University Of Zambia

School Of Natural Sciences

Department Of Computer Science

Research Project: Assisted and Augmented Reading (AAR)

Student Name : Tobiah Mwima

Computer Number: 10029788

Course Code **:** CSC4000

Submitted in partial fulfillment for the requirements for a bachelor's Degree in Computer Science (major in Software Engineering)

Supervisor **:** Mr. David Chisanga

Co-Supervisor: Mr. Samson Chibuta

Declaration of Originality

I hereby declare that this thesis is my

Original work except where stated. _____ _____

Date Signature

Abstract

The Mouse and keyboard is the major means of passing information from user to computer. Direct manipulation of objects via the mouse was a breakthrough in the design of a more natural and intuitive user interfaces for computers. However, in real life we have a rich set of communication methods at our disposal; when interacting with others, we, for example interpret their gestures, expressions and eye movements. This information can be used also when moving human –computer interaction toward the more natural and effective method. In particular, the eye gesture can be a more valuable source of information for a computer system to be aware of, if it has to provide assistance when appropriate.

The focus of this research is on examining how the information acquired from a user's eye movements in human-to-computer interaction can be used to assist electronic book readers. For this purpose a simple prototype called assisted and augmented reading will be developed. Enhancing the reading experience and awarding electronic book readers some reward points, which will be dependent on the tracked reading progress; this will be an encouraging and motivating technique to help increase literacy levels. However, finding the point of focus on the screen by eye tracking using the ordinary webcam is the main hypothesis behind the development of this prototype. The implementation of this prototype will be founded on a C++ library called Open Computer Vision and IDE for C++.

Acknowledgements

I would like to thank Mr David Chisanga my supervisor for his valuable support and guidance he gave me before he left for his PhD. This project would not have reached this far without his initial guidance and support.

I would like to acknowledge the help that came from my co-supervisor Mr Samson Chibuta, for the valuable guidance he provided in the absence of my supervisor.

I would also like to thank all the lecturers who have taught me in one or more courses from the Department of Computer Science for their generous support and guidance they gave me throughout the course of my programme and for giving me the opportunity do this research project.

Finally but not the least I am grateful to my Dad, Mom, Brothers and Sisters and relatives for their continuous encouragement and support they offered, and not forgetting the great part, I am grateful to the Lord for this life.

Dedication

This work is given to the Lord.

Table of Contents

List of Figure

Chapter 1 Introduction

1.1 Project Background.

In Zambia and Africa at large there have been an ongoing common saying ‘hiding a thing from a black man put it in a Book’ [22], this saying has a profound truth by white men that realizes the true place of a black man in relation to literacy . According to [25] literacy is the ability to read, write and understand. With reference to [23], Zambia National and Information Services reports that Mkushi District Education Board Secretary during an interview revealed the occurrence of low levels of literacy in Zambia as a result of no innovative methods by the education system of reducing illiteracy. If Zambia’s literacy levels are not addressed with effective methods development will be restricted socially, academically, financially and health wise [22]. This research study recognizes effective ways that will motivate, encourage and evaluate the reading attitude of people for the betterment of Zambian literacy levels. This will be developed through a proper understanding of eye movement.

However, eye movements of the user during his or her daily work in front of the computer, can be used to deduce user’s intentions [21], and provide assistance where helpful. Gaze can be seen as a substitution for user’s attention, and eye movements are known to be usually closely tied with mental processes in the brain, so a great deal about those processes can be observed by eye tracking. For example, by interpreting eye movements reading behavior of user can be detected, which most likely entails mental processes of understanding with regard to the currently read word [21].

In this research, the focus is to develop and implement a reading detection algorithm that is gaze aware to motivate, encourage and enhance the reading experience. Since reading, is probably the most common activity; done by most people sitting in front of the computer screen with or without knowledge that it is improving their literacy levels [21]. Hence eye tracking to monitor reading behavior is of great necessity.

1.2 Research Problem

This research study investigates the ability to create a prototype of a gaze aware system that assists and augments reading electronic documents using cheap hardware but with enough efficiency to be useful. Since a large number of computer users already have webcams and they are relatively cheap to buy, the hardware to be used will be a regular webcam. This is expected to provide motivation, encouragement and improve literacy for users through an enhanced reading environment.

1.3 Objectives and Aims

The final goal of this research is to find out for sure whether it is possible to create a prototype that takes gaze input by using regular webcam to provide motivation, assistance and augment reading. By knowing the point of focus the prototype should

I. manage to provide word meaning for some defined words as necessary

II. Moving to the next page, checking comments, checking added notes or clicking on an available link within the document should be done using gaze

III. Keep track of the reading progress of the user by indicating the last word read when the user stopped reading.

IV. And lastly reward user profiles award points based on the detected reading performance as a motivation technique

V. make the prototype easy to use and self-explained.

1.4 Outline/Expected Results

The design of the prototype will take advantage of Open Computer vision functions and other self-defined algorithms, which will help, implement the following modules to help achieve the objectives.

1.4.1 Facial recognition Module

This module will employ an algorithm that detects and recognizes faces and maps it to a particular user profile. The essence of this module is in making it possible for the prototype to be able to identify and rate a user profile.

1.4.2 Gaze with HCI Module

The focus of this function is fixation and saccades detection, to track eye movements of users as they are reading or perusing documents. Events such as moving to the next page; checking comments, checking added notes or clicking on an available link within the document will be done by using eye gaze commands through this function.

1.4.3 Book management Module

This will be a hardcoded C++ function that will allow the user to select and open a document with a ‘PDF ‘file extension by locating and selecting it. This function will use the preinstalled default portable document reader to open the file but in this research, it is prescribed to use Foxit Reader Version 6.0.5.0.618. This is due to its inbuilt professional features that it comes with providing an excellent environment for both creating and reading ‘PDF’ documents.

1.4.4 Defining User award giving function

This function will be having a calculated value that accumulates depending on the number of line-words read and the duration taken.

Chapter 2 Literature Review

This background review reports a wide range of knowledge that have been, and in some cases still, used for tracking eye movements. However, the interest of this research is not in eye movements but rather in gaze tracking. That is why the term "gaze tracking" will be preferred than eye movement because the essential point is to measure the direction of gaze, more accurately the point of gaze. How do i get from observing the movements of the eye to information on the point of gaze?

2.1 The Human Vision- Biological basis

The important organ responsible for human vision is the eye. The important factor to consider when looking at the eye structure is where the receptors are. **Receptors**. As stated in [10], it would be a problem if there was a uniform spread of receptors over the eye but instead there are higher and lower density areas. One such area is the fovea. As shown in figure 2.1, the fovea is a black spot on the back of the eye. According to [10], the fovea, fovea covers a one degree width with vertex of the eye. This implies that the further away from the eye we get the larger the area of uncertainty on the screen is. Outside of the fovea the clearness of a person's vision drops to half or less that of the fovea and in [10] it was found that peripheral vision is not good enough to focus or read text with. Considering that the wish is to find the focus point of vision, the fovea then becomes the area to consider in this research. It is mentioned in [17] that to focus on an object a person will move their eyes such that the focus area is within the fovea area. A problem with considering the fovea as, discussed in [10], is that within the fovea there is still one degree of error. If the focus area is sufficiently small enough then the focus point could shift without any eye movements since the focus point would still be within the fovea's one degree of vision. The consequence of this is that it is unlikely to get less than one degree of error when measuring using eye position. This is consistent with the errors in [20], [7], [16], [13] and [19].

