UNIVERSITY OF LIVERPOOL

2016/17

Student Name: **Weiwen Xu**
Student ID: 201138757

Project Title: COMP39X
Eye Anomaly Detection for Visual Impairment in Stroke Screening

Supervisor: **Terry Payne**

DEPARTMENT OF COMPUTER SCIENCE

The University of Liverpool, Liverpool L69 3BX

Abstract

Traditional way of inspecting eye movement disorder is by eye examination, which requires examiner to have strict medical training. However, eye motion could be subtle and rather hard to notice for people who do not receive such training. To address this problem, this research has been done to develop a software that uses simple hardware, namely iPad, to create a pupil detecting and tracking system. This software attempts to prove that by using iPad backwards camera, open source Computer Vision library OpenCV, and some other mathematical calculations, such pupil detecting and tracking system could be developed. This is achieved by researching a number of theories and techniques required for the system, and analyzing the performance of created system with various test cases. Analysis of the system concludes that although error occurring during performance is greater than desired, combining research done in this field, it is possible to create a robust system with simple hardware and apply it in daily life.

Contents

Chapter 1

Introduction

1.1 Statement of Problem and Research Aim and Objectives

According to [1], with an estimated 110,000 people in the UK having a stroke every year, stroke is becoming a more common disease. Visual impairment is one of the major consequence caused by stroke. While visual impairment includes numerable types of disorder, over half of the patients suffering from eye movement deficit [2]. However, diagnosing such eye motion disorder could sometimes be difficult because such eye movement could be rather subtle and unnoticeable. Therefore, computer-aided visual assessment could be helpful as it provides precise mathematical data in support of diagnosis.

There are a few commercial eye tracking systems on market currently such as those produced by LC Technologies (LCT) [3] and Applied Science Laboratories (ASL) [4]. However, these systems require user to wear a head-mounted device containing a camera sensor closed to the eye [3], which is fairly expensive and not practical for daily use.

In this research, a system that is able to run on common device such as iPad is expected. The anticipated system should be able to detect pupil object and return a visible mathematical data. It is expected to return precise pupil location in real-time and such data could be saved when user takes a picture. Deviation in available directions should be presented for later diagnosing.

Objective for such system is to be applied in hospital waiting room or hallway, where it could be used by nurses for primary diagnosis.

1.2 Challenges and Solution

Unlike face and eye detection, technique for pupil detection is not mature and majority of such study is relatively small.

Lack of stableness is one challenge in pupil detection. It is highly possible that pupil is not detected because it is relatively small compared to face and eye features. Another major

challenge is that it is hard to retrieve a precise pupil location. Other challenges include bad lighting on system performance, performance on difference races with various pupil color and etc.

Performance of detecting pupil from eye image could be largely influenced by the existence of brow, eyelid or eye lash. To exclude the influence from brow, eye image is cropped as brow is usually above eye. To reduce the influence caused by eyelid and lashes, the system used two algorithms to process eye image. One algorithm is to find circle in the area of interest and the other one is to find the darkest point according to gradients of the image. By combining the two, it is hoped that a relatively stable and precise result will be produced.

1.2 Overview of the project

The software is successfully produced. It fulfills most of its intended tasks. By processing input images, it is able to detect both left eye and right eye pupil locations in real-time at a time, and return deviations in four different orientations. Although error occurring during performance is higher than expected, this research could still be considered as successful.

Chapter 2

Background

2.1 Problem Background

Pupil detecting, as the name suggests, is detecting human pupil. Thus, the objective of pupil detecting is to get location of the pupil.

2.2 Human Eye

2.2.1 Eye Structure

Normally, from anterior view, eye contains three main components. They are sclera, iris and pupil. Eye structure is much more complex in eye anatomy. However, in topic of computer vision, pupil detection only interests in these three components.

As it is shown in the illustration below in Figure 2.1, pupil is normally the darkest park. Iris is in the shape of circle and pupil normally fills in the inner iris. Iris and pupil together could be seen as a circle with pupil being the center. There is a distinct edge between iris and sclera.

However, together with eye itself, there are upper and lower eyelids and eye lashes on both eyelids. Part of the iris could be covered by eyelid, which increases the noise for detecting precise pupil locations.

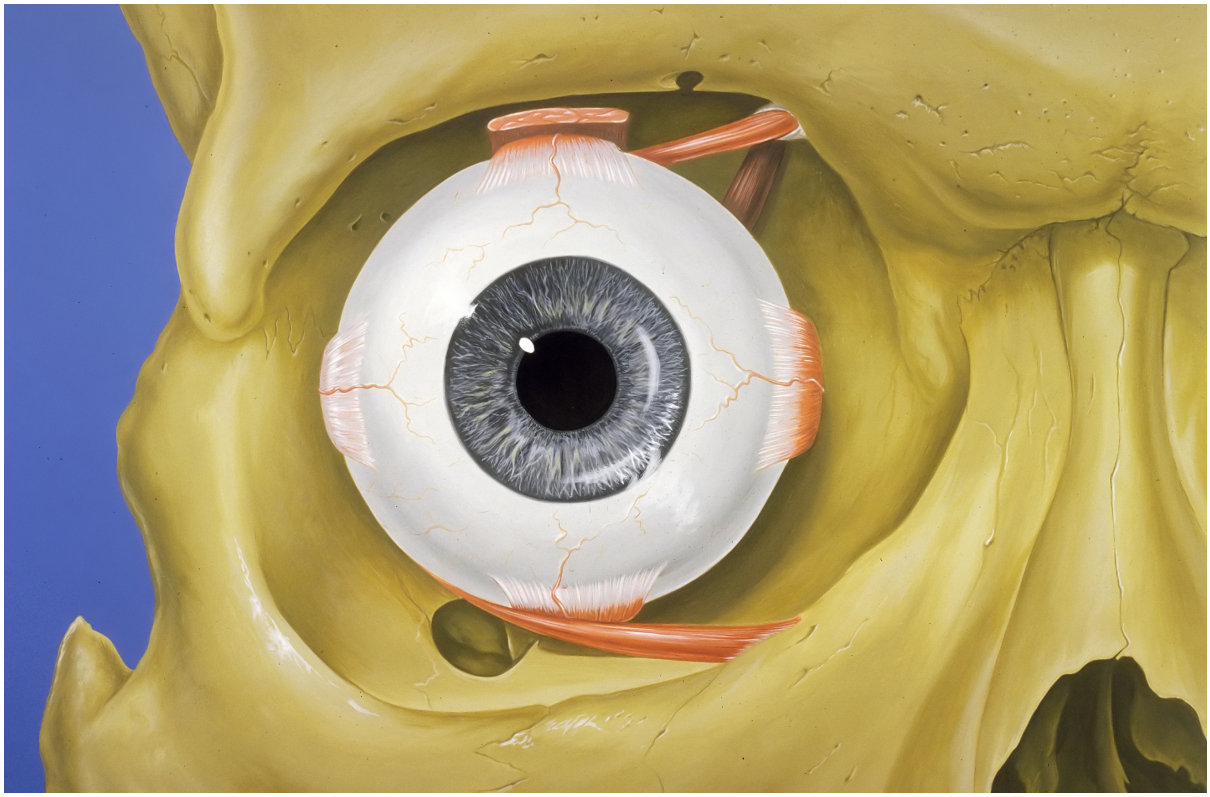


Figure 2.1 Normal anatomy of the human eye and orbit, anterior view: created by P. J. Lynch, medical illustrator, Yale University School of Medicine, Center for Advanced Instructional Media, 2006

2.2.2 Eye Movement Disorders

Eye movement disorders affected by neural impairment contain several types. These eye movement disorders could be examined at the bedside and by eye movement recording devices. There are four main methods in clinical examination of eye movement which are Fixation, Saccades, Smooth pursuit, and Vergence [5]. Fixation is to ask patient to stare at a particular object in fixed position. The eye should never be still during fixation however it remains in a small area. Saccades is to ask patients to look in four different directions, namely to the left, right, up, and down. In this project, it mainly focuses on this examination and all tests are based on this method for checking the accuracy of detected pupil location. Smooth pursuit is to ask patient to track a small target at about 1 m distance, while keeping the head still. Vergence is to ask patent track at an object that goes towards to themselves gradually.

2.3 Existing Approaches for Eye Tracking

2.3.1 Means of Gradient

Detecting eye center localization by analyzing the vector field of image gradients has been applied before by Kothari and Mitchell [6]. By drawing lines according to orientation of each gradient vector in the whole image, point which has the highest number of line intersect is considered to be the center of the eye [6].

However, their approach fails to consider the existence of eyelid and eye brows or glasses [7]. In [7], Fabian Timm and Erhardt Barth improves Kothari and Mitchell's approach by developing an innovative mathematical formula. As is shown in Figure 2.2, let **c** be the possible center. $\mathbf{g}_i$ is the gradient vector in position $\mathbf{x}_i$.

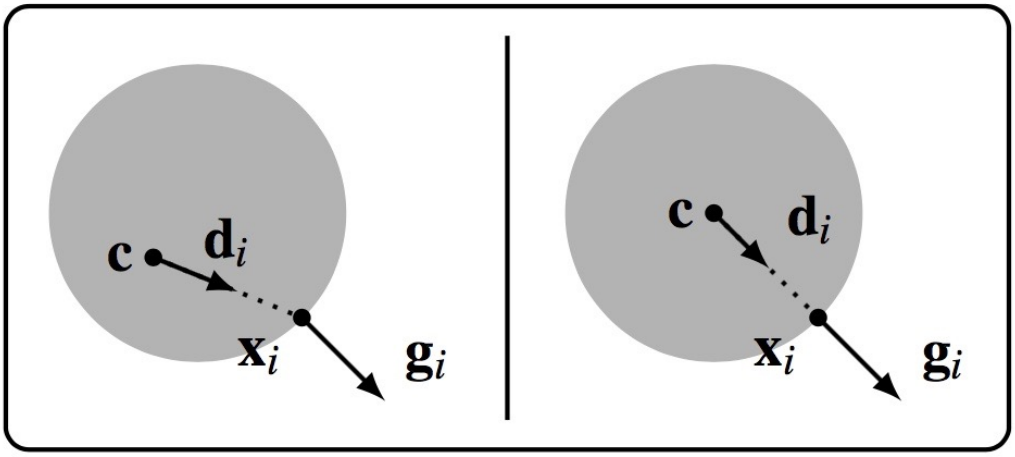


Figure 2.2 Artificial example with a dark circle on a light background, similar to the iris and the sclera. From: [7]

For all the possible points, the normalized displacement $\mathbf{d}_i$ should have the same orientation as $\mathbf{g}_i$. The equation for calculating optimal center c* is shown below [7]:

$$\overrightarrow{c^*} = \underset{\overrightarrow{c^*}}{\arg max} \left\{ \frac{1}{N} \sum_{i=1}^{N} \left(\overrightarrow{d_i^T} \overrightarrow{g_i} \right)^2 \right\}$$

For all the possible centers, the one with maximum value appears to be the most optimal center.

To obtain a uniform weight for all pixel positions within the vector field, the displacement vector is scaled to unit length [6]. Equation is presented below [7]:

$$\overrightarrow{d_i} = \frac{\overrightarrow{x_i} - \vec{c}}{\|\overrightarrow{x_i} - \vec{c}\|}$$

In order to improve robustness and performance in response to lighting change, the gradient vector is scaled to unit length [6]. Equation is provided below [7]:

$$\|\overrightarrow{g_i}\|_2 = 1$$

2.3.2 Circle Hough Transform

Hough Transform is a method for feature extraction [8]. In image processing, missing pixels or points in the desired feature boundary is highly possible [9]. This method, however, is able to allow that situation. The impact of noise on this method is relatively small as well [8,9], making it perfect for pupil and iris detection in this case. Its classic version deals with lines identification in images. The main algorithm being used in this section will be Circular Hough Transform, which is a modification of Hough Transform.

