alt.xml"
"haarcascade_frontalface_alt2.xml"
"haarcascade_frontalface_alt_tree.xml"
Each one will give slightly different results depending on your environment, so you could even use all of them and combine the results together (if you want the most detections).

## 4.2 HOW TO PREPROCESS FACIAL IMAGES FOR FACE RECOGNITION

Now that you have detected a face, you can use that face image for Face Recognition. However, if you tried to simply perform face recognition directly on a normal photo image, you will probably get less than 10% accuracy!

It is extremely important to apply various image pre-processing techniques to standardize the images that you supply to a face recognition system. Most face recognition algorithms are extremely sensitive to lighting conditions, so that if it was trained to recognize a person when they are in a dark room, it probably won't recognize them in a bright room, etc. This problem is referred to as "lumination dependent", and there are also many other issues, such as the face should also be in a very consistent position within the images (such as the eyes being in the same pixel coordinates), consistent size, rotation angle, hair and makeup, emotion (smiling, angry, etc), position of lights (to the left or above, etc). This is why it is so important to use a good image preprocessing filters before applying face recognition. The pixels around the face that aren't used, such as with an elliptical mask to only show the inner face region, not the hair and image background, since they change more than the face does, should be removed.

For simplicity, the face recognition system, will show you is Eigenfaces using greyscale images. So it will show you how to easily convert color images to greyscale (also called 'grayscale'), and then easily apply Histogram Equalization as a very simple method of automatically standardizing the brightness and contrast of your facial images. For better results, you could use color face recognition (ideally with color histogram fitting in HSV or another color space instead of RGB), or apply more processing stages such as edge enhancement, contour detection, motion detection, etc. Also, this code is resizing images to a standard size, but this might change the aspect ratio of the face.

Here you can see an example of this preprocessing stage:

Here is some basic code to convert from a RGB or greyscale input image to a greyscale image, resize to a consistent dimension, then apply Histogram Equalization for consistent brightness and contrast: Appendix B

## 4.3 HOW EIGENFACES CAN BE USED FOR FACE RECOGNITION

Now that we have a pre-processed facial image, we can perform Eigenfaces (PCA) for Face Recognition. Software comes with the function "cvEigenDecomposite()", which performs the PCA operation, however we need a database (training set) of images for it to know how to recognize each of your people.

So we should collect a group of preprocessed facial images of each person we want to recognize. For example, if we want to recognize someone from a class of 10 students, then we could store 20 photos of each person, for a total of 200 preprocessed facial images of the same size (say 100x100 pixels).

I will attempt to explain the theory of Eigenfaces .

Use "Principal Component Analysis" to convert all 200 training images into a set of "Eigenfaces" that represent the main differences between the training images. First it will find the "average face image" of the images by getting the mean value of each pixel. Then the eigenfaces are calculated in comparison to this average face, where the first eigenface is the most dominant face differences, and the second eigenface is the second most dominant face differences, and so on, until you have about 50 eigenfaces that represent most of the differences in all the training set images.