Another problem is that the fovea is within the eye socket and extremely small, as can be seen by figure 2.0. This makes using it as a feature for tracking difficult. Instead the pupil or iris of the eye is tracked and focus point inferred from that. This is possible because the object of focus is kept in the center of the eye and inferring the fovea's focus point from those results. The area of interest thus has become the pupil or iris as stated above [12].

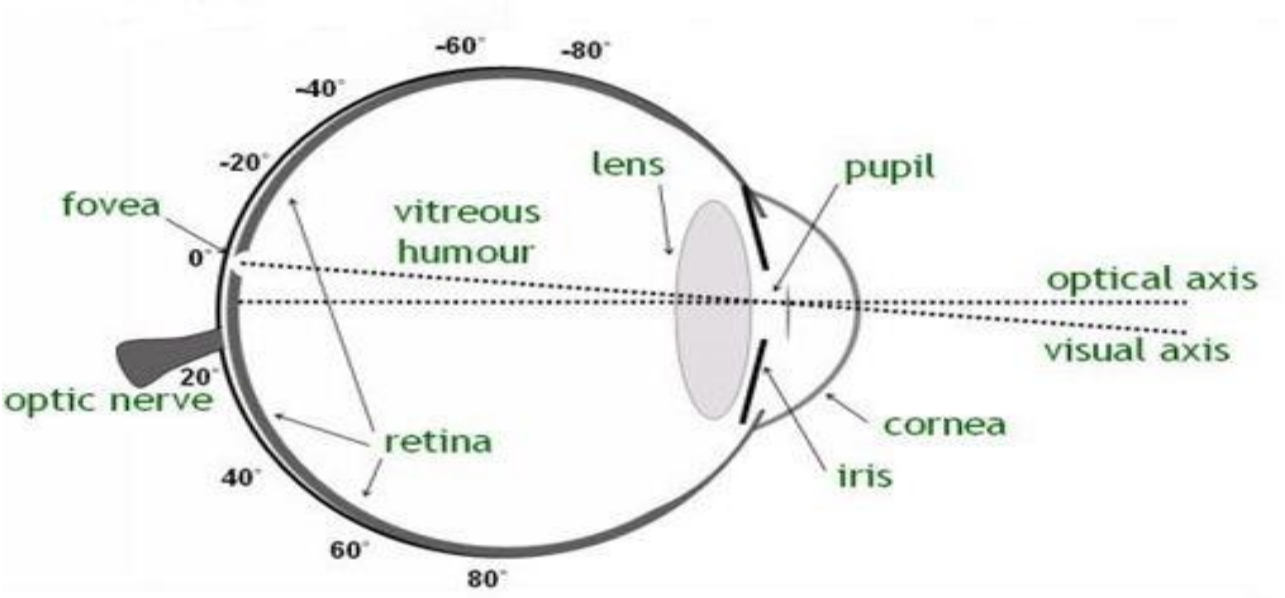

Figure 2.0 Cross Section of human Eye

2.2 Eye movement

Regular human interfacing with a computer using a mouse is smooth. The mouse moves smoothly across the working area while the user is moving it to the point of focus on the screen. This is perceived the same for our vision. In [9], [10] and [12], eye movements are described as a series of sudden jumps with rest points in between. The jumps are called Saccades and the rests in between are named Fixations [12].

Saccades

As stated above, saccades are extremely quick jumps between focus points. In [16], a paper on saccadic suppression, it was found that "trans saccadic spatial memory is sacrificed in order to maintain stability" [16]. This means that during the saccadic motion no visual information is gained in order to maintain the perception of smoothness and stability. This effect is known as saccadic suppression [12]. So to gain information into the focus points, it is not necessary to look into the saccades but rather the fixations become the more valuable source of information. It is noted in [12] that these quick movements need to be differentiated from three other eye movements, referred to as saccadic eye movements (SAC) in [6]. These movements are pursuit,

vergence and vestibular. Pursuit eye movements follow a moving focus point, performing saccades to catch up the point is moving too rapidly, but the general eye movement is notably slower than that of saccades. Vergence eye movements are when the eye move inward to focus on a point that is nearby. Vestibular movements are rotations of the eye to compensate for larger head or body movements. These three movements are less important than saccades for information processing and such as for gaze tracking [12].

Fixation

The rests in between saccades, where the eye is focusing and gaining visual information, are called fixations. Jacob describes as "a period of relative stability during which an object can be viewed" [10]. The reason for the term relatively in the above statement is because there are still tiny movements during fixations. These movements are named smooth pursuit eye movements (SPEMs) in [6]. [10].

There are three different types of these small movements and they are usually "within one degree radius" [10], which agrees with the previously mentioned fovea error. The first is a constant tremor of the eye called nystagmus. The second movement is known as drift, which is where the eye occasionally drifts slightly off the focus point due to a "less-than-perfect control of the nervous system" [12]. These drifts are then compensated for by micro-saccades, which are an even more rapid movement than saccades, which brings the focus of the eye back to the project. [12]

As mentioned above the requirement is only to look at the fixations to find the focus of the eye. This is good since the fixation last longer than the saccades. [12].

2.3 Methods of finding the eye

The most precise methods of finding the eye accurately involve uncomfortable equipment such as using a contact lens that is placed in place on the eye. Although it is accurate, it's invasive and uncomfortable to use. The following are the literature on the methodologies.

2.3.1 Infrared Reflection

In most approaches finding the focus point it is necessary to identify the region of interest (ROI).Since different parts of the eye have different refractive index to infrared light the pupil appears black and the rest of the eye is not. Using a threshold technology to process grey image, they can check for all points below the threshold to find the region of interest. The threshold uses two equations below [15].

$$x_{pupil} = \frac{1}{N} \sum_{n=1}^{N} x_n$$

$$y_{pupil} = \frac{1}{N} \sum_{n=1}^{N} y_n$$

This is the general idea of using Infrared; you use the light to simplify the Process of pinpointing the center of the eye. This shortens a potentially time Consuming task which can improve the efficiency of an Eye tracking system. The negative side of this though is that a light is needed which makes the Product slightly harder to distribute. There could also be other light sources, or shadows that disrupt this and create noise making it harder to calculate the focus point.

2.3.2 Haar Classifiers

Haar classifier objects are based on a compilation of Haar-like features. These features use the changes in contrast values between adjacent rectangles of pixels to determine relative light and dark areas. Two or three rectangles with relative contrast differences form a single Haar-like feature. The features, as shown in figure 3.1 below, are then used to detect objects in an image. These features can be scaled up and down easily by increasing or decreasing the size of the pixel group. This allows Haar-like features to detect objects of various sizes with relative ease. [14].