Circular Hough Transform is almost identical to its classic version. However, in this case, Circular Hough Transform focuses on finding circles. Circles can be described using the equation:

$$(\mathrm{X} - \mathrm{X}_0)^2 + (\mathrm{Y} - \mathrm{Y}_0)^2 = \mathrm{R}^2 \qquad (1),$$

where (X_0, Y_0) is the coordinate of circle's center and r is radius of the circle.

Within the circular Hough Transform method, if the desired circle has known radius R, the parameter space will be 2 dimensional because it only needs to find out center point (X, Y). It will be achieved by generating circles with radius R and known edge points. A 2D Accumulator matrix is used to record the number of intersections points between these circles. Each cell in accumulator matrix represents each pixel in the image. The point with highest number of intersections is center of the desired circle. This is also called voting [10]. However, in this case, the radius R is unknown. Therefore, the parameter space is 3 dimensional making accumulator matrix 3 dimensional. This time, it still tries to find the point where highest number of intersections exists but with random radius.

Pseudo code is:

$\mathrm{A}[\mathrm{m}, \mathrm{n}, \mathrm{R}] = 0;$

For each cell (X, Y)

    For each radius R from A to B

        Find (m, n) that according to $(\mathrm{X} - \mathrm{m})^2 + (\mathrm{X} - \mathrm{m})^2 = R^2$ could be circle center;

        $\mathrm{A}[\mathrm{m}, \mathrm{n}, \mathrm{R}] + +;$

    End

End

It is computational expensive compared to detecting circles with known radius. J. Illingworth and J. Kittler proposed a method called The Adaptive Hough Transform which could successfully reduce the cost of Circular Hough Transform [10]. It does not use (1) directly. Instead, it reconstructs the circle finding problem into two-stage algorithm [11]. The first stage is to find the center and the second is a histogram step to identify the radius [10].

A pupil location method used in [9] is to find the inner and outer boundary of the iris. if two circle object being detected with different radius have the same center point in the given image, it will be seen as iris. Then, the location of pupil could be easily calculated to be the center within inner boundary of the iris.

2.4 Comparison with Developed Solution

Detecting pupil location only with means of gradient often promises to find a center. However, this algorithm is not so successful to eyes where there is low contrast between iris and sclera. This error appears more significantly under bad lightning condition and when the device is not closed enough to the patient. Another problem in this algorithm is that it fails to consider concentrated light reflected by the eye. This could result in extreme pale point within pupil and iris area, which has negative impact on this algorithm. In addition, bold eye lashes also increase noise for this algorithm, and when device is not closed to the patient, this algorithm very often sees lashes as center of pupil.

In the developed solution, Circle Hough Transform algorithm and means of gradient algorithm are combined. Circle Hough Transform gives more precise result than means of gradient algorithm. However, it is highly possible for Circle Hough Transform algorithm to fail to detect the existence of iris circle within eye image. Therefore, with the combination of the two algorithms, result of detected pupil location should be more precise than using means of gradient algorithm only, yet more promising for users to get results than using only Circle Hough Transform.

Chapter 3

Data Required

3.1 Data Requirement and Obtain

Date being used in this system are images and integer. Because this software uses iPad backwards camera to detect face and pupil, it allows only image obtained from this camera. Images are captured by frame and all the detecting algorithms run in real-time. Before user choose to save the image, there is only one data transaction in this system, which is images captured by the camera being sent to detector component. Each pupil location detected by the algorithm is in the form of an OpenCV data type called Point which contains two integers. Data transactions in the system other than above circumstance are in the form of Object ResultData and ResultList created by the developer. Each one ResultData contains one image, and some integer objects representing pupil locations. ResultList is an array of ResultData.

Testing data are images obtained from BioID dataset, which is an online open source dataset that provides data for computer vision related research. In this dataset, images containing human faces and corresponding pupil locations in each image are provided.

3.2 Ethical Use of Data

This project is related to pupil detection in computer vision area. It does require to use real human data which is face images for testing its performance. However, in this project, free source datasets are used. These are datasets that are created in support of researches in computer vision related area, and have obtained approval from donators to use their face images in those research area and distribute freely in public domain. While this project takes great consideration of ethical use of human data, Research ethics approval from the university is not needed. The dataset used in this project is BioID dataset.

Chapter 4

Design

4.1 Design Overview and Changes being made from Design Stage

The system is designed into two main parts. One is detecting part and the other one is presenting part. Detecting part is mainly for running the detector. User should give instructions to their patients asking them to look forward, to left, right, up, and down. For each instruction, user is able to take a picture by the SNAP button on screen to save the data, which includes the frame image and detected pupil location in each frame.

After examining all the directions user is interested in, user can choose OK button to enter present part in order for user to review all the images, pupil location data, and deviation results. All the results are divided into three sections which are left eye, right eye and both eyes. In each section, user is able to choose deviation in which direction to review.

In previous design stage, the system is designed to ask user to either import image or take picture with iPad backwards camera, and give the detected pupil location by processing imported image. However, if allowing user to import images, image quality such as resolution could not be ensured which could result in bad performance in detecting pupil location and inconsistency in result data. Therefore, in response to solving this problem, design was revised. The system now detects pupil location in real-time with iPad backwards camera and shows results on screen as well. User can choose to snap the camera screen while they think appropriate. At the same time as user snap the camera screen, image and pupil location will be saved for later calculation of deviation.

Using image and result taken from eye facing frontal position as basic standard, main program will calculate deviation in four directions, which are to left, right, up and down. Therefore, frontal position is compulsory while other positions are optional.

Considering using this system in reality, patient, namely people being shot for detecting pupil location, and user, namely people using the system to shot, might change positions while using the system. This means, for example, when taking image of frontal position, patient and user might be in distance of 50 cm. When taking image of left position, their distance changes to 1 meter in the same round. This could lead to error in result of pupil locations. In order to solve this problem, pupil deviation is calculated according to size of detected eye, scaling the data when necessary.

Brief overview of old design could be found in Appendix F. Design being discussed here is the revised one.

4.2 Hardware Being Used

This project requires a hardware of iPad to install and run the system. iPad backwards camera is used for capturing images. This is a relatively cheap and widely-used device compared to other head-mounted pupil detection devices.

4.3 Programming Languages

In this project, two programming languages are used. One is Objective-C, which is for IOS development. The other is C++, which is used for interacting with OpenCV library and other utility use.

4.4 User Interface Design

- The Detector

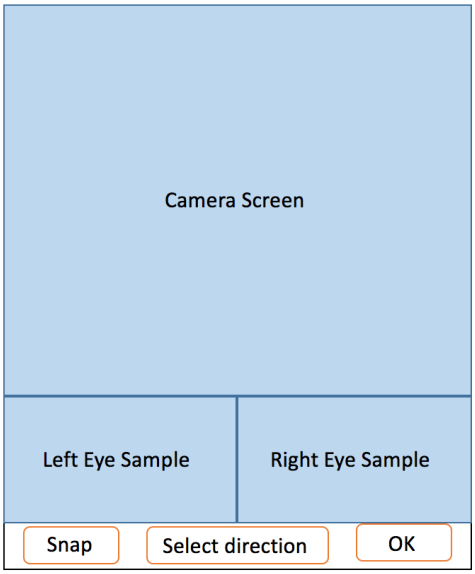


Figure 4.1 Detector UI Design

The detector interface design is shown as Figure 4.1. The Detector interface mainly contains three parts. The top part is the camera screen where frames captured by the camera and the pupil location results detected by the algorithm are presented. The bottom part contains three buttons. Snap button is the button for taking a picture. Select direction button generates a drag down menu, allowing user to select the eye direction they choose to detect. OK button is the one allowing the user to go to the present page. Middle part is where cropped images of two eyes are present. User could check whether image is clear or whether the eye in the image is closed from here.

- The Present Page

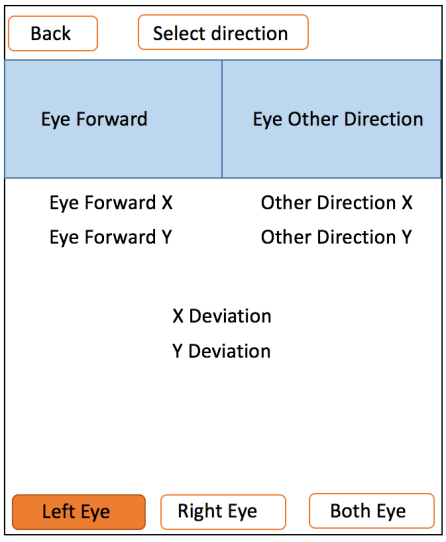


Figure 4.2 Present Page UI Design: Left Eye

The present page interface is shown in Figure 4.2 and 4.3. The bottom part contains three tabs for user to switch between results that they are interested in. Figure 4.2 shows the present page for left eye. Present page for right page is the same as this one. There is a button to choose which direction the eye is looking to. In this present page, it focuses on showing all the collected data including the detected pupil location and the eye images that was taken in detector. In eye forward section in Figure 4.2, it shows image of left eye looking forward. Below is the X and Y coordination of detected pupil location. The right part is image of left eye looking to other directions. The direction could be chosen from the button above. The eye forward image will be shown when user enter this page if available. The eye other direction image will be presented after user has chosen the direction they would like to look into. After user choosing the direction, the deviation of the two detected will be calculated and shown below.

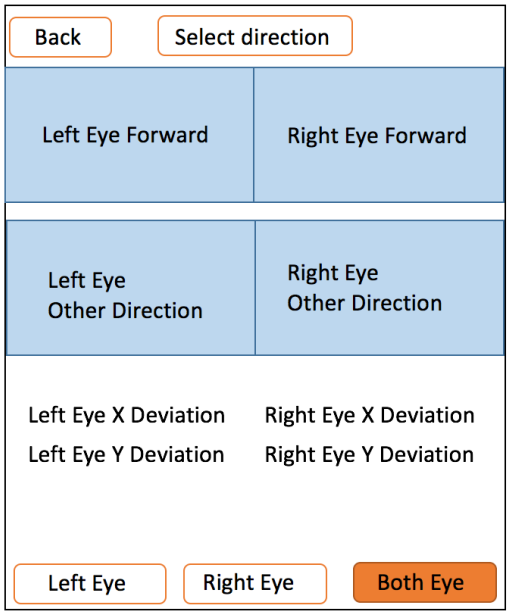


Figure 4.3 Present Page UI Design: Both Eye

The present page for both eye is little different from present page for left and right eye. The two image of eyes looking forward are default to be shown on screen if available. Similar to present page for left and right eye, after user choosing the direction they are interested in, the images of both eye looking in that direction will be shown below the default images, and pupil deviation for both eyes is calculated and presented as well.