**Fig. 14:** Eigen faces

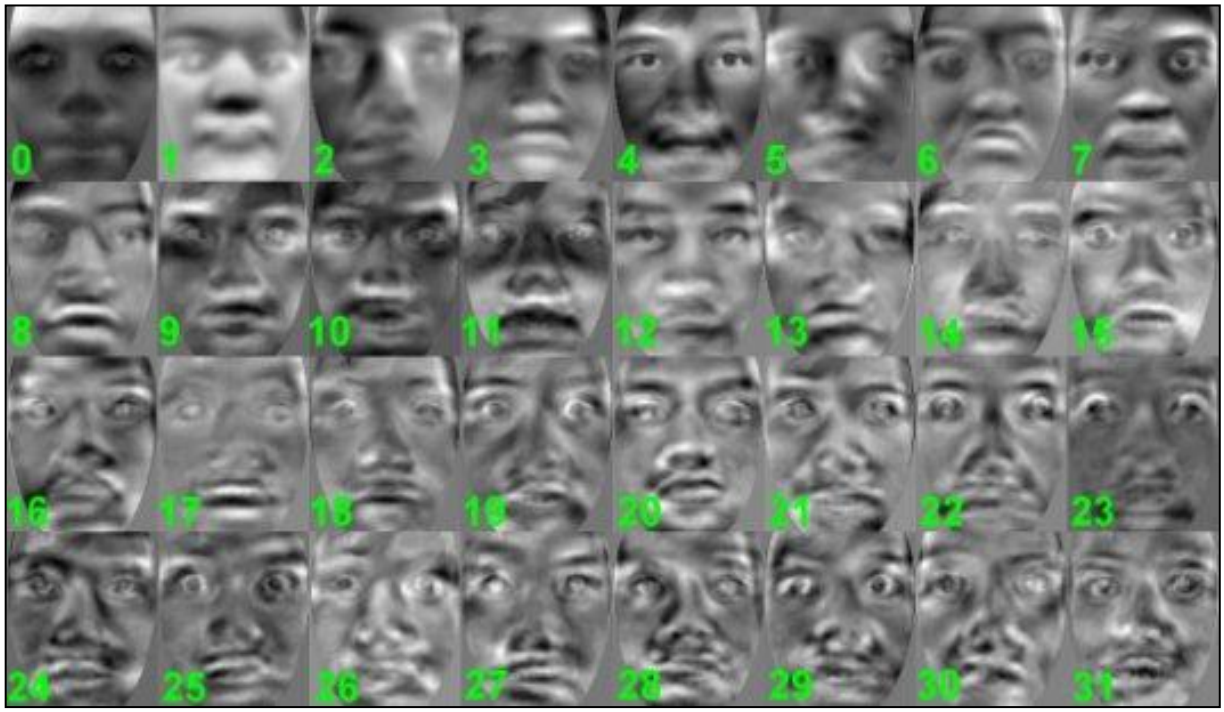


**Fig. 15:** set of training images

In these example images above you can see the average face and the first and last eigenfaces that were generated from a collection of 30 images each of 4 people. Notice

that the average face will show the smooth face structure of a generic person, the first few eigenfaces will show some dominant features of faces, and the last eigenfaces (eg: Eigenface 119) are mainly image noise. We can see the first 32 eigenfaces in the image above.

# 4.4 EXPLANATION OF FACE RECOGNITION USING PRINCIPAL COMPONENT ANALYSIS

To explain Eigenfaces (Principal Component Analysis) in simple terms, Eigenfaces figures out the main differences between all the training images, and then how to represent each training image using a combination of those differences.

So for example, one of the training images might be made up of:
(averageFace) + (13.5% of eigenface0) - (34.3% of eigenface1) + (4.7% of eigenface2) + ... + (0.0% of eigenface199).
Once it has figured this out, it can think of that training image as the 200 ratios:
{13.5, -34.3, 4.7, ..., 0.0}.
It is indeed possible to generate the training image back from the 200 ratios by multiplying the ratios with the eigenface images, and adding the average face. But since many of the last eigenfaces will be image noise or wont contribute much to the image, this list of ratios can be reduced to just the most dominant ones, such as the first 30 numbers, without effecting the image quality much. So now its possible to represent all 200 training images using just 30 eigenface images, the average face image, and a list of 30 ratios for each of the 200 training images.
Interestingly, this means that we have found a way to compress the 200 images into just 31 images plus a bit of extra data, without loosing much image quality.
To recognize a person in a new image, it can apply the same PCA calculations to find 200 ratios for representing the input image using the same 200 eigenfaces. And once again it can just keep the first 30 ratios and ignore the rest as they are less important. It can then search through its list of ratios for each of its 20 known people in its database, to see who has their top 30 ratios that are most similar to the 30 ratios for the input image. This is basically a method of checking which training image is most similar to the input image, out of the whole 200 training images that were supplied.