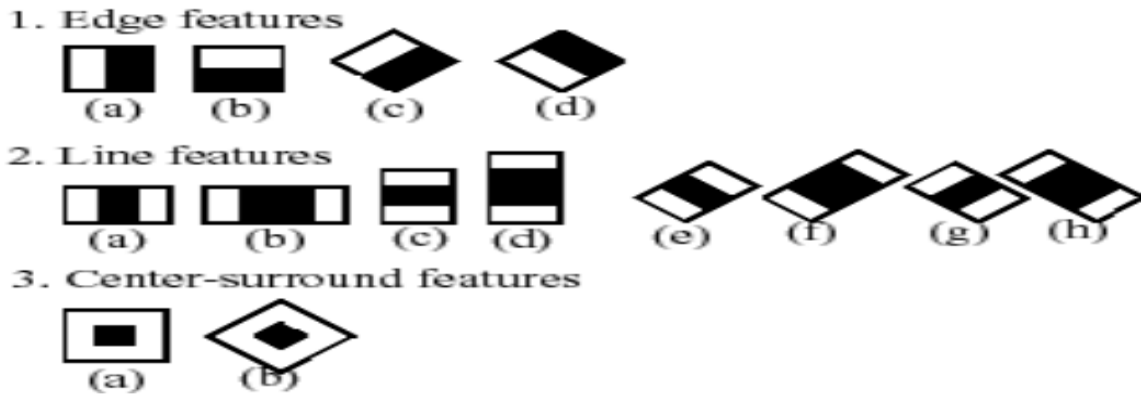

Figure 2.1 Haar-like features

The rectangles themselves are calculated using an intermediate image called the integral image. This integral image contains the sum of the intensities of all the pixels to the left and above the current pixel. The equation for this is shown below. [14]

$$AI[x,y] = \sum_{\substack{x' \leq x \\ y' \leq y}} A(x',y')$$

However, the rotated features, as seen in 3.1, create a different integral image called a rotated integral image based off the equation below instead. [14]

$$AR[x,y] = \sum_{\substack{x' \leq x \\ x' \leq (x - |y - y'|)}} A(x',y')$$

Using the integral image or rotated integral image as needed, and taking the difference between two or three connected rectangles, it is possible to create a feature of any scale. A benefit of Haar-like features is that calculating features of various sizes requires the same effort as two or three pixels. This makes Haar features fast and efficient to use as well. [14]

A requirement to create classifiers for specific objects is to train the classifiers using a set of positive samples containing the image to be detected and a negative set not containing the image. Once trained, the classifiers can be used to detect the objects [14]. Some of the more popular uses of Haar Cascades are Eye and Face detection.

2.4 Algorithms

OpenCV (Open Source Computer Vision) is an open source library of programming functions for real time computer vision projects. OpenCV is released under a BSD license and hence it's free for both commercial and academic use. It has C++, Python and Java (Android) interfaces and supports Windows, Linux, Android, IOS and Mac OS. It has more than 2500 optimized algorithms for image and video analysis. Since its introduction in 1999, it has been largely adopted as the primary development tool by the community of researchers and developers in computer vision. OpenCV was originally developed at Intel by a team led by Gary Bradski as an initiative to advance research in vision and promote the development of rich, vision-based CPU-intensive applications. After a series of beta releases, version 1.0 was launched in 2006. A second major release occurred in 2009 with the launch of OpenCV 2 that proposed important changes, for the C++ language, which will be used in this research. At this time of writing, the latest release is 2.4.8 (January 2014). The following are some of the algorithm that is worth reviewing for the research.

2.4.1 Real-Time gaze tracking algorithm with head movement compensation (RTAC)

The explanation of RTAC in [7] begins by referring to the "Pupil Center Corneal Reflection (PCCR)" [7, p395]. According to the PCCR, gaze direction calculations firstly acquire the pupil-glint vector, and then use a gaze mapping function. The glint is the reflection of the infrared light source in the eye. The pupil-glint vector is then the 2D vector between the glint and the pupil. The algorithm then uses a mapping function to map this vector to the 3D gaze direction. The mapping function is specified by the following functions:

$$\theta_h = b_{\theta h} V_x + a_{\theta h}$$

$$\theta_v = b_{\theta v} V_y + a_{\theta v}$$

Where Θ_h is the angle between the gaze direction and the horizontal direction and likewise Θ_v is the angle between the gaze direction and the vertical direction. The coefficient 'a' and 'b', are estimated using the sets of pairs of **pupil-glint vectors**. The next step in this algorithm is the head movement calculation. The previous calculation does not include adaption to head movements. As such, the system needs to compensate for these head movements. This calculation is quite complicated. The basis for it, however, is these two equations:

$$\Delta I_x = (b_{2x}\Delta L_h + a_{2x})\Delta d^2 + (b_{1x}\Delta L_h + a_{1x})\Delta d + (b_{0x}\Delta L_h + a_{0x})$$

$$\Delta I_y = (b_{2y}\Delta L_v + a_{2y})\Delta d^2 + (b_{1y}\Delta L_v + a_{1y})\Delta d + (b_{0y}\Delta L_v + a_{0y})$$

These two equations describe the changes in head position in the horizontal and vertical plane respectively. The derivation of these equations can be found in [7, p397]. Once the head movement has been calculated then the system needs to compensate for this movement. The compensation method employed by this algorithm uses two sets of equations:

The calculation of the proportional change in magnification and then using these values to calculate the compensation value in the horizontal and vertical directions. Then the next step is fixing the users head during calibration in order to get the initial parameters required. After this calibration, the user can move their head freely and the parameters are compared to do the calculations. In [7], it was found that this algorithm had an accuracy rate of approximately one

degree. However, the challenging part is to compensate for the error incorporated when the head is moved either out or nearly across the cameras' capturing boundary.

2.4.2 Neural Networks

The first step of using neural networks, according to [13], is Face and Eye detection. The eye and face are detected using the intensities of the pixels in a region and then each region is given a score. The scores are then arranged hierarchically and fed to an OpenCV library for detection [13]. The next step is to reduce the Region Of interest to a smaller picture. This picture is usually extremely small being less than a thousandth of the size of the original image and only containing the eye. From there the image is processed to make a clearer input for the neural network. The two processes applied are histogram equalization that brightens the sclera and darkens the boundary of the eyes, and resizing the image to a smaller format using a bicubic interpolation. This reduces the pixel by fourfold which cuts the training time of the NN from several minutes to a time generally less than 60 seconds. [13]

Chapter 3 Methodology

The type of methodology used in the implementation stage is a hybrid of incremental and iterative development that incorporates the principle of separation of concern achieved through program modularization. It has to be noted, though, that several modifications took place during development phase in contrast to what was planned during project proposal presentations to finish the project within the required timeline. The OpenCV library, which has been developed for image processing by Intel Corporation, was utilized to write the necessary functions. The library is essentially aimed at real time computer vision, and at handling different image processes such as motion tracking, face and gesture recognition, object identification, etc. To build the system, a divide and conquer approach was used called modularization were the entire system was partitioned into several modules classified by programming language functions. This approach enabled work to be done simultaneously in parallel on separate small tasks. The next step was to assemble them and test their functionality altogether. The IDE for coding used was Microsoft visual studio 2010 and Qt creator 5.2.0 for the User interface. The following is a picture of the UI designed.