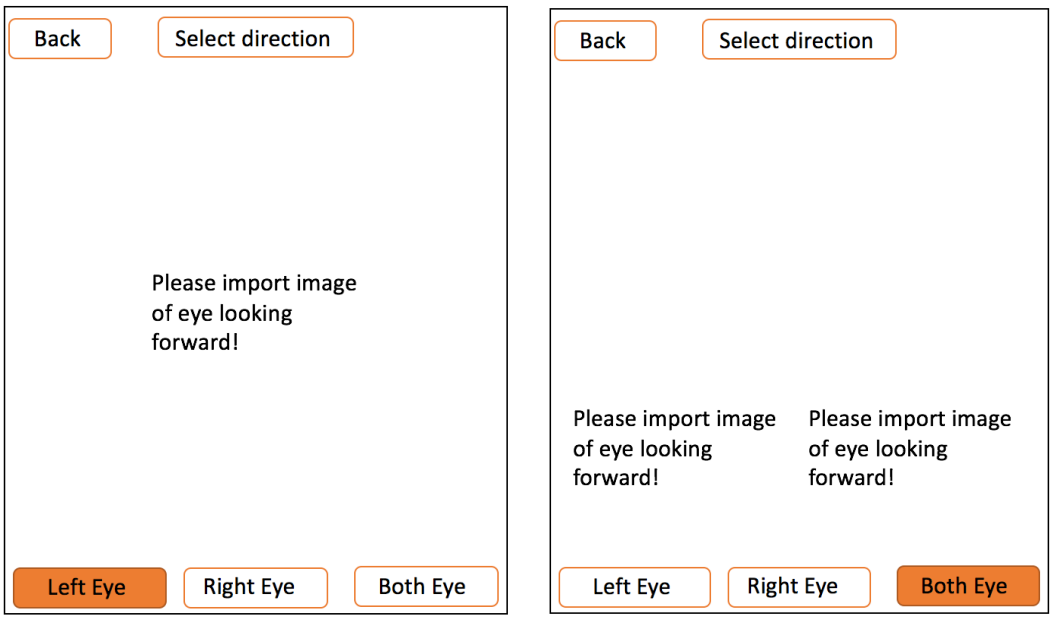


Figure 4.4 Present Page UI Design: Error Message of NO frontal image

In Figure 4.4, it shows the user interface for present page when the user did not take any image of eye looking forward. It should show the error message and no other data will be shown before the user takes the image of eye looking forward. The user interface of present page allows the deficiency of images in any direction except for image of eye looking forward. If image of eye looking in one direction is missing, when user select to see the result of that direction, it will tell the user that he did not import it. This user interface is shown as Figure 4.5.

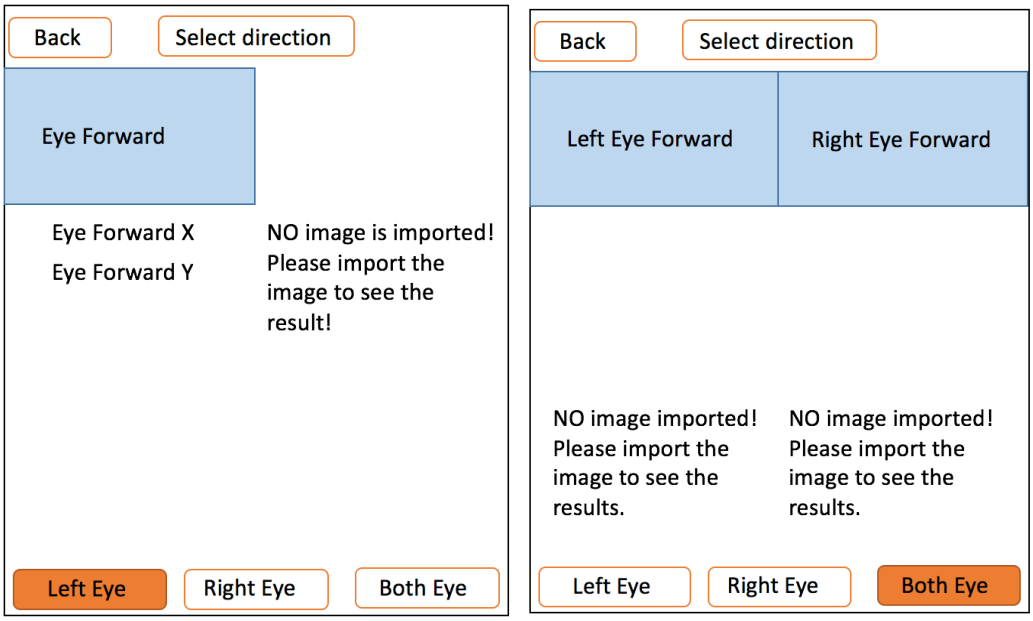


Figure 4.5 Present Page UI Design: Error Message of NO other direction image

4.5 Interface Design

Apart from the user interface discussed above, there are several interfaces between different parts within this system.

First of all, there will be an interface between the iPad backwards camera and the detector. Data being passed through this interface are frames captured by the camera. For each captured frame, the detector will then process it.

Secondly, after the detection, the detector should pass all the images and result data to the main program. Therefore, there is an interface between detector and the main program. Data being passed between the two are images and pupil locations, namely OpenCV data type points.

Last but not least, when user chooses to go to the present page, all the data saved and the deviation calculated in the main program should be passed to present UI. Therefore, an interface is built between main program and present UI. Data being transferred are images, pupil locations, and deviation results.

A data flow diagram is shown as below in Figure 4.6.

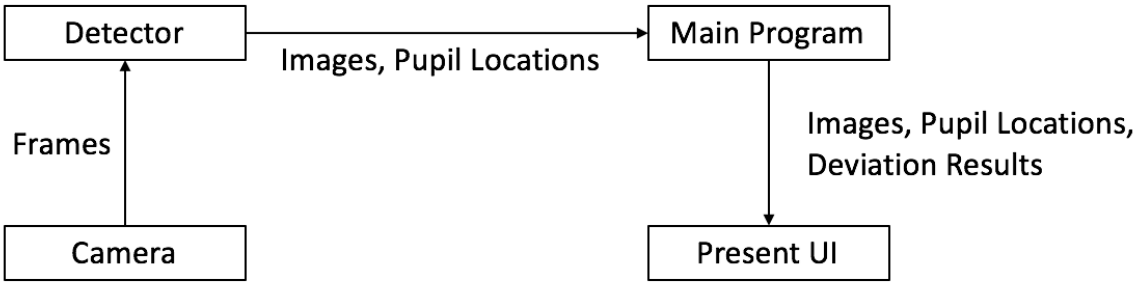


Figure 4.6 Data Flow Diagram

4.6 Use Case Diagram

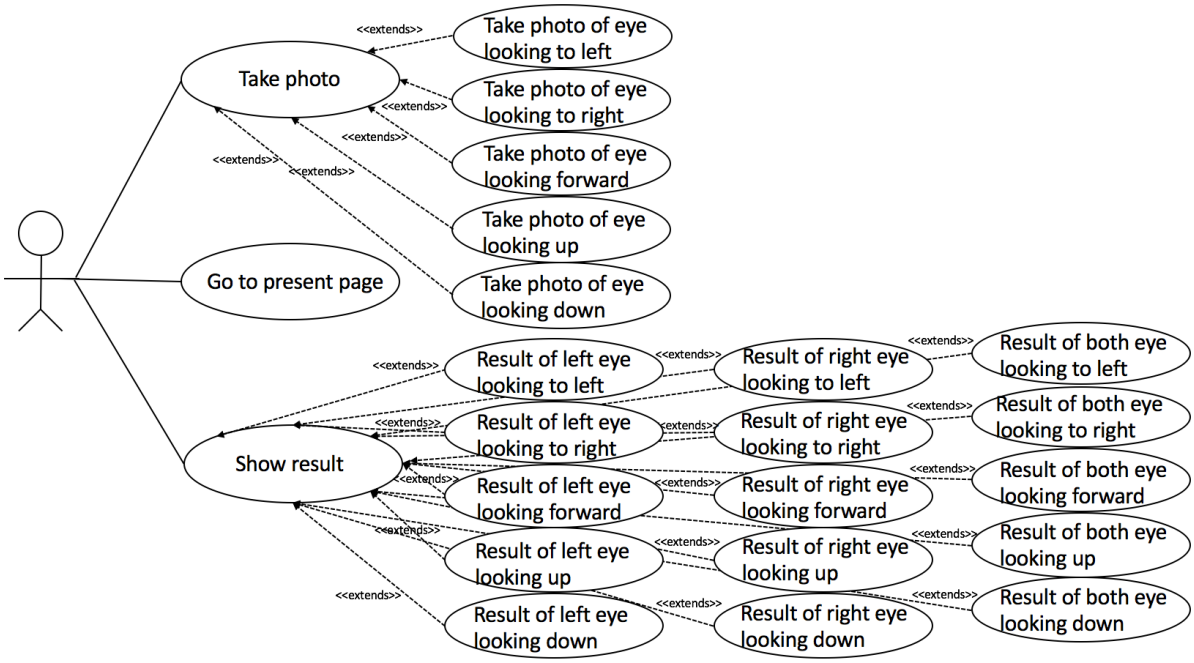


Figure 4.7 Use Case Diagram

4.7 Interaction Chart

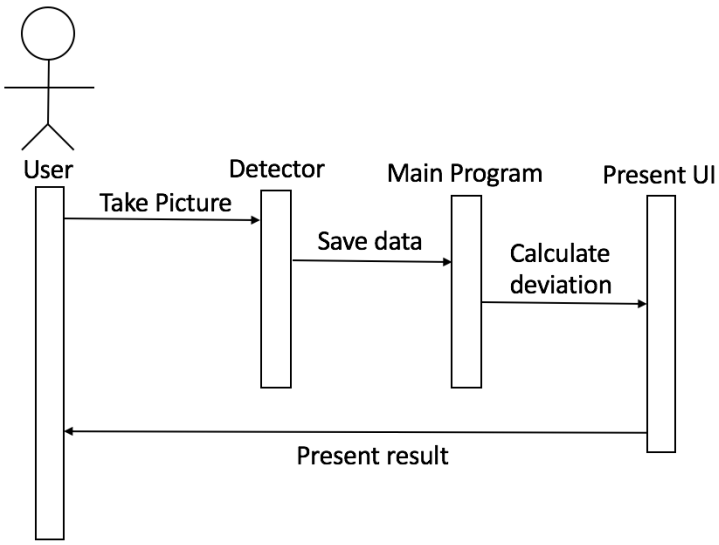


Figure 4.8 Interaction Chart

4.8 Algorithm Design for Each Component

Algorithm being used in this system is split into two main parts. One is the detector that contains the continuous loop for detecting. All the detections are completed within the detector. The other one is the main program that processes the data obtain from the detector.

4.8.1 Algorithm for Detector

The detector is essentially a loop for applying face detection, eye detection and pupil detection algorithms in a hierarchy way. The form of design is described as Figure 4.9 below.

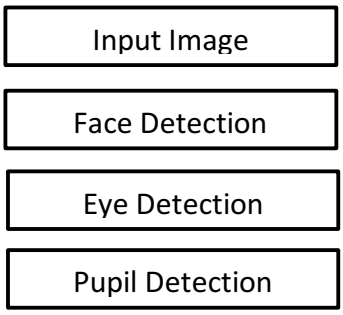


Figure 4.9 Detector Loop: Logical Design

The first step is to retrieve one image from camera as input image. This will automatically begin when the camera is turned on before entering this loop. Once image is input, detector first will start to apply face detection algorithm, trying to detect face object. Once one face object is detected, it will start eye detection. For every detected eye object, pupil detection will only take cropped image of detected eye feature as input instead of applying the algorithm to the whole image. This is for the purpose of improving calculating speed and reduce noise.