# 4.5 IMPLEMENTING FACE RECOGNITION

For implementation of face recognition, where files are used as input and output through the command-line. Basically, to create a facerec database from training images, we create a text file that lists the image files and which person each image file represents.

For example, you could put this into a text file called "4_images_of_2_people.txt":

1 Shervin data\Shervin\Shervin1.bmp

1 Shervin data\Shervin\Shervin2.bmp

1 Shervin data\Shervin\Shervin3.bmp

1 Shervin data\Shervin\Shervin4.bmp

2 Chandan data\Chandan\Chandan1.bmp

2 Chandan data\Chandan\Chandan2.bmp

2 Chandan data\Chandan\Chandan3.bmp

2 Chandan data\Chandan\Chandan4.bmp

This will tell the program that person 1 is named "Shervin", and the 4 preprocessed facial photos of Shervin are in the "data\Shervin" folder, and person 2 is called "Chandan" with 4 images in the "data\Chandan" folder. The program can then load them all into an array of images using the function "loadFaceImgArray()". Note that for simplicity, it doesn't allow spaces or special characters in the person's name, so you might want to enable this, or replace spaces in a person's name with underscores (such as Shervin_Emami).

To create the database from these loaded images, we use OpenCV's "cvCalcEigenObjects()" and "cvEigenDecomposite()" functions, eg:

```
// Tell PCA to quit when it has enough eigenfaces.
CvTermCriteria calcLimit = cvTermCriteria( CV_TERMCRIT_ITER, nEigens, 1);
// Compute average image, eigenvectors (eigenfaces) and eigenvalues (ratios).
cvCalcEigenObjects(nTrainFaces, (void*)faceImgArr, (void*)eigenVectArr,
        CV_EIGOBJ_NO_CALLBACK, 0, 0, &calcLimit,
        pAvgTrainImg, eigenValMat->data.fl);
// Normalize the matrix of eigenvalues.
cvNormalize(eigenValMat, eigenValMat, 1, 0, CV_L1, 0);

// Project each training image onto the PCA subspace.
CvMat projectedTrainFaceMat = cvCreateMat( nTrainFaces, nEigens, CV_32FC1 );
int offset = projectedTrainFaceMat->step / sizeof(float);
for(int i=0; i<nTrainFaces; i++) {
        cvEigenDecomposite(faceImgArr[i], nEigens, eigenVectArr, 0, 0,
                pAvgTrainImg, projectedTrainFaceMat->data.fl + i*offset);
}
```

We now have:

- the average image "pAvgTrainImg",
- the array of eigenface images "eigenVectArr[]" (eg: 200 eigenfaces if you used nEigens=200 training images),
- the matrix of eigenvalues (eigenface ratios) "projectedTrainFaceMat" of each training image.

These can now be stored into a file, which will be the face recognition database. The function "storeTrainingData()" in the code will store this data into the file "facedata.xml", which can be reloaded anytime to recognize people that it has been trained for. There is also a function "storeEigenfaceImages()" in the code, to generate the images shown earlier, of the average face image to "out_averageImage.bmp" and eigenfaces to "out_eigenfaces.bmp".

For implementation of the face recognition stage, where the face recognition system will try to recognize who is the face in several photos from a list in a text file. The same sort of text file lists the images that should be tested, as well as the correct person in that image. The program can then try to recognize who is in each photo, and check the correct value in the input file to see whether it was correct or not, for generating statistics of its own accuracy.

The implementation of the face recognition is as follows:

1. The list of image files (preprocessed faces) and names are loaded into an array of images, from the text file that is now used for recognition testing (instead of training). This is performed in code by "loadFaceImgArray()".
2. The average face, eigenfaces and eigenvalues (ratios) are loaded from the face recognition database file "facedata.xml", by the function "loadTrainingData()".
3. Each input image is projected onto the PCA subspace using the software function "cvEigenDecomposite()", to see what ratio of eigenfaces is best for representing this input image.