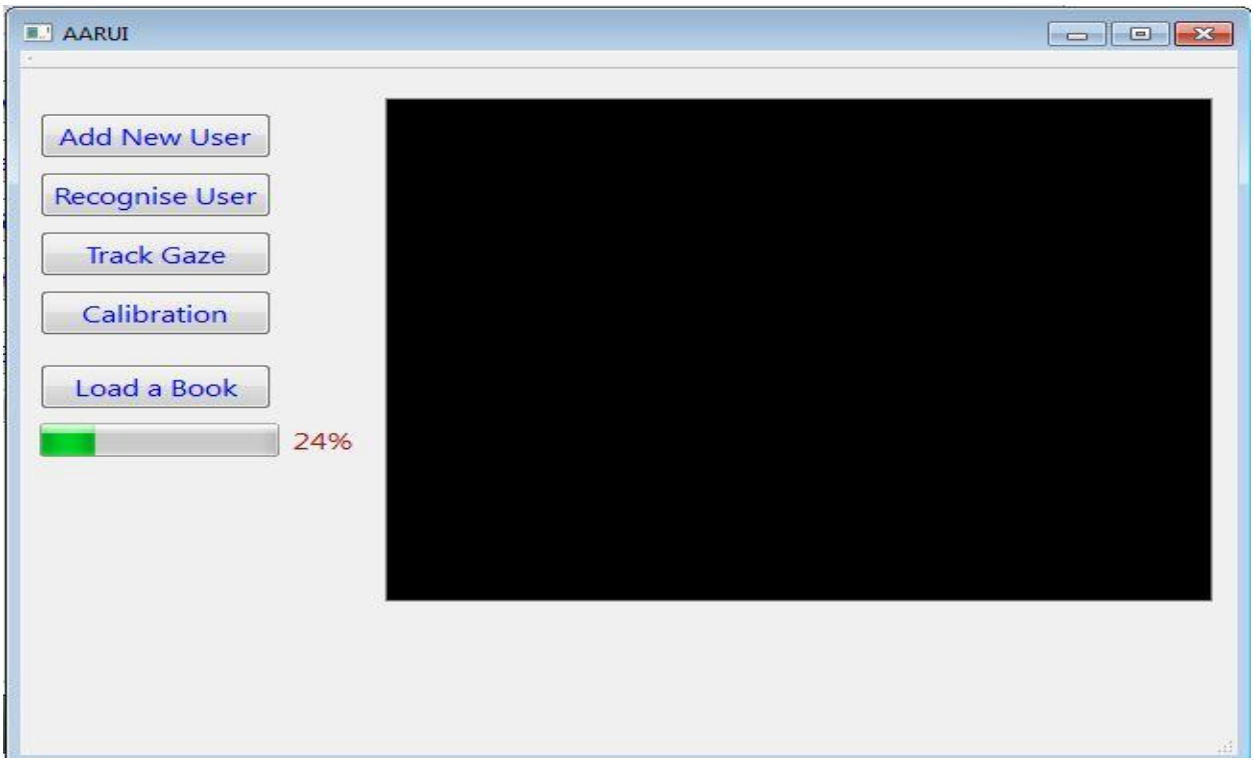

Figure 3. 0 expected UI

Chapter 4 Design

This chapter explains all the necessary algorithms and in-built functions that helped achieve the goal of the face recognition and Gaze tracking module for assisted and augmented reading prototype.

4.1 Design Choices.

Programming Language Of choice.

A natural choice to make was which programming language the system should be implemented in. OpenCV supports C, C++ and Python, C was a good choice in that it operates close to the hardware but C++ was a much better choice in that it was an improvement of C whilst Python was a choice below par as it was not close to the hardware. C++ became the choice with an extra advantage of it been fully supported by the IDE been used called 'Visual Studio 2010'.

Algorithm of choice for Face Recognition

The algorithm of choice for face recognition was Eigen Face based on Principal Component Analysis. The reasons were:

- Simplest and easiest method to implement
- Very fast computation time
- Accurate – this method was definitely not the most accurate of face recognition algorithms but considering the requirements for the project it was judged to be accurate enough
- PCA is supported within the OpenCV library – this key point made integration with the face detection algorithm very easy

The algorithm used for face detection was viola jones.

4.1.1 Theory of Eigen Faces

Eigen face technique uses principal component analysis of the images of faces. This analysis reduces the dimensionality of the training set. Leaving only those features that are critical for face recognition. Eigenfaces are a set of eigenvectors used in computer vision problem of human face recognition. This approach of using eigenfaces for recognition was developed by Sirovich

and Kirby (1987) and used by Turk and Alex Pentland in face classification. It is considered the first successful example of facial recognition technology.

4.1.1.1 Mathematical basis of PCA

Prepare a training set of face images. The pictures constituting the training set should have been taken under the same lighting conditions and must be normalized to have eyes and mouths aligned across all images. They must also be all resampled to the same pixel resolution. Each picture has 'R' rows and 'C' columns.

Step One

Convert each picture in the training set into a vector of length' R x C' by concatenating the rows of pixels of the original image

Step Two

If there are 'n' training set images then create a matrix, M where the number of rows = "N" and the length of each row ="R x C". Each row will be represented by one of the image vectors.

Step Three

Calculate the mean image 'A' of the 'N' image vectors. Subtract A from each row of 'M' to obtain the matrix 'T'.

Step 4

The covariance matrix 'S' is given by $S = T^t . T$ where T^t is a transpose matrix of T.

Computing covariance matrix using $T^{t.}*T$ instead of $T*T^t$ overcomes a great computational problem that might crash the computing device or reduce its performance badly. It consumes processor and memory resources because the resulting matrix is very large

Step Five

Calculate the eigenvectors and eigenvalues of 'S'. R x C number of eigenvectors is obtained but the main idea about the principal components is to store only the eigenvectors with the highest value eigenvalues.

4.1.1.2 Recognizing an input Face

Step One

Obtain an Input Image, I

Step two

Subtract the mean image A from the input image

D = I-A

Step Three

Project D onto the face space

P = Eigenfaces' x D

Step Four

Find the Mahalanobis distance of this projected image to the images already in the face space from the training set. Return the one with the smallest distance.

4.1.1.2.1 Eigenvector computation

The OpenCV library has functions specifically designed to calculate the eigenvectors and eigen-values of a set of input images. The first one used is the function that "Calculates orthonormal Eigen basis and averaged object for group of input images".

cvCalcEigenObjects(int nObjects, void* input, void* output, int ioFlags,int ioBufSize,void* userData,CvTermCriteria* calcLimit,IplImage* avg, float* eigVals);

Here is the description of each argument for the above function:

1. nObjects : number of input images to be used.
2. input : pointer to an array of input images.
3. output: pointer to an array eigen objects (vectors).
4. ioFlags : input/output flags.
5. ioBufSize : input/output buffer size in bytes.
6. userData : pointer to structure containing the data for callback functions.
7. calcLimit : criteria used to determine when to stop computation of eigen objects.
8. eigVals : pointer to array of eigenvalues in descending order.

Before invoking this function there were a few variables that needed initializing, the number of input objects was set to the number of users of the system. The eigenvectors were stored as an IplImage variable which is the same as the input images. The termination criterion was also defined before the function is called. Here are the constraints contained in the criteria:

```
criteria.type = CV_TERMCRIT_ITER|CV_TERMCRIT_EPS;
criteria.max_iter = 10;
criteria.epsilon = 0.1;
```

This shows that the criteria variable contains two constraints. The max iteration variable set at 10 means that the maximum number of principal eigenvectors computed is 10. Epsilon set at 0.1 means that the function will stop calculating eigenvectors when the current eigenvector contains 10 times less variance then the eigenvector with the highest variance i.e. eigenvector with the highest eigenvalue. The eigenvalues are initialized as a floating point variable.