The detector takes image with its input and prints the detected pupil location on screen. It also saves the pupil location data in real-time and associates this data with main program. This association part will be discussed in the design of main program.

4.8.1.1 Algorithm for Face Detection and Eye Detection

Both face detection and eye detection uses Haar-like Feature-based Cascade Classifier brought by Paul Viola and Michael Jones [11]. This has achieved great successful in face and eye features and provides excellent result in previous researches. Moreover, for the purpose of improving result, left eye and right eye are detected separately with two different cascade classifier.

There should be some preprocessing to the image before applying the actual detection. The image should be converted into grayscale image and its contrast should be increased. As for

the actual detection, OpenCV provides the API for detecting face and eye features with haar-like cascade classifiers.

For eye detection in particular, it should be applied on the detected face feature. For the purpose of ensuring accuracy, the detected face feature should be divided into two parts with each part containing one eye feature. This means that left eye detection should be applied on left side of the face and right eye detection should be applied on right side of the face.

4.8.1.2 Algorithm for Pupil Detection

Pupil detection consists of two algorithms to obtain a better result. The two algorithm is applied at the same time and they give two detected locations by returning two Points. These two points will then be calculated to get an average detected pupil point.

a) Circle Hough Transform

   The first algorithm being used will be Circle Hough Transform. This algorithm is used for detecting the circle in the imported image. In this case, the imported image will be the detected eye image and the circle being detected will be the iris. Pupil location will be at the center of detected iris which is the center of the circle being detected by Circle Hough Transform. This algorithm will return detected pupil location point.

   Reducing image noise will be helpful to Circle Hough Transform. Therefore, a median filter will be applied in order to smooth the image first.

   After each circle is detected, an average points of all detected circles centers will be calculated and this is the return value of first pupil detection algorithm being used.

b) Means of Gradient

   As is discussed in Chapter 2, pupil center is detected by calculating the relationship between a possible center and the orientations of all image gradients. In general, in this part, the algorithm is to realize the formula $\overrightarrow{c^*} = \underset{\overrightarrow{c^*}}{\arg max} \left\{ \frac{1}{N} \sum_{i=1}^{N} \left(\overrightarrow{d_i^T} \overrightarrow{g_i} \right)^2 \right\}$.

To achieve this goal, this part will be divided into two parts. The first is to calculate $\frac{1}{N}\sum_{i=1}^{N}\left(\overrightarrow{d_i^T}\overrightarrow{g_i}\right)^2$ for each possible point and the second is to find the point with maximum value. For the first part, image gradients should be calculated. In [7], it is also advised to use a threshold to normalize the gradient. Inspired by [12], when calculating $\overrightarrow{d_i^T}\overrightarrow{g_i}$ which shows the orientation of the two vectors, the opposite directions are not needed. Therefore, those with negative dot products could be turned to zero.

Because the algorithm is applied in real-time, the location of detected by this algorithm will vary between each frame even if the eye direction remains unchanged. Therefore, an average point will be calculated which is the return point for this algorithm.

4.8.2 Main Program

Main program is mainly responsible for user interaction and data process. A logical design for the main program is provided as Figure 4.10 below.

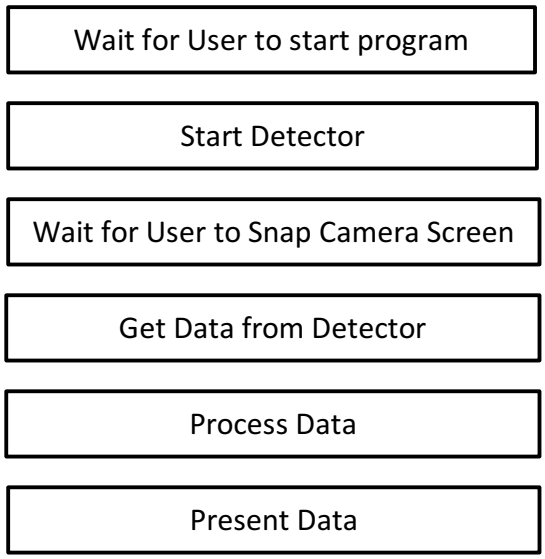


Figure 4.10 Main Program: Logical Design

In main program, it first of all waits for user to start. It then tells the detector to start capture. Main program interacts with detector by getting data from detector when user snap the camera screen. These data include an image, two eye boundary data, and two pupil location data. The image is cropped according to the two eye boundary. The cropped eye image along with the pupil location for that eye will be used to initialize a result object. All the result objects are

then saved into a list object. Data transfer between view controllers are all list objects. In presenting view, the original data including the image and pupil location being detected will be present after user choosing to along with the deviation result that is calculated.

The calculation of deviation results will also be completed in main program. Considering that distance between device and patient might change which results in different size of eye image, deviation result should not be calculated by plain subtraction. Because deviations in all directions are based on pupil location in frontal direction and the direction being selected, the data in that direction will be scaled to match the scale of frontal direction. An illustration is presented below in Figure 4.11.

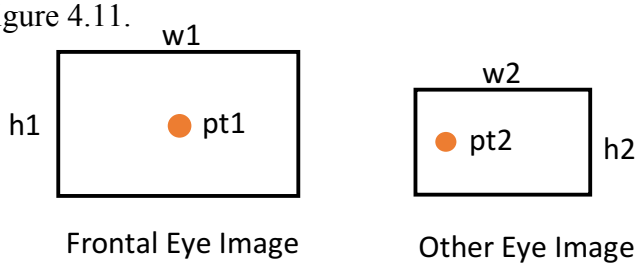


Figure 4.11 Main Program: Deviation Calculation

As in Figure 4.5, x of pt2 after scaling should be $\frac{w1}{w2} \times pt2.x$ and y of pt2 after scaling should be $\frac{h1}{h2} \times pt2.y$.

The pupil location being presented on screen are location in X and Y coordination according to the eye image instead of the full image being captured by the camera. This helps to eliminate the error that might rise because of patient's head movement or user's movement in horizontal coordination.

4.9 Testing Design

Testing will be divided into two parts as the system contains two main parts, namely the detector and the main program.

The main program is responsible for communicating with the detector, processing the raw data retrieved from detector, and presenting the final results. Therefore, testing of main program should be the first step in testing part because if the main program did not work, testing of detector would be pointless. Test cases for the main program are presented as Table 4.1.

Scenario	Test Step	Expected Result
Verify that tabbing snap button when pupil is detected, data is saved.	Apply the detection algorithm to the frame image and tab the snap button when pupil location is detected	Data of pupil location is saved
Verify that tabbing snap button when pupil is not detected, data will not be saved.	Apply the detection algorithm to the frame image and tab the snap button when pupil location is not detected	Data of pupil location is not saved
Verify that tabbing snap button when pupil is detected, the data being saved is indeed the data being returned.	Apply the detection algorithm to the frame image and tab the snap button when pupil location is detected. Compare the saved data to the data printed on screen.	The two data are identical.
Verify that user can snap more than one time in all directions	Select each direction and snap 3 to 5 times	New result overwrites old ones every time user tabs snap button
Verify that data being retrieved is successfully sent to presenting page	Apply the detection algorithm to the frame image and tab the snap button. Then go to presenting page	Result is shown on presenting page
Verify the data being retrieved is saved and presented successfully according to the eye direction	Apply the detection algorithm to the frame image and tab the snap button. Repeat the step in different eye directions. Then go to presenting page	Result is shown on presenting page according to the eye direction

Table 4.1 Test Case for Main Program

The most essential part to test is testing the accuracy of pupil location that is detected. Because two eyes are detected separately, pupil location for two eyes should also be tested separately as well. In addition, the distance between device and patient could influence the accuracy of detected of pupil location as well. Therefore, distance should also be a testing part to be addressed. Other factors influencing accuracy of pupil detection are lighting condition, human

races, wearing makeup or not, wearing glasses or not, and device differences. Therefore, the testing will be split into three main groups which are people with light skin color, people with dark skin color, and Asian people. Within each group, first it will be detected under good lighting condition with no makeup or glasses. It will then be tested under bad lighting condition keeping other conditions unchanged. All the factors should be tested one by one like that. Within each group, gender of people should be half female and half male to show no bias on either gender. For each group, test cases will be rather similar. Therefore, here only test cases for one group are listed as below Table 4.2.

Scenario	Test Step	Expected Result
Verify the accuracy of detected face feature in good lighting condition	Apply face detection algorithm to the frame image in good lighting condition and get the result	Result face feature is the desired face feature
Verify the accuracy of detected eye feature in good lighting condition	Apply eye detection algorithm to the frame image in good lighting condition and get the result	Result eye features are the desired eye features
Verify the accuracy of detected pupil location in good lighting condition when patient looks forward	Apply the detection algorithm to the frame image in good lighting condition and get the result	Result location appears at center of pupil
Verify the accuracy of detected pupil location in good lighting condition when patient looks left	Apply the detection algorithm to the frame image in good lighting condition and get the result	Result location appears at center of pupil
Verify the accuracy of detected pupil location in good lighting condition when patient looks right	Apply the detection algorithm to the frame image in good lighting condition and get the result	Result location appears at center of pupil
Verify the accuracy of detected pupil location in	Apply the detection algorithm to the frame image in good lighting condition and get the result	Result location appears at center of pupil

good lighting condition when patient looks up		
Verify the accuracy of detected pupil location in good lighting condition when patient looks down	Apply the detection algorithm to the frame image in good lighting condition and get the result	Result location appears at center of pupil
Verify the accuracy of detected face feature in bad lighting condition	Apply the face detection algorithm to the frame image in bad lighting condition and get the result	Result face feature is the desired face feature
Verify the accuracy of detected eye features in bad lighting condition	Apply the face and eye detection algorithm to the frame image in bad lighting condition and get the result	Result eye features are the desired eye features
Verify the accuracy of detected pupil location in bad lighting condition when patient looks forward	Apply the detection algorithm to the frame image in bad lighting condition and get the result	Result location appears at center of pupil
Verify the accuracy of detected pupil location in bad lighting condition when patient looks to left	Apply the detection algorithm to the frame image in bad lighting condition and get the result	Result location appears at center of pupil
Verify the accuracy of detected pupil location in bad lighting condition when patient looks to right	Apply the detection algorithm to the frame image in bad lighting condition and get the result	Result location appears at center of pupil
Verify the accuracy of detected pupil location in bad lighting condition when patient looks up	Apply the detection algorithm to the frame image in bad lighting condition and get the result	Result location appears at center of pupil
Verify the accuracy of detected pupil location in	Apply the detection algorithm to the frame image in bad lighting condition and get the result	Result location appears at center of pupil

bad lighting condition when patient looks down		

Table 4.2 Test Case for Detector

4.10 Investigation of Hypothesis

The investigation in this project is mainly about the accuracy of pupil location result. This could be affected by several facts. These facts should be tested one by one without affecting each other.

- Lighting Condition

  The lighting condition ideally should have no or little affection on the performance of the algorithm. Therefore, for eyes in all direction, it should be tested under different lighting condition. Lighting condition could be separated into 4 types, which are uniform daylight, concentrated light, color tone artificial lighting, and warm tone artificial lighting.

  The anticipated result should be that, for the same image, detected pupil location remains unchanged in all lighting conditions.

- Glasses or Not

  Glasses are a major factor that affects the performance of this algorithm. There are two facts that might affect the detection of pupil location, which are the reflection of light by the lens and the frame of glasses.

  As for the fact of reflection of light, test cases could be patient not wearing glasses, patient wearing glasses without reflection, patient wearing glasses with slight reflection, and patient wearing glasses with strong reflection. During the whole process, eye direction should remain unchanged.