But now that it has the eigenvalues (ratios of eigenface images) to represent the input image, it looks for the original training image that had the most similar ratios. This is done mathematically in the function "findNearestNeighbor()" using the "Euclidean Distance", but basically it checks how similar the input image is to each training image, and finds the most similar one, the one with the least distance in Euclidean Space. The

distance between the input image and most similar training image is used to determine the "confidence" value, to be used as a guide of whether someone was actually recognized or not. A confidence of 1.0 would mean a good match, and a confidence of 0.0 or negative would mean a bad match. But beware that the confidence formula I use in the code is just a very basic confidence metric that isn't necessarily too reliable, but I figured that most people would like to see a rough confidence value. You may find that it gives misleading values for your images and so you can disable it if you want (eg: set the confidence always to 1.0).

Once it knows which training image is most similar to the input image, and assuming the confidence value is not too low (it should be atleast 0.6 or higher), then it has figured out who that person is, in other words, it has recognized that person!

# 4.6 HOW TO USE THE REALTIME WEBCAM FACE-RECOGNITION SYSTEM

If you have a webcam plugged in, then you should be able to test this program by just double-clicking the EXE file in Windows (or compile the code and run it if you are using Linux or Mac). Hit the Escape key on the GUI window when you want to quit the program.

After a few seconds it should show the camera image, with the detected face highlighted. But at first it wont have anyone in its face rec database, so you will need to create it by entering a few keys. Beware that to use the keyboard, you have to click on the DOS console window before typing anything (because if the software window is highlighted then the code won't know what you typed).

In the console window, hit the **'n'** key on your keyboard when a person is ready for training. This will add a new person to the facerec database. Type in the person's name (without any spaces) and hit Enter.

It will begin to automatically store all the processed frontal faces that it sees. Get a person to move their head around a bit until it has stored about 20 faces of them. (The facial images are stored as PGM files in the "data" folder, and their names are appended to the text file "train.txt").

Get the person in front of the camera to move around a little and move their face a little, so that there will be some variance in the training images.

Then when you have enough detected faces for that person, ideally more than 30 for each person, hit the **'t'** key in the console window to begin training on the images that were just collected. It will then pause for about 5-30 seconds (depending on how many faces and people are in the database), and finally continue once it has retrained with the extra person. The database file is called "facedata.xml".

It should print the person's name in the console whenever it recognizes them.

Repeat this again from step 1 whenever you want to add a new person, even after you have shutdown the program.

If you hit the Escape key in the console (or GUI) then it will shutdown safely. You can then run the program again later and it should already be trained to recognize anyone that was added.

Depending on the speed of your computer and camera, it might be recording faces too fast for you (ie: when you click 'n', it saves 100 faces when you only want about 20 faces). So you might want to modify the code to just run at about 1 frame per second (adjust the "cvWaitKey(10)" code to something like "cvWaitKey(1000)").

Note that the code currently doesn't let you delete a previous person or add more training images to an existing known person, etc. These features aren't too hard to add (deleting a person means deleting the image files and lines from the training file, adding more images to an existing person requires keeping track of how many images where already loaded for that person). But there will always be more features that could be added!

# Chapter 5

# SOFTWARE REQUIREMENTS SPECIFICATION

Ultimately the requirement phase translates the ideas whatever is in the mind of client (the input) into a formal document (the output of the requirement phase.). In a more general way the SRS is a document that completely describes "What" the proposed system should do without describing "How" the software will do it.

## FEASIBILITY STUDY

The feasibility study concerns with the consideration made to verify whether the system fit to be developed in all terms. Once an idea to develop software is put forward the question that arises first will pertain to the feasibility aspects.

There are different aspects in the feasibility study:

- Operational Feasibility.
- Technical Feasibility.
- Economical Feasibility.

## OPERATIONAL FEASIBILITY:

There in no difficulty in implementing the system, if the user has the knowledge in internal working of the system. Therefore, it is assumed that he will not face any problem in running the system. The main problem faced during development of a new system is getting acceptance from the users. As users are responsible for initiating the development of a new system this is rooted out.

## TECHNICAL FEASIBILITY:

Technical feasibility deals with the study of function, performance, and constraints like resources availability, technology, development risk that may affect the ability to achieve an acceptable system.