The function below how the above function was achieved in the face recognition module:

```
cvCalcEigenOb-
nOb-
jects(numberTrainFaces,(void*)faceImgArray,(void*)eigenVectArray,CV_EIGOBJ_NO_CALL
BACK,0,0,&calcLimit,pAverageTrainImg,eigenValMat->data.fl);
```

This function returned the eigenvalues and the eigenvectors of the set of images.

4.1.1.3 Calculating decomposition coefficients

This function follows the cvCalcEigenObjects function. It calculates the decomposition coefficients of each input image i.e. it determines where the input image is located in the Eigen space by expressing them as coefficients of the eigenvectors.

```
void cvEigenDecomposite( IplImage* obj, int eigenvec_count, void* eigInput, int ioFlags, void*
userData, IplImage* avg, float* coeffs );
```

The function's arguments are described here:

1. obj : input object (only one input image may be processed at a time).
2. eigenvec_count: the number of eigenvectors used.
3. eigInput: pointer to an array of input eigen objects (outputted from the function in 4.2.1).
4. ioFlags : input/output flags.
5. userData : pointer to structure containing the data for callback functions.
6. avg : averaged object (outputted from the function in 4.2.1).
7. coeffs : pointer to an array of coefficient values.

The statement below shows how the above function was implemented in the face recognition module.

cvEigenDecompo-
site(preprocessedFaceImg,numberEigens,eigenVectArray,0,0,pAverageTrainImg,pTestFace);

Now the decomposition coefficients of each image are stored. Before these coefficients are used in the next function they must be normalized so each vector coefficient is given equal importance.

4.1.1.3.1 Calculating the Mahalanobis distance

To Calculate the Mahalanobis distance between the faces in the Eigen space, this required using AI function called findNearestNeighbor() that implemented a mathematical formula for calculating the Mahalanobis distance, this function could return the closet image on the Eigen space to the image passed to it as a parameter, the threshold value that helped achieve better recognition accuracy is 0.5f, this threshold value was a choice made in compliance with the recommended value for implementing Eigen face algorithm.

4.1.2 Theory of Viola Jones Algorithm

Viola Jones Algorithm uses Haar classifier objects based on a compilation of Haar-like features. These features use the changes in contrast values between adjacent rectangles of pixels to determine relative light and dark areas. Two or three rectangles with relative contrast differences form a single Haar-like feature. The features, as shown in figure 4.0 below, are then used to

detect objects in an image. These features can be scaled up and down easily by increasing or decreasing the size of the pixel group. This allows Haar-like features to detect objects of various sizes with relative ease.

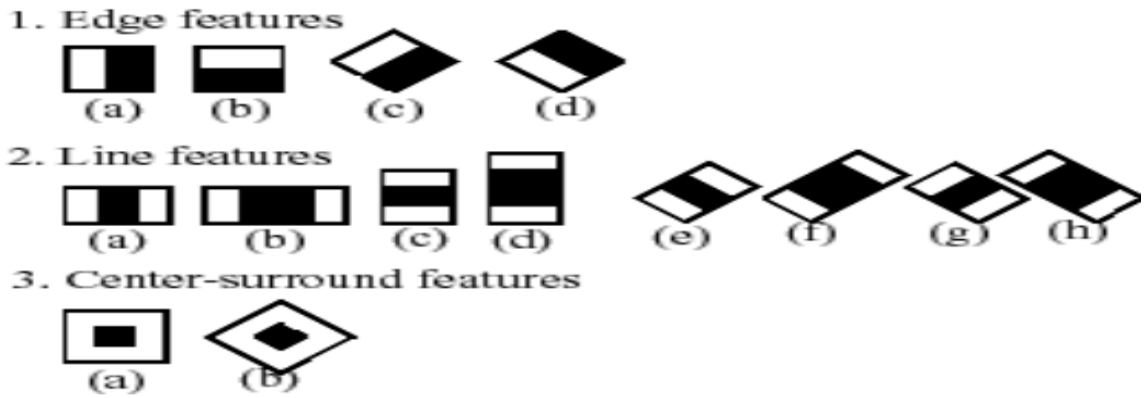

Figure 4.0 Haar-like features

The rectangles themselves are calculated using an intermediate image called the integral image. This integral image contains the sum of the intensities of all the pixels to the left and above the current pixel. The equation for this is shown below. [14]

$$AI[x, y] = \sum_{\substack{x' \leq x \\ y' \leq y}} A(x', y')$$

However, the rotated features, as seen in 4.1, create a different integral image called a rotated integral image based off the equation below instead. [14]

$$AR[x, y] = \sum_{\substack{x' \leq x \\ x' \leq (x - |y - y'|)}} A(x', y')$$

Using the integral image or rotated integral image as needed, and taking the difference between two or three connected rectangles, it is possible to create a feature of any scale. A benefit of Haar-like features is that calculating features of various sizes requires the same effort as two or three pixels. This makes Haar features fast and efficient to use as well.

A requirement to create classifiers for specific objects is to train the classifiers using a set of positive samples containing the image to be detected and a negative set not containing the image. Once trained, the classifiers can be used to detect the objects. OpenCv comes with pre-trained cascade classifier for detecting the face, which am going to use. The only requirement is that the frame must be converted to gray scale image before applying the classifier.

4.2 System Design

This section explains in details the design of each developed module namely face recognition module, Gaze tracking module except for the expected user interface of the system which has been shown under chapter 3.

4.2.1 Face Recognition Module

Face recognition was a task of identifying an already detected face as known or unknown face and in more advanced cases it could mean telling exactly whose it belongs to. The following figure gives a visual explanation of face recognition.