  Anticipated result should be that pupil location is detected for each time and remain unchanged.

- Distance between Device and Patient

Change of distance between device and patient results in different scaling. Such different leads to errors in the location data. A method in design has been proposed to solve this problem. The error that remains after applying this method should be tested and evaluated.

However, the distance between device and patient raises another potential problem that the detector might fail to detect the pupil. For distance that is long enough, the detector might even fail to detect eye feature or face feature. Therefore, the extreme distance of successful detection should be tested as well. Distance is set initially to be 20 cm and increases 5 cm for each time up until distance is 1 meter.

- Contrast between Iris and Sclera
  Iris color varies between different people. Light colored iris leads to low contrast between iris and sclera. This contributes to low contrast in pixel level, which is challenging for such algorithm to find the accurate pupil location. Therefore, the error it performs among different iris colors should be tested as well.

  It is ideal that the algorithm does not give bias on darker or lighter iris color.

Chapter 5

Realization

5.1 Introduction

In this chapter, all steps of implementation throughout this project will be introduced. This software has been implemented on Apple MacBook Pro with 8 GB of RAM and Intel Core i7 @3.1 GHz Processor on Operating System of MacOS version 10.12. All codes was done on Xcode version 8.3.2.

5.2 Implementation of Each Component

During the implementation, use of all the open APIs provided by OpenCV in first three components is under the instruction on OpenCV website with some changes in parameters. Main APIs being used are provided in text below. Reasons for changing those parameters are provided. Codes in the last component namely pupil detection by means of gradient are implemented mainly according to [7] with some advice in [12] for improvement.

5.2.1 Facial Detection

Facial detection is the first part being implemented in this software. Face cascade classifier of haar-Like cascade has been used in this part. OpenCV predefines Haar cascade classifier, making it easier to implement.

First of all is to load the classifier. The classifier being used here is *haarcascade_frontalface_alt2*. It could be loaded by code below.

```
NSString *faceCascadePath = [[NSBundle mainBundle] pathForResource:faceCascadeFile
ofType:@"xml"];
faceCascade.load([faceCascadePath UTF8String]);
```

The classifier then could be used in the face detection method after being loaded. The face detection algorithm is embedded in the method called processImage with each frame image as input. In face detection method, firstly, the frame image should be pre-processed before using

the classifier to detect face features. The frame image is converted into grayscale, and, by applying OpenCV method *equalizeHist*, the Histogram is equalized to improve contract in the image. After these prerequisite, the frontal face cascade classifier is used by code below.

```
faceCascade.detectMultiScale(grayscaleFrame, faces, 1.1, 2, HaarOptions, cv::Size(60, 60));
```

Here, in this line of code, "grayscaleFrame" is the input image, which has been turned into grayscale and being equalized its histogram. The "faces" stores an array of all detected face features. It is suggested by [13] that, by accessing the array by index, all the detected faces could be used to reset area of interest for detecting eye features. However, in my implementation, Xcode fails to find method in OpenCV of setting area of interest. Therefore, in implementation of this software, within each face that is detected, the method of eye detection is called without resetting area of interest beforehand. For the purpose of improving the accuracy of eye detection, with each eye detection method applied, the result will be tested whether it is within the desired side of face. Therefore, after each face is detected, it is divided into two equal parts, namely the left side of face and right side of face. This is achieved by simply mathematical calculation which can be found in appendix B.a in line 14 to 25. The whole face feature is in the rectangle of pt2 and pt3. Left side of face is in the rectangle of pt4, pt1 and right side of face is in the rectangle of pt2, pt3. The illustration of it is provided in figure 5.1 below.

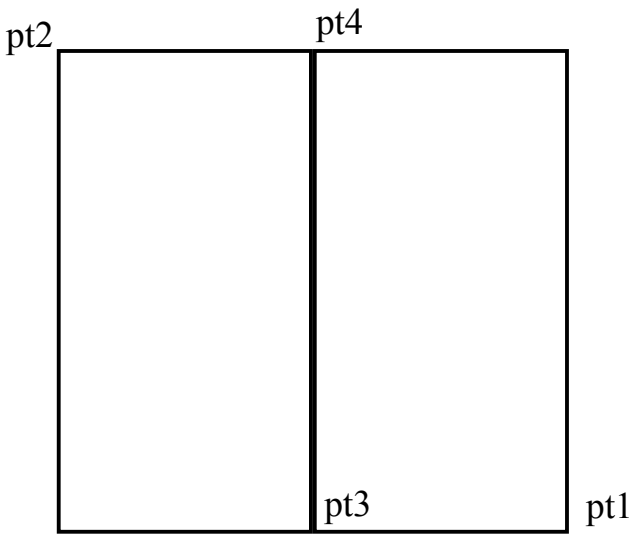


Figure 5.1 Face Splitting

In addition, for each face object that is detected, it draws a square indicating that this is the face feature being detected. A screen shot could be seen in Appendix C.1.

5.2.2 Eye Detection

The eye detection is implemented in the similar way as face detection except that eye detection requires to check detected eye location is on the face in order to reduce error. By passing the id name from face detection method, eye detection method is able to know which side it is looking into. This checking step could have been waived if the area of interest could be reset to only the detected face features from previous stage. However, such resetting method fails to execute which has been discussed in last section. Therefore, a checking method has been created to accomplish this requirement.

The first step in this section is to load the classifier which is the same as that in face detection. In eye detection section, because there are two eyes appearing symmetrically on face, two eye cascade classifiers will be used. They are *haarcascade_lefteye_2splits.xml* and *haarcascade_righteye_2splits.xml*. The two different cascade classifier files will be passed from face detection method to eye detection method along with id name that indicates which eye it is looking into.

The second step is to check whether the detected eye object is within the desired area. This part is implemented as a simple mathematical method that checks both the upper left point of detected eye area and lower right point of detected eye area are within the desired side of face. When right eye is being detected, it should check right eye is in the area of right side face, and similarly, when left eye is being detected, it should check left eye is in the area of left side face. An illustration is provided in figure 5.2 where checking right eye on right side of face is provided as an example. The desired face area is rectangle ptA and ptB and the detected eye area is rectangle pt2 and pt1.

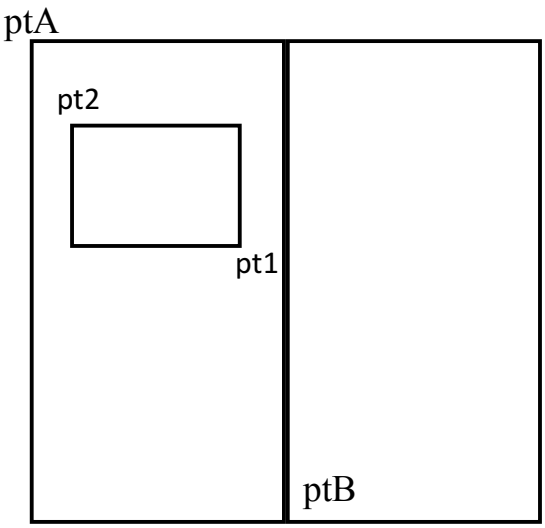


Figure 5.2 Face Splitting

After each eye feature is checked and proved to be valid, it then crops the image to be eye area only. The brow will be removed by increasing y of pt2. This detected eye feature is then used to detect pupil location by calling pupil detection method for detecting pupil location. A screen shot could be seen in Appendix C.2.

5.2.3 Pupil Detection

This method contains two parts that use different algorithm to detect location of pupil with provided eye image. The two algorithms are used in order and return two point values. The two values are calculated to get an average point.

5.2.3.1 Circle Hough Transform

Circle Hough Transform algorithm is the first algorithm used for detecting pupil location. Pre-processing of the imported image will be applied first to improve the performance of this algorithm. This is achieved by smoothing the image with a median filter. The *HoughCircles* method embedded in OpenCV is then used to detect circles with Circle Hough Transform. The code is provided as below:

```
cv::HoughCircles(image, circles, CV_HOUGH_GRADIENT, 1, 8, 100, 30, 5, 30);
```

The parameter image is the imported image of eye feature. The parameter named *circles* is the output array that contains all the circles it detects. *CV_HOUGH_GRADIENT* and *1* are two

default values that remain unchanged from the OpenCV tutorial. All the rest of parameters are the configuration used when detecting circles. The minimum distance between two detected circles is set to be 8 which is small. This is consistent to the fact and reduces the error that might rise from reflection of light. In addition, because this method is mainly used for finding the boundary of iris which is normally relatively small, the maximum radius of circle being detected is set to be 30. This reduces the error when it might see the shape of eyelid as part of a circle if one's eye is round shaped.

After applying the method, all the circle centers detected are considered as the potential pupil locations and an average of these points is calculated. It then draws the circle and center of circle on screen and saves the data as output of this method.

The outcome of this method could be found below in Appendix C.3.

5.2.3.2 Means of Gradient

As is discussed in previous chapters, this method is mainly for realizing the formula $\overrightarrow{c^*} = \arg\max_{\overrightarrow{c^*}} \left\{ \frac{1}{N} \sum_{i=1}^{N} \left(\overrightarrow{d_i^T} \overrightarrow{g_i} \right)^2 \right\}$ (1). Before actually getting into this formula, it should go through several pre-processing steps.

To begin with, the image gradient in both horizontal and vertical orientation is computed. This is achieved by rewriting the method used in Matlab into C++ codes which is brought out by [12]. With gradients in the two directions, the gradient magnitude is computed by $sqrt\big((gX * gX) + (gY * gY)\big)$ for all locations.

To continue with, for the purpose of improving computing speed and saving space, image gradients are normalized with a gradient magnitude threshold value. All the gradients where the gradient magnitude is below the threshold are removed. This value is brought out by [7]. The image is smoothed afterwards by a Gaussian filter and inverted.

Next is to calculate according to the formula for all the locations where gradient now is not zero. In [7], it is suggested that, by applying a weight wc for each possible center, the error occurring from low contrast between iris and sclera could be largely reduced. Therefore, the

actual formula being computed here is $\sum_{i=1}^{N} w_c \left(\overrightarrow{d_i^T} \overrightarrow{g_i}\right)^2$. The code for this part is in Appendix B.d from line 78 to 103.

The last step is to find out the location which has the largest value as suggested in the formula (1). This could be achieved simply by OpenCV API `minMaxLoc`. The center location with maximum value is considered to be the darkest point in the given image which is then considered to be the pupil location. Therefore, the output data is the point being found here.

The outcome of this method is shown below in Appendix C.4.

5.3 Encountered Problems and Their Solutions

Several problems have occurred during the implementation.

In eye detection, as has been discussed above, eye detection could not be directly applied within the area of face that has been detected in previous stage. Therefore, another method is created to mathematically check each detected eye is exactly on the desired side of face.

In pupil detection, it was previously designed to use Circle Hough Transform algorithm only for completing the task. However, during implementing and testing stage, it is discovered that Circle Hough Transform algorithm is not so reliable and it is highly possible for it not to detect any circle in imported image. Therefore, pupil detection component is then revised to use both Circle Hough Transform algorithm and Means of Gradient. The two result points are then calculated to get an average point for a better result. In Circle Hough Transform algorithm, it is very often that the algorithm sees the shape of eyelid as the shape of circle when the eye being detected is a round shaped eye, which causes error in result. To address this problem, the maximum radius of potential circle is set to be 30. In Means of Gradient, in order to improve calculating speed and save space, the gradient where gradient magnitude in this location is below the threshold is removed as proposed by [12].