## ECONOMICAL FEASIBILITY:

One of the factors, which affect the development of a new system, is the cost it would incur. The existing resources available in the company are sufficient for implementing the proposed and hence no extra cost has to be incurred to run the system developed. Thus, the system is financially feasible.

# Chapter 6

# SYSTEM REQUIREMENTS

**HARDWARE REQUIREMENTS ( MININUM)**

CPU : 1.6 GHz

RAM : 384 MB

DISPLAY : 1024*768 Monitor

Hard Disk : 8 GB

Webcam : **Type** Color

- **Still Image** 160 x 120,
  176 x 144,
  320 x 240,
  352 x 288

**SOFTWARE REQUIREMENTS**

OS : Windows XP/Vista/7/Linux

Software : Visual Basic 4 or above (can use to compile)

**Portable :Exexcutable**

**Coding : VB and assembly language**

# Chapter 7

# MODEL USED

To develop software a particular development strategy is used which encompasses the process, methods and tools. The strategy is often referred to as process model of Software Engineering Paradigm. A process model for software is chosen based in the nature of the project and application, the methods and tools to be used and the controls and deliverables that are required. The model which is being used in this project development is WATERFALL MODEL.

## WATERFALL MODEL

The waterfall model derives its name due to the cascading effect from one phase to the other as is illustrated in Figure1.1. In this model each phase well defined starting and ending point, with identifiable deliveries to the next phase.

Note that this model is sometimes referred to as the linear sequential model or the software life cycle. The model consists of six distinct stages, namely:

- **REQUIREMENTS ANALYSIS**

In the requirements analysis phase:
(a) The problem is specified along with the desired service objectives (goals).
(b) The constraints are identified.

- **SPECIFICATION PHASE**

In the specification phase the system specification is produced from the detailed definitions of (a) and (b) above. This document should clearly define the product function.
Note that in some text, the requirements analysis and specifications phases are combined and represented as a single phase.

- ## SYSTEM AND SOFTWARE DESIGN PHASE

In the system and software design phase, the system specifications are translated into a software representation. The software engineer at this stage is concerned with:

**a.** Data structure
**b.** Software architecture
**c.** Algorithmic detail and
**d.** Interface representations

The hardware requirements are also determined at this stage along with a picture of the overall system architecture. By the end of this stage the software engineer should be able to identify the relationship between the hardware, software and the associated interfaces. Any faults in the specification should ideally not be passed 'down stream'.

- ## IMPLEMENTATION AND TESTING PHASE

In the implementation and testing phase stage the designs are translated into the software domain:

- Detailed documentation from the design phase can significantly reduce the coding effort.
- Testing at this stage focuses on making sure that any errors are identified and that the software meets its required specification.

- ## INTEGRATION AND SYSTEM TESTING PHASE

In the integration and system testing phase all the program units are integrated and tested to ensure that the complete system meets the software requirements. After this stage the software is delivered to the customer [Deliverable – The software product is delivered to the client for acceptance testing.]

- **MAINTENANCE PHASE**

The maintenance phase the usually the longest stage of the software. In this phase the software is updated to:

- Meet the changing customer needs
- Adapted to accommodate changes in the external environment
- Correct errors and oversights previously undetected in the testing phases
- Enhancing the efficiency of the software

Observe that feed back loops allow for corrections to be incorporated into the model. For example a problem/update in the design phase requires a 'revisit' to the specifications phase. When changes are made at any phase, the relevant documentation should be updated to reflect that change.

# CONCLUSION

Many different techniques exist and continue to be developed, while the algorithms to detect faces also advances quickly. The adaptive median filter successfully removes impulsive noise from images. It does a reasonably good job of smoothening images that contain non-impulsive noise. Local SMQT features were found to be able to cope with illumination and sensor variation in object detection.We expect that a combination of these approaches that uses soft assignment of each pixel to an object class and one or more change models would produce a very general, powerful,and theoretically well-justified detection algorithm.