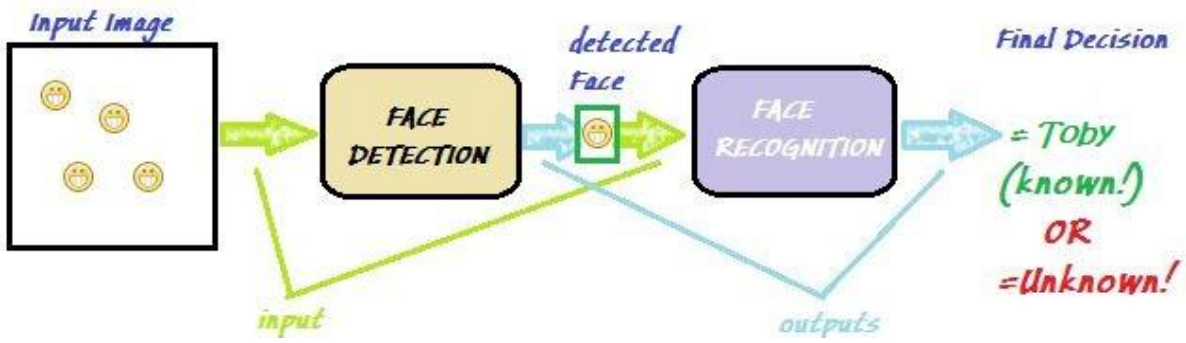

Figure 4.1 Face Recognition Process

4.2.1.1 Component Design Structure.

Component Overview

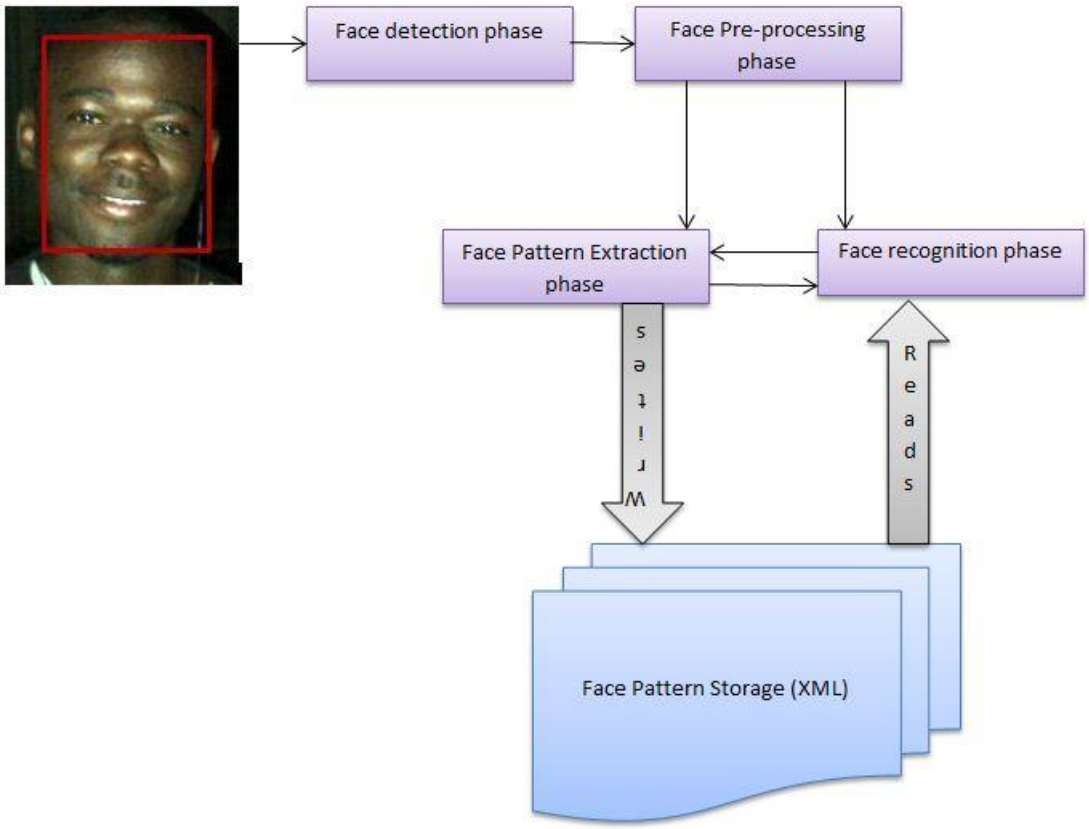

Image acquisition phase

The initial stage of this module was to acquire an image in real time through the webcam, by invoking OpenCV function called cvCaptureFromCAM (0), passing to it the index number for the camera to open, this function was initiated in an infinite while loop that made it possible to continuously open the webcam, to get a single frame form the opened video stream cvQueryFrame () was called on the object holding the webcam video input stream, with this function a single frame could be captured from the video input stream, this set the ground for the next phase of face detection.

Face Detection Phase

Upon succeeding with image acquisition phase, the camera used produces BGR image type which conflicted with the requirements of using viola jones for face detection, to meet the requirements of viola jones a duplicate copy of the original frame was created via copyTo(), using the cvCvtColor () with appropriate parameters the copy could be converted to grey image, by loading the cascade classifier and applying it to the grey scale image using cvHaarDetectObjects () function , a rectangle object that contained a face was returned using the CvRect class.

Face Pre-processing phase

The goal of this phase was to help reduce noise from the detected face so as to help increase the accuracy for recognition. By cropping the face image to the rectangle size containing a face, to be precise, removing parts of the image that didn't contain the face and then resizing it using cvResize() facilitated in meeting the requirements for using cvCalcEigenObjects() and cvEigenDecomposite() for learning and recognition respectively. Lastly to finalise this phase cvEqualisHist() was invoked on the resized image, this function automatically equalised the pixel intensity in the image, this yielded the results required for learning and recognition although several other functions could have been used to help improve accuracy.

Face pattern extraction (learning) phase:

The core of this phase was to extract a unique pattern from an image that could be used to help differentiate it from other images. Eigen face algorithm implementation in OpenCV for learning

is achieved by calling cvCalcEigenObjects (), this is an iterative algorithm. OpenCV library hides the implementation of this function. The programming technique behind using cvCalcEigenObjects () was to make its input parameters as global variables. When this function was invoked and it returned from the call, it produced good intended side effects that updated the global variables some results from its computation are irrelevant such as the smallest Eigen values with their respective Eigen vectors which required normalising by using cvNormalize() to yield the necessary output. When the Eigen value is very small it implies that its corresponding Eigen vector has less unique features to help identify the image it represent. To avoid the overhead of computing Eigen space on the image database every time when trying to performing recognition, persistence functions from CvFileStorage class were used to create a file storage object. These persistence functions are specifically designed for storage of persistence data, their advantage is that they automatically created the xml file structure, and inserted node names with their respective node values.

Recognition Phase:

The success of face recognition was based on the fashion in which Eigen vectors are mapped onto the Eigen space, Eigen vectors that are similar in pattern to each other have the smallest distance in between, by projecting the result from the pre-processing phase using cvEigenDecomposite() onto the space with other important result such as the average training image from the cvCalcEigenObjects(), by defining a mathematical formulation called the mahanalobis distance in an AI function called findearestNeighbor(), using the threshold value of 0.5f it could return the nearest most similar Eigen vector existing on that space within that threshold value. This finalised the aim of face recognition.

4.2.2 Gaze tracking with HCI Module

Gaze tracking was a task of tracking the movement of eyes while gaze estimation, the process of finding the point of focus on the computer screen. To track the movement of eyes required locating the centre of the eye (to be specific the pupil) which in this module was achieved by implementing the algorithm proposed by Fabian Timm and Erhardt Barth in a research paper called “accurate eye centre localisation by means of gradients”, to implement his algorithm the requirements where doing a face detection phase and then performing a rough bisection of the detected face into two parts where one contained the right eye region and the other left eye region.

The general idea behind their algorithm is using the image gradient to find the centroid of the eye to be more precise, pupil and iris localisation. Their algorithm uses simple mathematics, is accurate and robust to different lighting conditions this was an enticement to implement it in this research project. Their idea was achievable in OpenCV by using Sobel operators. The following sections give a detailed description of their algorithm in a paraphrased format with some form of abstractions.

In their work they analysed the vector field of image gradients and derived a novel mathematical formulation of the vector field characteristics, with this in mind they mathematically describe the relationship between a possible centre and the orientations of all image gradients using results expected from the cross products of the image gradient that intersect each other at the centre of the eye

Gaze tracking design

The following gives a detailed description of the gaze tracking module designed and algorithms used based on the research paper for accurate eye centre localisation by means of gradients.

Face detection phase

The implementation of this phase is the same as in face recognition module.

Upon succeeding with image acquisition phase, the camera in use produces BGR image type which conflicted with the requirements of using viola jones for face detection, to meet the requirements of viola jones a duplicate copy of the original frame was created via copyTo()

method, using the cvCvtColor () with appropriate parameters the copy was converted to grey image, and by loading the cascade classifier and applying it to the grey scale image using cvHaarDetectObjects () function a face was detected, this function returned a rectangle object that contained a detected face.