In the main program where it interacts with the user, it shows when testing that the detected pupil location that should already be marked on the frame does not appear very time user saves the frame. It is intended to solve this problem by drawing the point circle afterwards on the

image and presents it onto the UI. However, it gives the similar result as directly saving the frame that the point circle does not promises to appear.

5.4 Testing Implementation and Results

Testing is conducted by simulating the situation when the software is used in real life. It is conducted by directly using the software on device and pointing it to the images obtained from some open source datasets which provide image and eye location data. By comparing the pupil location detected by the software with the location provided by the dataset, the error could be calculated.

The dataset being used here is BioID dataset [14]. It is an open source dataset that is aimed at providing testing data for computer vision related researches. All the data within this dataset have obtained approval from the donator for such researches.

Because the pupil location provided by BioID is X and Y coordination to the whole image while the software originally returns X and Y coordination to the image of the eye, the data are further processed. An illustration is provided as below in Figure 5.3 to explain the process of these data.

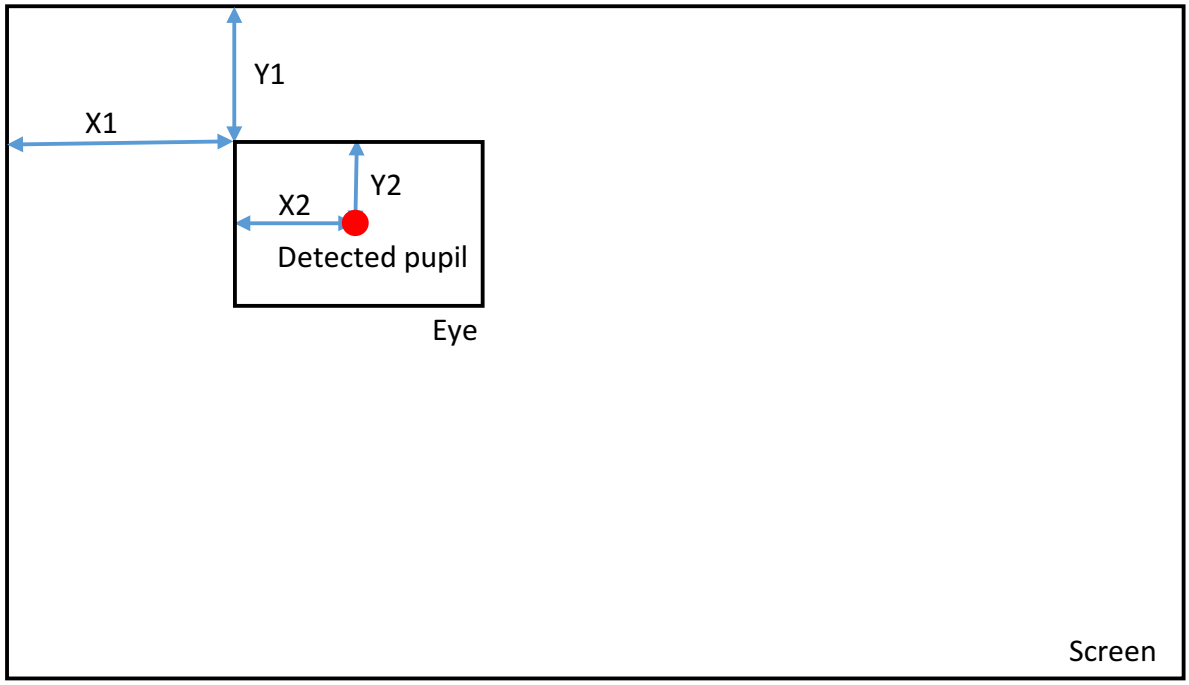


Figure 5.3 Data Process in Testing

Therefore, as is shown above in the figure, when testing, the X of detected pupil location should be X1 + X2 and Y of detected pupil location should be Y1 + Y2. Additionally, when testing the upper left corner of the screen should coincide with the upper left corner of the image being tested.

The testing is implemented in a hierarchy way for different components. It is mainly achieved by drawing the detected area and printing data on screen. Testing data used in testing stage are open-source datasets that are used for iris and pupil detecting research. These datasets have obtained approval from the data provider to use their face images in iris and pupil detecting related researches. These datasets provide image as well as mathematical data of accurate pupil location in that image. The image is used in this project as the import image data and the mathematical data of pupil locations are used for calculating the error in the performance of this software.

Testing of face detection is simply implemented as applying face detection component and drawing a rectangle on screen around the detected area. In this part, there is no mathematical calculation because this software requires to detect one face at a time only, and the success of face detection is that the algorithm finds the correct face. Therefore, there will no data recorded by the algorithm. It could simply be tested by checking whether the rectangle is around the face area.

Testing of eye detection is similar to that of face detection. Two distinct rectangle should be drawn around the two detected eye.

Testing of pupil detection algorithm for both eyes is completed together. It is designed to draw the two pupil locations on screen and print the two data on screen. Therefore, by comparing the data with the one provided by the datasets, errors could be calculated. There is an example table of results taken from the testing stage in Table 5.1. Size of the test example is 10. The full table could be found in Appendix A, where testing data for both eyes in four directions are included. In Appendix A, other testing data according to test cases in Chapter4 (4.5) are also included.

Left Eye Forward				**Right Eye Forward**			
X	Y	Error in X	Error in Y	X	Y	Error in X	Error in Y
238	109	0	2	156	115	4	2
228	110	4	0	158	109	4	1
223	102	0	1	153	100	1	2
211	95	1	0	150	95	1	1
226	110	1	2	145	111	0	0
213	114	0	1	134	113	0	1
225	124	5	4	155	121	3	5
223	120	5	3	151	120	4	3
247	89	1	3	165	92	3	1
230	120	0	2	142	126	3	1
Average Error		1.7	1.8	Average Error		2.3	1.7

Table 5.1 Test Result for Eye Looking Forward

Chapter 6

Experiment

6.1 Experiment Design and Setting

Experiments being conducted is according to the hypothesis being brought out in Chapter 4 (4.9). Therefore, there will be 4 different experiments to consider about.

- Lighting Condition

  Because the software is tested by using the device to some images from dataset, the testing of lighting condition influencing software performance should be about the lighting condition of the taken picture. Three different lighting conditions will be tested, which are dark, normal, and over exposed condition.

- Glasses or not

  BioID dataset provides images with same person but both with and without wearing glasses. Therefore, experiment for this part is simply use the software to detect pupil locations in those images.

- Distance between Patient and Device

  Same as device being used in real life, the distance between device and patient could not be the same all the time. Therefore, for the same image, when testing, the distance between device and image should be adjusted and results for different distances should be recorded and analyzed.

- Contrast between Iris and Sclera

  Low contrast between iris and sclera resulting from iris with light color is another significate fact that influences the performance of the algorithm. Therefore, performance on people with light color iris should also be tested.

6.2 Experimental Results

- Lighting Condition

  Performance under normal lighting condition is the best. Error under dark lighting condition is worse than that under normal lighting condition but the three detection methods still work in such condition. Performance in exposed images is the worst in that it is possible that face feature could not be detected and therefore neither eye or pupil detection method could be trigger.

- Glasses or not

  Performance in the condition of glasses reflecting strong light is poor in that eye feature could not be detected. Therefore, pupil detection method could not be applied. This means no result is shown in this case. However, the performance when patient is wearing glasses in good lighting condition where there is no or little reflection on the glass is similar to the condition when patient does not wear glasses.

- Distance between Patient and Device

  The ideal distance between patient and device is between 30 cm to 50 cm. The lower value of distance range ensures the calculation speed, making not too slow to compute. Both lower and upper value of distance range also ensures that the camera could capture a face feature in order to trigger the eye and pupil detection method. Results taken in different distance are scaled as could be found in Appendix A.g where it gives the originally detected pupil locations and scaled locations. The first row in Appendix A.g is the standard pupil location with respect to eye image. From the table, it is shown that scaling location reduces the error. Errors appearing in table after scaling could rise from other facts such as error in algorithm itself.

- Contrast between Iris and Sclera

  The result of this case is provided in Appendix A.f. This is the result of error in detected pupil location when contrast between iris and sclera is low. Images used in this part are images of eye looking forward. Therefore, the error result could be compared with table in Appendix A.a which contains images of eye looking forward but with eye having strong contrast between iris and sclera. It is clear that the error is higher in low contrast

ones than high contrast ones. Error mainly occurs in X coordination and detected pupil location is often deviated to the crease between eyelid and sclera where it is darker.

6.3 Discussion of Results

The results show that the algorithm still has not solve the influence that lighting has on the performance. It also does not solve the problem of inaccuracy result for people having light color iris. It succeeds to reduce the influence the glasses have on the result of pupil detection.

However, this conclusion is mainly based on one database and the algorithm did not be tested in real life where normally the condition of strong light does not appear like the way it does in over exposed images. Hence, more testing should be taken before making any conclusion.

Chapter 7

Evaluation

7.1 Evaluation Criteria and Approaches

Evaluation of this project involves 8 criteria. These are under instruction of [15] but tailored to fit the evaluation of this particular project.

- Algorithmic Accuracy

  Algorithmic accuracy evaluation focuses on evaluating how accuracy the results are. This could be evaluated by analyzing the testing results. The testing results should include errors in X and Y coordination respectively. The errors are the deviation of detected pupil compared of the desired location.

- Understandability

  Understandability is evaluated on how straightforward it is to understand the following aspects. First is the software purpose and what it does. Second is the targeted users and market. The last is the functions the software provides.

  Evaluation of understandability should check the availability of detailed description of intended users and software functionality. In addition, architectural overview, with diagrams, and intended use cases should also be provided.

- Documentation

  The evaluation of documentation mainly focuses on evaluation of user documentation on the completeness, accuracy, clarity, and appropriateness.

  Evaluation of documentation should check the availability of high-level overview of the software. Whether clear, step-by-step instructions is included, whether anticipated view is provided for each stage, and whether error messages and their solutions are provided should be checked thoroughly.

- Installability

  Evaluation of installability should inspect whether it is straightforward to meet the requirement for the software to install on the target platform, and whether instructions for installation are provided.

- Testability

  Evaluation of this part includes the evaluation of testing the software. It should investigate whether unit tests and integration tests are conducted, and whether it fulfills all the test cases that are designed in the testing stage.

- Portability

  Evaluation of portability should look at all the platforms that this software could be built on and run. In this project, because an application is created, it should check the operating systems that this application could be built on, such as IOS system, Android system, or Windows system.

- Supportability

  Supportability should be evaluated by looking at to what extend the product is supported. It includes all the documents and platforms on which supports could be reached.

- Anlaysability

  This includes the evaluation of implementation architecture and how the source codes fit in the design. It should look into the facts whether code is structured into modules and those are directedly related to the design, and whether source codes are well commented.

7.2 Evaluation of Results

- Algorithmic Accuracy

  Evaluation of algorithmic accuracy is based on the test results in Appendix A. There are five tables in Appendix A which describes the pupil locations detected by the software in five directions and their error in X and Y coordination respectively. Each

table includes data for both left eye and right eye. Test results of pupil location in two coordination are provided as well as their errors. Each row is one entry for one detection action. Average errors in two coordination for each eye are provided in the last row of the table.

For all the five tables, it is clear that eye looking forward has the lowest error. This means that pupil detection when patients' eyes are looking forward is the most successful as the error is kept under 2 in average. Error for eye looking down is the highest for an average of 3.77 deviation from desired location. Error for eye looking right, up, and down are very similar. However, error for eye looking left is relatively small compared those three.