We expect future algorithm developments to be fueled by increasingly more integrated approaches combining elaborate models of change, implicit pre- and post-processing, robust statistics and global optimization methods.

# FUTURE SCOPE OF WORK

Future enhancements recommended are :

1. Consumption of time can be reduced.
2. The size of the image can be reduced.
3. A well standard algorithm can be implemented.
4. User Interface can still be made better.

# REFERENCES


1) Rafael C. Gonzalez and Richard E. Woods, *Digital Image Processing*. Addison-Wesley, 1992., markschulze.
2) Ch.Ravi Kumar, S.K. Srivathsa, *EHW Architecture for Design of Adaptive Median Filter for Noise Reduction*.
3) D.Dhanasekaran, K.Boopathy Bagan, *High Speed Pipelined Architecture for Adaptive Median Filter*.
4) B. Froba and A. Ernst, "Face detection with the modified census transform," in *Sixth IEEE International Conference on Automatic Face and Gesture Recognition*, May 2004, pp. 91–96.
5) Mikael Nilsson, Mattias Dahl, and Ingvar Claesson, *The successive mean quantization transform.*
6) Mikael Nilsson, J¨orgen Nordberg, and Ingvar Claesson, *face detection using local smqt features and split up snowclassifier.*

# APPENDIX – A

**Mixing Assembly language with Visual Basic**

**What you need:**

- **Visual Basic** (I have only tried with VB5, but VB4 or above should work). If you don’t have Visual Basic, then you can get the "Express Edition" (was called the "Custom Control Edition") for free at the Microsoft site!
- **NASM** (I have only tried with v0.97). If you don’t have NASM, then you can get it for free at the NASM site (http://www.nasm.us/).
- A simple text editor. You can use Notepad, but I recommend something better like PFE, Scite or Notepad++ .

**Using NASM with VB:**

To use NASM with Visual Basic, you have to make a DLL with NASM, and then you can use that DLL with Visual Basic (VB3 doesn't let you use dll's). You can use the DLL's you made, the same way as if you where using a Windows system DLL, such as 'user32.dll' or 'gdi32.dll'.

To make a DLL with NASM, there are three steps to it:
You first need to write your NASM code
Then you must compile your NASM code and link it with Visual Basic's linker.
Then you can use it with VB.

# APPENDIX – B

## CODING /ALGORITHMS

The adaptive filter works on a rectangular region **Sxy**. The adaptive median filter changes the size of **Sxy**
during the filtering operation depending on certain criteria as listed below. The output of the filter is a
single value which the replaces the current pixel value at (x, y), the point on which *Sxy* is centered at the
time. The following notation is adapted from the book and is reintroduced here:
**Zmin** = Minimum gray level value in **Sxy**.
**Zmax** = Maximum gray level value in **Sxy**
**Zmed** = Median of gray levels in **Sxy**
**Zxy** = gray level at coordinates **(x, y)**
**Smax** = Maximum allowed size of **Sxy**
**The adaptive median filter works in two levels denoted Level A and Level B as follows:**
Level A: A1= **Zmed - Zmin**
A2= **Zmed - Zmax**
If A1 > 0 AND A2 < 0, Go to level B
Else increase the window size
If window size <= **Smax** repeat level A
**Else output Zxy**.
Level B: B1 = **Zxy – Zmin**
B2 = **Zxy – Zmin**
If B1 > 0 And B2 < 0 output Zxy
Else output Zmed.
The algorithm has three main purposes:
To remove ‘Salt and Pepper’ noise
To smoothen any non impulsive noise
To reduce excessive distortions such as too much thinning or thickening of object boundaries.