Face Bisection Phase

The Bisection phase involved dividing the detected face roughly into two regions, one region containing the right eye and the other the left eye, however converting the two regions into binary images was achieved via thresholding as required by the algorithm in that research paper, and this phase was a preparation done on the frame for the next phase.

Sobel operation phase

The algorithm for Sobel operations computed the change in image gradient between two sequential rows and two sequential columns separately, in the x-axis (rows) and y-axis (column) of the image with a pre-knowledge of the composition of a binary image, a single pixel value has a 1 or 0 for its value, the algorithm performed two types of sequential operations, the first sequential operation was to scan the image from the top first row to the last one and the last bottom row of the frame and the second operation was to perform a scan in a similar order for the column pixels, in each operation a change in gradient was computed, when two consecutive pixels have two different values in a single operation ,that was marked as an edge. A problem that arose was when performing an operation to get the image gradient for the first row and first column, to solve this problem Fabian Timm and Erhardt Barth proposed the use of masking, where a mask of a column and a row is created and attached to the actual pixel column and pixel row respectively, this solved the problem. The following is the Sobel operation design for the x (set of pixels in each row) gradient. To get the y gradient only a transpose matrix of a transpose was performed (computeSobelXOperation (roi.t ().t)). However, in that research paper there were several other important things that were missing, through an intensive research solutions were captured that helped make tremendous progress in gaze tracking module. The following highlights the implementation of calculating the image gradient in the x-axis of an image.

```
 cv::Mat computeSobelXOperation (const cv::Mat &mat)
{
 cv::Mat out(mat.rows,mat.cols,CV_64F);
 for (int y = 0; y < mat.rows; ++y)
 {
   const uchar *Mr = mat.ptr<uchar>(y);
   double *Or = out.ptr<double>(y);
   Or[0] = Mr[1] - Mr[0];
   for (int x = 1; x < mat.cols - 1; ++x) {
     Or[x] = (Mr[x+1] - Mr[x-1])/2.0;
   }
   Or[mat.cols-1] = Mr[mat.cols-1] - Mr[mat.cols-2];
 }
 return out;
}
```

Chapter 5 Developed and Expected System

This section highlights the design of the developed and the expected system. The developed and expected design shows what has been and has not been implemented respectively. The following diagrams show this information. The reading detection algorithm has not yet been implemented.