For each table, in Appendix A.a, to begin with, error is kept similar for each eye in each coordination. To continue with, in Appendix A.b, error in X coordination is higher than that in Y coordination. This is the same for Appendix A. c and A.e. In A.d, error in Y coordination is higher than that in X coordination.

The reason for condition when eyes are looking forward having the lowest error could be that in this condition, pupil is normally in the center of the eye and it is away from eyelid and eye lashes. Therefore, eyelid and lashes have less impact on the detection of pupil location. However, for other eye directions like looking left, right or up, pupil is more closed to eyelid, lashes, and even eye corner. These increase the noise for the algorithm especially for dark shade lashes. Shadows casted onto crease and eye corner also increase pixel grey values which algorithm might consider to be possible location of pupil location. However, it is different when eye looks down. Error for eye looking down mainly could raise from the fact that existence of the pupil and iris is not clear for hand hold iPad backwards camera. Most part of the pupil and iris are cover by the upper eyelid and eye lashes and for hand hold device it is not possible to be closed up enough to capture the eye. However, test result of Appendix A.b might not be able to represent the overall situations and their errors of detected pupil location in that the most of the images being tested are images taken when lighting is in one direction. This could account for the over positive result for eye looking left because when lighting is directed instead of being uniform, the shadow casted on face is in one direction and this might give favor for pupil detection in that direction.

For Appendix A.b and A.c, the errors are both higher in X coordination than that in Y coordination. That might rise from the fact that when eye is looking to left or right, pupil and iris are more closed to eye corners, and shadows casted in eye corners effect more in X coordination. In fact, during testing, it is shown from the screen that the detected pupil location is indeed more closed to eye corner.

For Appendix A.d, errors are higher in Y coordination than that in X coordination. The reason is highly possible to be similar to the reason mentioned above. However, the issues affecting accuracy of test result when eye is looking up might not be the shadow but the eye lashes. This is also supported by the fact that detected pupil detection appearing on screen is more closed to upper eye lashes.

For Appendix A.e, errors are higher in X coordination than that in Y coordination. This is because, in Y coordination, the majority is covered by upper eyelid. Upper eyelid, when eye is looking down, appears to be bright and easy to be excluded from being possible location for pupil location. However, when eye is looking down, iris and pupil are largely covered by eyelid and eye lashes. The existence of eye lashes increase noise for detecting and it is normally in X coordination.

All the tested images are from BioID dataset and it is a very challenging dataset to test. When testing for each table, it takes skin color and gender into consideration. All the entities in each table are a mix of dark and light skin color as well as male and female. However, more test entities are required for a more cogent result. In addition, all the tests are taken in the form of detecting pupil from images. However, this software is intended to be used in real life. Therefore, face-to-face testing towards patient should be included as well.

Other test case results could be found in Appendix D. It could be found that main program realises all the expected functions.

- Understandability

A detailed description of intended users is provided in Chapter 1. Software functionality could be found in Design and Implement stage. User interface, Use Case Diagram, and Interaction Char are all clearly provided.

- Documentation

  Overview of the system has been provided in Chapter 1 and detailed design is provided in Chapter 4. Clear, step-by-step instructions, anticipated view, and error messages and their solutions are included in Appendix E in the user guide.

- Installability

  Detailed requirement for the system to install on the target platform, and instructions for installation are provided in Appendix E in the user guide.

- Testability

  Unit tests are conducted for each component and integration tests are conducted as well for the system as a whole. All test cases brought out in design stage has been fulfilled thoroughly. This could be found in Appendix A and D.

- Portability

  This software could be built on and run on IOS system only as required in the project requirement.

- Supportability

  All the documents that are for support are all included in this dissertation report.

- Anlaysability

  Source codes are well commented as could be seen from the code submitted. Additionally, all the code is structured into modules and those are directedly related to the design as could be found in implementing stage described in Chapter 5.

7.3 Project Strength and Weakness

As for the strength in the final product, firstly, it considers a range of detailed conditions that could influence the performance of the algorithm and proposes a method to improve the

performance. This is achieved by combing two algorithms in order to get a more reliable result. Secondly, the detector prints the pupil location data on screen in real time. It is more straightforward for users when using the software. In addition, user could take more than one pictures at a time and each new image will replace the old one. This is helpful when the image is taken badly because of unstable hands or the eyes happen to be closed because patient blinks. Moreover, it does not restrict the number of eye directions for examination. This is helpful when the user only need to exam patient's pupil deviation in some particular direction. Last but not least, it takes patient's head movement and user's movement into consideration when designing and implementing the way data are collected and presented. It eliminates error that could rise from patient's head movement and user's movement by setting pupil location as location within eye image and scaling the data when calculating the deviation.

As for strength for the whole project, it first of all considers several situations that might affect performance of the product in design stage, and offers experiment on those situations. Outcomes are evaluated in an objective way by providing mathematical result data. Additionally, the test data are obtained from the very challenging BioID database. Hence, there is no particular bias included in the errors.

The weakness first of all is that the image data used in testing stage is not big enough. Additionally, it fails to test in real life as the software is designed to be used in real life. Furthermore, the algorithm being used is computational complex. Hence, the speed of processing is relatively slow.

7.4 Ethical Use of 3rd Party Evaluation

Approval for ethical use of 3rd party evaluation is not needed for this project for that it does not involve 3rd evaluation within the project. Evaluation on testing results are mainly based on the mathematical data which does not need 3rd party evaluation. Other evaluations are based on check list and could be fulfilled without 3rd party evaluation.

Chapter 8

Learning Points

Through this research project, there are several knowledge that I have learned and several skills that I have developed.

- Skill of Time Management for Individual Project
  Although such skill of time management should have been developed through assignments in previous study, this project is a long term project that lasts an academic year and it is still challenging to retain a good time management throughout such a long for individual project. Therefore, this experience has trained me once more in developing better time management skill and helped me become a more organized person.

- Skill of Self-Study
  In the beginning of this research project, I only had a little knowledge in developing IOS application and a little knowledge of C++ programming. Those knowledge is definitely not enough to fulfill the project task. Therefore, self-study is then important and I did put a lot of effort in self-study and researching for papers and websites for all the knowledge that I needed.

- Knowledge of Computer Vision
  This project is about pupil detection based on computer vision. During the project, I went through quite a few books and papers in related area and therefore gained abundant knowledge in such area.

- Skill of Developing IOS Application and Coding in Objective C
  I have never developed an IOS application before. Through the project, I created the first IOS application in my lifetime and it is completed in programming language of Objective C and C++. This is also the first time that I learnt and coded in Objective C.

Chapter 8

Professional Issues

8.1 Code of Practice

This part is completed under the instruction of [16].

Practices Common to all Disciplines

This project is pupil detection based on IOS system developing iPad backwards camera. Therefore, basic knowledge of IOS system, use of Xcode, objective-c language is required. These knowledge is obtained from iTunes U Course *Developing IOS 7 Apps for iPhone* provided by Stanford University and from Apple APIs. Knowledge of computer vision and image processing is obtained from library books about Computer Vision and other online resources. In addition, for the use of OpenCV Library, basic knowledge about OpenCV is from tutorials and APIs on OpenCV website.

This project is conducted adhering to the regulations. It considers user with non-discriminatory legislation in the area of gender, race, ethnic origin, disability or age. It uses appropriate methods and tools while developing. The device used to conduct this project is MacBook pro and iPad. The workload is managed efficiently. It also performances critical evaluation on the performance of the product.

Practices Specific to Research Functions

This research is aimed to develop a software that could be used in daily life. It contributes to make pupil detection more accessible in that is requires common device which is iPad. While providing relatively accurate result, it does not require expensive head-mounted device, which offers benefits to the society.

All the contributions from other people or organizations are properly cited and referenced. It does not involve any illegal IT support or performance. It also does not violate any private data

in that it uses only open source data that has retrieved approval from the donator that those data could be used in research of related area.

Practices Specific to Business Functions

User guide is provided to support users to use this software. All the result data are explained explicitly. The software being developed takes needs of performance, capacity, access, security, connectivity into consideration and is created to provide the most cost-effective solutions. In specification, objective of the project is clearly stated as well as performance, constraints, environment required.

The codes created in this project is well-documented and is strived to conduct well-structured considering the length of code, capacity it takes, and security issues. Platform of using this software is restrained to be IOS system only and device is limited to be iPad.

Testing is conducted covering as many paths possible as possible. Test environment is implemented separately where the results are predictable and it does not have any negative influence on the actual software. It also gives a detailed test log. The documentation guide is sufficient for user's use in that it describes the steps for using the software in detail, and any maintenance issue.

8.2 Code of Conduct

This part is completed under the instruction of [17].

Public Interest

The legitimate rights of Third Parties are well concerned as the project is being conducted in that all the testing data which involves personal data have been obtained approval from those donators. In addition, all other codes and open APIs provided by any third party are well cited and referenced. This project also provides unbiased access and seek to encourage the inclusion of all sectors in society.

Professional Competence and Integrity

This project is conducted only within my professional competence. Technological developments, procedures, and standards in the area of computer vision have been well aware

during the project. My professional knowledge and skills also have been well improved. Criticism of the work is respected and valued. This project tries to avoid hurting others, properties or reputation.

Duty to Relevant Authority

Relevant Authority's requirements are taken serious consideration and professional responsibilities have been taken in accordance with that. Any condition that might rise conflict against the Relevant Authority is avoided. All the contribution from third party has been legally permitted by Relevant Authority. No misrepresentation exists and it does not benefit from any inexperience from others.

Duty to the Profession

Conduct of this project holds up the professional reputation and no such action that may cause disrepute exists in conduct of this project. By developing, using and enforcing, the professional standards are hoped to be improved. Respect is highly shown to all members of BCS and with members of other professions with whom I work in a professional capacity.

Bibliography

[1] National Audit Office., *Reducing Brain Damage: Faster Access to Better Stroke Care*, 2005LondonNAO

[2] F. Rowe, D. Brand, C. A. Jackson, A. Price, L. Walker, S. Harrison, C. Eccleston, C. Scott, N. Akerman, C. Dodridge, C. Howard, T. Shipman, U. Sperring, S. MacDiarmid and C. Freeman, "Visual impairment following stroke: do stroke patients require vision assessment?," *Age Ageing*, vol. 38, no. 2, pp. 188-193, 2009.

[3] Cleveland et al., "Eye tracking method and apparatus.," U.S. Patent 5 231 674 A, Jul 27, 1993.

[4] Anon., "Field Proven Systems for Quantitative Analysis of Eye Movement, Pupillometry, and Visual Function.," *Applied Science Laboratory*, 1982.

[5] C Danchaivijitr, C Kennard., "Diplopia and Eye Movement Disorders.," *J Neurol Neurosurg Psychiatry*, vol.75, pp. 24-31, 2004.

[6] R. Kothari and J. Mitchell, "Detection of eye locations in unconstrained visual images.," *In Proceedings of the IEEE ICIP*, vol. 3, pp. 519–522, IEEE, 1996.

[7] F. Timm and E. Barth, "Accurate eye centre localisation by means of gradients.," *In VISAPP*, pp. 125–130, 2011.

[8] Linda G. Shapiro and George C. Stockman, "Computer Vision.," pp. 279-325, New Jersey, Prentice- Hall, ISBN 0-13-030796-3, 2001.