The function "cvHaarDetectObjects" in software performs the actual face detection, but the function is a bit tedious to use directly, so it is easiest to use this wrapper function:

```
// Perform face detection on the input image, using the given Haar Cascade.

// Returns a rectangle for the detected region in the given image.

CvRect detectFaceInImage(IplImage *inputImg, CvHaarClassifierCascade* cascade)

{

        // Smallest face size.

        CvSize minFeatureSize = cvSize(20, 20);

        // Only search for 1 face.

        int flags = CV_HAAR_FIND_BIGGEST_OBJECT |
CV_HAAR_DO_ROUGH_SEARCH;

        // How detailed should the search be.

        float search_scale_factor = 1.1f;

        IplImage *detectImg;

        IplImage *greyImg = 0;

        CvMemStorage* storage;

        CvRect rc;

        double t;

        CvSeq* rects;

        CvSize size;

        int i, ms, nFaces;


        storage = cvCreateMemStorage(0);

        cvClearMemStorage( storage );
```

```
// If the image is color, use a greyscale copy of the image.
detectImg = (IplImage*)inputImg;
if (inputImg->nChannels > 1) {
        size = cvSize(inputImg->width, inputImg->height);
        greyImg = cvCreateImage(size, IPL_DEPTH_8U, 1 );
        cvCvtColor( inputImg, greyImg, CV_BGR2GRAY );
        detectImg = greyImg; // Use the greyscale image.
}

// Detect all the faces in the greyscale image.
t = (double)cvGetTickCount();
rects = cvHaarDetectObjects( detectImg, cascade, storage,
                search_scale_factor, 3, flags, minFeatureSize);
t = (double)cvGetTickCount() - t;
ms = cvRound( t / ((double)cvGetTickFrequency() * 1000.0) );
nFaces = rects->total;
printf("Face Detection took %d ms and found %d objects\n", ms, nFaces);

// Get the first detected face (the biggest).
if (nFaces > 0)
        rc = *(CvRect*)cvGetSeqElem( rects, 0 );
else
        rc = cvRect(-1,-1,-1,-1);       // Couldn't find the face.

if (greyImg)
```

```
        cvReleaseImage( &greyImg );

    cvReleaseMemStorage( &storage );

    //cvReleaseHaarClassifierCascade( &cascade );


    return rc;      // Return the biggest face found, or (-1,-1,-1,-1).

}
```

Here is some basic code to convert from a RGB or greyscale input image to a greyscale image, resize to a consistent dimension, then apply Histogram Equalization for consistent brightness and contrast:

```
// Either convert the image to greyscale, or use the existing greyscale image.
IplImage *imageGrey;
if (imageSrc->nChannels == 3) {
    imageGrey = cvCreateImage( cvGetSize(imageSrc), IPL_DEPTH_8U, 1 );
    // Convert from RGB (actually it is BGR) to Greyscale.
    cvCvtColor( imageSrc, imageGrey, CV_BGR2GRAY );
}
else {
    // Just use the input image, since it is already Greyscale.
    imageGrey = imageSrc;
}

// Resize the image to be a consistent size, even if the aspect ratio changes.
IplImage *imageProcessed;
imageProcessed = cvCreateImage(cvSize(width, height), IPL_DEPTH_8U, 1);
// Make the image a fixed size.
// CV_INTER_CUBIC or CV_INTER_LINEAR is good for enlarging, and
// CV_INTER_AREA is good for shrinking / decimation, but bad at enlarging.
cvResize(imageGrey, imageProcessed, CV_INTER_LINEAR);

// Give the image a standard brightness and contrast.
cvEqualizeHist(imageProcessed, imageProcessed);

.....  Use 'imageProcessed' for Face Recognition ....
```

```
if (imageGrey)
        cvReleaseImage(&imageGrey);
if (imageProcessed)
        cvReleaseImage(&imageProcessed);
```

# PUBLICATION

Research paper titled as "**Turing Machine for i-head Hydra**" is published with IEEE and indexed at IEEE Xplore, at *International Conference on Computer Modelling and Simulation*, UKSim `10, Cambridge , 24 – 26 March 2010 .