5.1 Developed Design

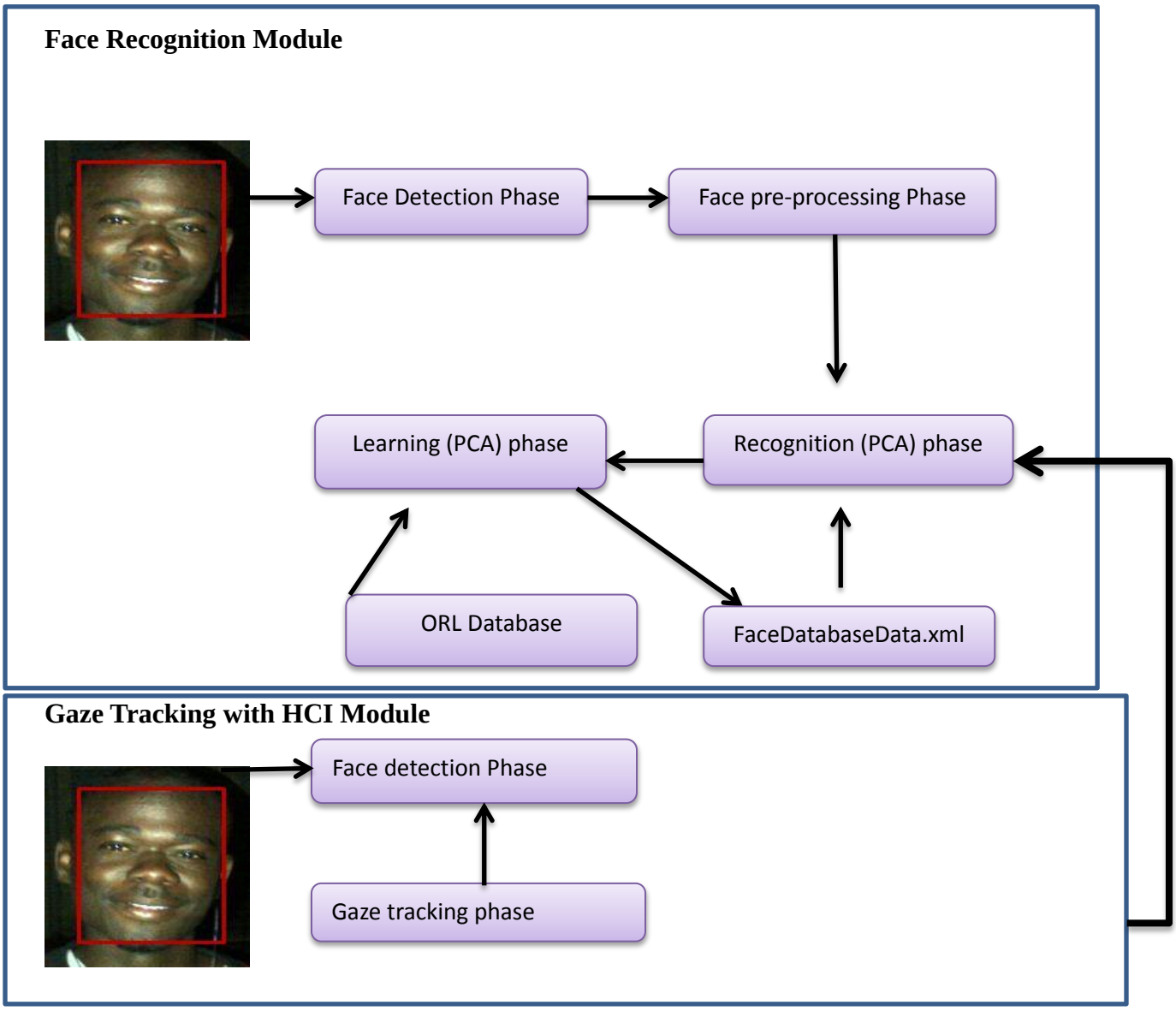

Figure 6.0 developed system

5.2 Expected Design

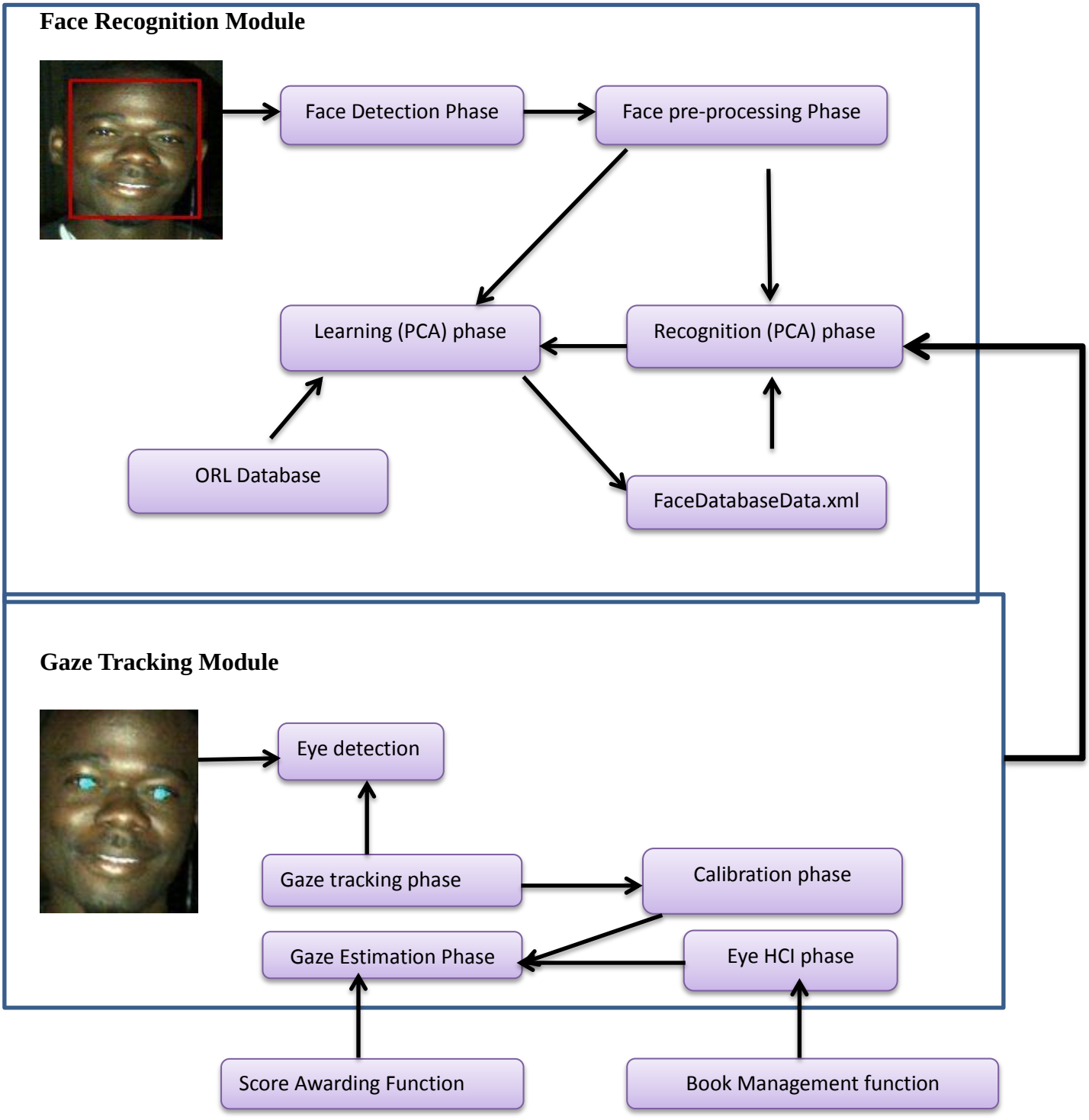

Figure 6.1 Expected Design

Conclusions

The aim of the project was to design an interactive and entertaining reading experience through webcam eye tracking. In relation to the progress of the project, one thing is indisputable that assisted and augmented reading is achievable through an ordinary webcam. Among all the objectives, the ones that were achieved are a face recognition and gaze tracking module with a user interface created using an open source IDE called Qt creator, though momentarily not yet integrated to the other two modules at the time of writing. During the development of the face recognition system the ORL databases of faces was used, which is readily available for experimental usage in the design of computer vision based systems.

This was an interesting and challenging project during which a lot was learned about real time image processing for face detection, eye detection eye tracking, face recognition etc. Valuable experience has been gained in working with the windows operating system and the Microsoft visual studio 2010. Also to mention further experience has been gained with C++ programming language and several other programming techniques and approaches that are worth considering when creating huge software systems. Working on this project highly enhanced my research skills, programming competence and art of solving various problems through programming abstractions.

Future work

Developing a sign Language recogniser using an image processing approach where a video camera is used to capture face and hand movements. The recognised gesture can be translated into words. By using the text-to speech convert the words can be further translated into speech. This would help incorporate the well-abled in hearing that are not able to understand sign language during news briefing by the mute or deaf.

Developing an Electronic Class Attendance Register for High School pupils that use webcam face recognition to register the present pupils. This would help automate pupil attendance registration and help eliminate the manual filing system present, which is not secure enough for record keeping (using hardcopy register books).

Webcam Eye tracking incorporated into application licensing issue agreement. This would help an application have an eye to see if the person installing it has gone through the licensing issue agreement upon which access to install the application can be granted only. This could enforce people know what they agree to when installing a particular application.

Glossary

Term	Description
AI	Artificial intelligence
Covariance matrix	A set of images in vector representation converted into a matrix where each row in that matrix is a vector representation of a single image
HCI	Human to computer interaction
Average image	Is an image that results by adding all the images then dividing by the total number of the added images
Learning	Extracting important data from an image that can be used to differentiating it from others
Recognition	Find two eigenvector representations of images that have their data patterns similar between the two spaces in the Eigen space.
Face detection	Locate the face of a person in an image
Training Images	A set of images taken as known(face recognition compares against this images)
Eigen space	A space where a set of eigenvectors with their eigenvalues are mapped
Image gradient	The result is a vector and its computed on a single row or column of an image using Sobel operations (dx, dy). A change is the difference in image gradient between two consecutive columns or rows
Eigenvectors	Vector representation of an image
Pre-processed face	Output of face pre-processing phase
PCA	Principal component analysis
ROI	Region of interest

Anything starting with "cv"	Belongs to OpenCV

Appendix A

Installing OpenCV

OpenCV is an open source computer vision library dedicated to easing the implementation of computer vision related programs. It is an extensive library. The following highlights the steps I found that helped me configure it with Microsoft Visual Studio 2010 IDE.

A.1 Installing in Windows

Installing OpenCV in windows is fairly a simple process when known. However, to easy linking the library in your project, it is useful to add some additional steps.

Thereafter downloading OpenCV. The installation run and the library extracted to a path, my preference is the 'C: /' drive. Then add the following to the system path under the variable name "path", the x86 in the path stands for a 32 bit OS.

"C:\OpenCV-2.4.2\build\x86\vc10\bin"

This completes the installation.

A.2 Linking Your Visual Studio Project

Once you have created a project in Visual C++, by right clicking on the "debug" you can create the "project property sheet" give it any name, then it should appear among files that are under debug. Double click on the created project property sheet, under the tab VC++ Directories include the directories below:

C:\OpenCV-2.4.2\include\opencv

C:\OpenCV-2.4.2\include\

Under the library directories and VC++ directories it is necessary to include the directory that corresponds to your system and the name of the folder to which the OpenCV was extracted to. In the case of this system the directory was' C: /' and the folder name 'Opencv-2.4.2' under the library add the following path:

C:\OpenCV-2.4.2\build\x86\vc10\lib

Under the heading linker-Input –additional dependences the following files are required to be added.

opencv_calib3d242d.lib

opencv_contrib242d.lib

opencv_core242d.lib

opencv_features2d242d.lib

opencv_flann242d.lib

opencv_gpu242d.lib

opencv_highgui242d.lib

opencv_imgproc242d.lib

opencv_legacy242d.lib

opencv_ml242d.lib

opencv_nonfree242d.lib

opencv_objdetect242d.lib

opencv_photo242d.lib

opencv_stitching242d.lib

opencv_ts242d.lib

opencv_video242d.lib

opencv_videostab242d.lib

The 242 stands for the version of OpenCV and the “d” at the end of the file is for debug solution when creating the release solution the “d” should be removed .