# BRIEF BIO-DATA (RESUME)

## ACADEMIC QUALIFICATIONS

| Standard | College/School | Year | CGPA/Percentage |
|---|---|---|---|
| B-Tech (CSE) | Jaypee University of Information Technology, Solan. | 2011 | C.G.P.A. = 5.8 (equivalent to 65.45% upto 7th semester) |
| 12th (CBSE) | St Andrews Public School, Agra. | 2007 | 75.6% |
| 10th (ICSE) | St. Peter's College, Agra | 2005 | 69.66% |

## TECHNICAL SKILLS

**Languages:** C, C++, Java (Core), J2EE
**Database Management Concepts:** MySQL 5.0
**Operating Systems:** Microsoft Windows, Linux (Ubuntu 9.04).
**Scripting Languages:** HTML, Java Server Pages (JSP), PHP.
**Tools:** Adobe Photoshop, Turbo C++ IDE, Matlab 7.1, Net Beans 6.7.1, Adobe Flash 9, Eclipse.

## INDUSTRIAL TRAINING

Industrial training at **Bharat Sanchar Nigam Limited** (BSNL), Agra, July 2010.
**Project:** Made a project on Broadband Database Management System of Agra. It manages the data of BSNL connections and used as database for further reference.
**Technology Used**: J2EE 5.0, SQL 4.0

## I.T CERTIFICATIONS

- **(SCWCD) SUN MICROSYSTEMS** certified Web Component Developer for J2EE 1.5 (81%).
- **(SCJP) SUN MICROSYSTEMS** certified Java Programmer for standard platform 1.5 (80%).
- **DB2-9 Fundamentals IBM** certified Database Associate (70.31%).

## RESEARCH PUBLICATIONS

NITIN, **Harshul Singhal**, Rohan Kundra, "**Turing Machine For i-Head Hydra**" in 12th International conference on Computer Modeling and Simulation, (UKSim), Emmanuel College, Cambridge, England, 21st March 2010.

## TECHNICAL PROJECTS

- Final Year project (under development): **Image Based Security System using** J2SE 5.0, JMF1.1, J2EE 5.0. It's a system that removes noise from a noisy image using filters and successively performs face detection and recognizes human faces from that image.
- Developed a project **SafePad** using J2SE 5.0 under the course of Software Testing & Debugging, an alternate to Microsoft Notepad (Team: 4).
- Developed a project **Online Forum** using HTML, JSP, PHP, SQL 5.0 web components under the course of Web-Technology (Team: 3).
- Developed a project **Traffic Simulation** using HTML, J2SE 5.0 for controlling the traffic.

## PERSONAL ACHIEVEMENTS

- Awarded "**Outstanding student Volunteer Award"** 2010, for contribution towards IEEE student activities at student branch and section level.
- **Webmaster**: IEEE student branch of JUIT for the session 2009-10.
- Represented College in **AGM (ANNUAL GENERAL MEETING OF IEEE)** 2010 held at IIT-Delhi.
- Represented JUIT at "**IEEE-XTREME 09**", a 24 hour programming competition at international level.
- Developed an official Web-site of JUIT Student Branch of I.E.E.E
  Web-site: http://ewh.ieee.org/sb/delhi/juit, http://juit.ac.in/ieee
- Awarded 1st prize under event "**Hack**" in **Le-Fiestus 2009** (annual college fest).
- Won 2nd prize under event **"Codez"** in **Murious 4.0 `09** (annual college Tech-Fest).

## WORKSHOPS AND EXTRA CURRICULAR ACTIVITIES

- Attended a two day workshop organized by **IBM** on Eclipse, WAS sphere, DB2.
- Attended a two day workshop in **Robo Beam** held at Thapar University, Patiala on 4th-5th October, 08.
- Member of the **Program Committee (08-09)**, **Web Team** and **Automation Wing (09-10)** of JUIT Branch of IEEE.
- Organized IEEE Fest named **Konkurrenz** in college in the year 2010.
- Attended Seminar on **Robotics MoRETA, Cyber security and Computer Forensics** by Insparc Industry Professionals at Jaypee University of Information Technology, Solan.

## HOBBIES AND OTHER INTERESTS

- Playing Cricket
- Automaton
- Listening music