[9] T. Moravč iḱ , "An Approach to Iris and Pupil Detection in Eye Image.," *XII International PhD Workshop*, University of Zilina, Slovakia, pp. 239-242, 2010.

[10] J. Illingworth, J. Kittler, "The Adaptive Hough Transform.," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, pp. 690-698, 2009.

[11] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features.," *In Computer Vision and Pattern Recognition, CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1, pp. I-511. IEEE, 2001.

[12] T.Hume, "Simple, accurate eye center tracking in OpenCV.," 2012 [Online]. Available: http://thume.ca/projects/2012/11/04/simple-accurate-eye-center-tracking-in-opencv/. [Accessed 22nd March 2017]

[13] “Face Detection using Haar Cascades.,” OpenCV Tutorial [Online]. Available: http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html. [Accessed 28th January 2017]

[14] “The BioID Face Database.,” BioID [Online]. Available: https://www.bioid.com/About/BioID-Face-Database. [Accessed 4th February 2017]

[15] M. Jackson, S. Crouch and R. Baxter, “Software Evaluation: Criteria-based Assessment,” software sustainability institute, 2011. [Online]. Available: https://www.software.ac.uk/sites/default/files/SSI-SoftwareEvaluationCriteria.pdf. [Accessed 14th April 2017]

[16] “Code of Good Practice.,” The British Computer Society, 2011 [Online]. Available: http://www.bcs.org/upload/pdf/cop.pdf. [Accessed 21st April 2017]

[17] “BCS, THE CHARTERED INSTITUTE FOR IT TRUSTEE BOARD REGULATIONS - SCHEDULE 3 CODE OF CONDUCT FOR BCS MEMBERS.,” The British Computer Society, 2015. [Online]. Available: http://www.bcs.org/upload/pdf/conduct.pdf. [Accessed 21st April 2017]

Appendix A

Testing Results

a. Pupil Location When Looking Forward

Left Eye Forward				**Right Eye Forward**			
X	Y	Error in X	Error in Y	X	Y	Error in X	Error in Y
238	109	0	2	156	115	4	2
228	110	4	0	158	109	4	1
223	102	0	1	153	100	1	2
211	95	1	0	150	95	1	1
226	110	1	2	145	111	0	0
213	114	0	1	134	113	0	1
225	124	5	4	155	121	3	5
223	120	5	3	151	120	4	3
247	89	1	3	165	92	3	1
230	120	0	2	142	126	3	1
Average Error		1.7	1.8	Average Error		2.3	1.7

b. Pupil Location When Looking to Left

Left Eye to Left				**Right Eye to Left**			
X	Y	Error in X	Error in Y	X	Y	Error in X	Error in Y
234	130	2	2	189	130	2	0
219	126	1	2	173	129	3	4
185	142	5	1	137	147	2	3
228	90	6	3	174	88	5	1
235	96	2	1	176	100	3	3
233	96	3	0	175	96	4	2
230	87	2	1	176	89	2	0
243	95	3	3	181	91	1	1

Average Error	3	1.6	Average Error	2.8	1.8

c. Pupil Location When Looking to Right

Left Eye to Right				**Right Eye to Right**			
X	Y	Error in X	Error in Y	X	Y	Error in X	Error in Y
225	95	6	2	136	93	12	6
239	80	7	5	157	76	8	4
153	121	3	2	100	124	3	3
215	113	5	1	164	111	3	0
218	102	3	3	166	104	5	2
214	125	2	0	157	123	1	2
207	130	6	3	151	132	6	1
171	102	3	1	127	100	4	2
Average Error		4.4	2.1	Average Error		5.3	2.5

d. Pupil Location When Looking Up

Left Eye Up				**Right Eye Up**			
X	Y	Error in X	Error in Y	X	Y	Error in X	Error in Y
222	71	4	4	163	73	3	5
210	59	3	0	147	56	4	1
200	90	6	3	139	82	2	2
205	67	1	2	140	67	4	5
205	67	3	3	145	63	4	2
212	65	0	3	152	62	1	2
228	77	2	2	165	67	1	2
203	75	3	6	140	74	4	8
207	96	1	3	141	88	0	4
218	57	5	2	150	54	1	1
Average Error		2.8	4.6	Average Error		2.4	3.2

e. Pupil Location When Looking Down

Left Eye Down				Right Eye Down			
X	Y	Error in X	Error in Y	X	Y	Error in X	Error in Y
207	106	12	4	120	103	12	7
229	135	9	3	145	134	10	4
231	93	3	2	150	93	6	1
237	51	8	3	148	48	7	3
239	46	5	2	150	44	3	1
233	161	1	2	148	163	5	4
167	113	0	3	110	111	2	1
187	110	4	3	130	114	1	3
220	75	2	1	145	73	2	3
150	152	1	4	106	156	2	2
Average Error		4.5	2.7	Average Error		5	2.9

f. Pupil Location for Low Contrast between Iris and Sclera

Left Eye Down				Right Eye Down			
X	Y	Error in X	Error in Y	X	Y	Error in X	Error in Y
202	132	6	2	126	129	4	6
200	128	4	2	130	129	1	2
209	141	1	2	139	142	3	1
198	126	3	1	130	126	6	1
212	142	2	1	144	140	4	1
205	135	5	4	137	135	5	6
202	127	4	1	142	127	4	3
Average Error		3.57	1.86	Average Error		3.86	2.86

g. Pupil Location for Different Distances

Distance	Left Eye Down				Right Eye Down			
	X	Y	Scaled X	Scaled Y	X	Y	Scaled X	Scaled Y
60cm	36	17			29	12		
55cm	38	18	37	18	31	13	29	12

50cm	39	19	37	18	33	16	29	13
45cm	40	22	38	19	36	22	29	15
40cm	43	25	38	19	40	26	29	14

Appendix B

Codes

a. Code of Face Detection

```
- (void) processImage: (Mat&) image;{

    Mat grayscaleFrame;
    cvtColor(image, grayscaleFrame, CV_BGR2GRAY);
    // Convert frame image to gray scale for later processing
    equalizeHist(grayscaleFrame, grayscaleFrame);
    // Improve contract in the image

    std::vector<cv::Rect> faces;
    faceCascade.detectMultiScale(grayscaleFrame, faces, 1.1, 2, HaarOptions, cv::Size(60, 60));
    // Detect face with face cascade classifier

    for (int i = 0; i < faces.size(); i++){
        cv::Point pt1(faces[i].x + faces[i].width, faces[i].y + faces[i].height);
        cv::Point pt2(faces[i].x, faces[i].y);

        cv::Point pt3(faces[i].x + faces[i].width*0.5, faces[i].y + faces[i].height);
        cv::Point pt4(faces[i].x+faces[i].width*0.5, faces[i].y);

        cv::rectangle(image, pt1, pt2, cvScalar(0, 255, 0, 0), 1, 8 ,0);
        // Draw face area

        cv::Rect newObjectRight(pt2,pt3);
        // Get area of right half of detected face
        cv::Rect newObjectLeft(pt4,pt1);
        // Get area of left half of detected face

        [self detectEyeWithClassifier:rightEyeCascade withFrame:grayscaleFrame withOriginalImage:image check:newObjectRight id:@"right"];
        // Detect right eye on right half side area
        [self detectEyeWithClassifier:leftEyeCascade withFrame:grayscaleFrame withOriginalImage:image check:newObjectLeft id:@"left"];
        // Detect left eye on left half side area
}
```

b. Code of Eye Detection

```
1. - (void)detectEyeWithClassifier: (CascadeClassifier) classifier
   withFrame:(Mat&) grayscaleFrame withOriginalImage:(Mat&)image
   check:(cv::Rect)face id:(NSString*)name;{
2.
3.     std::vector<cv::Rect> object;
4.     classifier.detectMultiScale(grayscaleFrame, object, 1.1, 2, HaarOptions,
   cv::Size(60,60));
5.     // Detect new eye with cascade classifier
6.
7.     for (int i = 0; i < object.size(); i++){
8.         cv::Point pt1(object[i].x + object[i].width, object[i].y + object[i].height);
9.         cv::Point pt2(object[i].x, (object[i].y+30));
10.        // Crop new eye area to remove brow
11.        cv::Rect newEye(pt1,pt2);
12.        // Set up new eye area
13.
14.        if ( [self checkObjectEye:newEye onFace:face] ){
15.        // Check whether new eye area is on face
16.            // Yes, then
17.            cv::rectangle(image, pt1, pt2, cvScalar(0, 255, 0, 0), 1, 8 ,0);
18.            // Draw eye area
19.
20.            cv::Mat croppedNewEye = image(newEye);
21.            // Reset area of interest to be detected eye area
22.
23.            [self detectPupilwithHough:croppedNewEye onFace:newEye id:name];
24.            // Call method for detecting pupil
25.        }
26.    }
27. }
```

c. Code of Pupil Detection with Circle Hough Transform

```
- (cv::Point)detectPupilwithHough:(Mat&)image onFace:(cv::Rect)eye
id:(NSString*)name;{

    Mat cimg;
    if (image.type()==CV_8UC1) {            // Convert image to grayscale
        cvtColor(image, cimg, CV_GRAY2RGB);
    } else {
        cimg = image;
        cvtColor(image, image, CV_RGB2GRAY);
    }

    medianBlur(image, image, 5);          // Smooth the image with median filter

    std::vector<Vec3f> circles;
    // Detect iris circle with Circle Hough Transform
    cv::HoughCircles(  image
                     // InputArray (image being processed)
                     , circles
                     // OutputArray (array of circles being detected)
                     , CV_HOUGH_GRADIENT
                     // Method being used here
                     , 1
                     // Set inverse ratio of resolution
                     , 8
                     // Set minimum distance between detected centers
                     , 100
                     // Set upper threshold for the internal Canny edge detector
                     , 30
                     // Set threshold for center detection
                     , 5
                     // Set minimum radio to be detected
                     , 30
                     // Set maximum radio to be detected
                    );

    int circle_x = 0;
    int circle_y = 0;
    cv::Point circle_center;
    int circle_radius = 0;
    int index = 0;

    for ( size_t i = 0; i < circles.size(); i++ ){
        Vec3i c = circles[i];
        int x = c[0];
        int y = c[1];
```

```
45.         cv::Point center(x, y);
46.         int radius = c[2];
47.
48.         // Calculate average location of detected circles
49.         if(i==0){
50.             // If only one circle is detected, then
51.             circle_x = x;                                   // do nothing
52.             circle_y = y;
53.             circle_center = center;
54.             circle_radius = radius;
55.             index++;
56.         }else{
57.                 // If more than one circle is detected, then
58.                 index++;
59.                 circle_x = (circle_x + x)/ index;           // calculate average X
60.                 circle_y = (circle_y + y)/ index;           // calculate average Y
61.                 cv::Point tmp(circle_x, circle_y);          // get average point
62.                 circle_center = tmp;
63.                 circle_radius = (circle_radius+radius)/index;
64.                 // calculate average radius
65.         }
66.     }
67.
68.     circle( cimg, circle_center, circle_radius, Scalar(255,0,0), 1, CV_AA);
69.     // Draw detected circle
70.     circle( cimg, circle_center, 1, Scalar(0,255,0), 1, CV_AA);
71.     // Draw circle center
72.
73.
74.     cv::Point circleGradient = [self detectPupilwithGradient:image
    withRect:eye];
75.     // Process image with means of gradient method
76.     circle( cimg, _center_Gradient, 1, Scalar(0,0,255), 5, CV_AA);
77.
78.     return circle_center;
79. }
```

d. Code of Pupil Detection with Means of Gradient

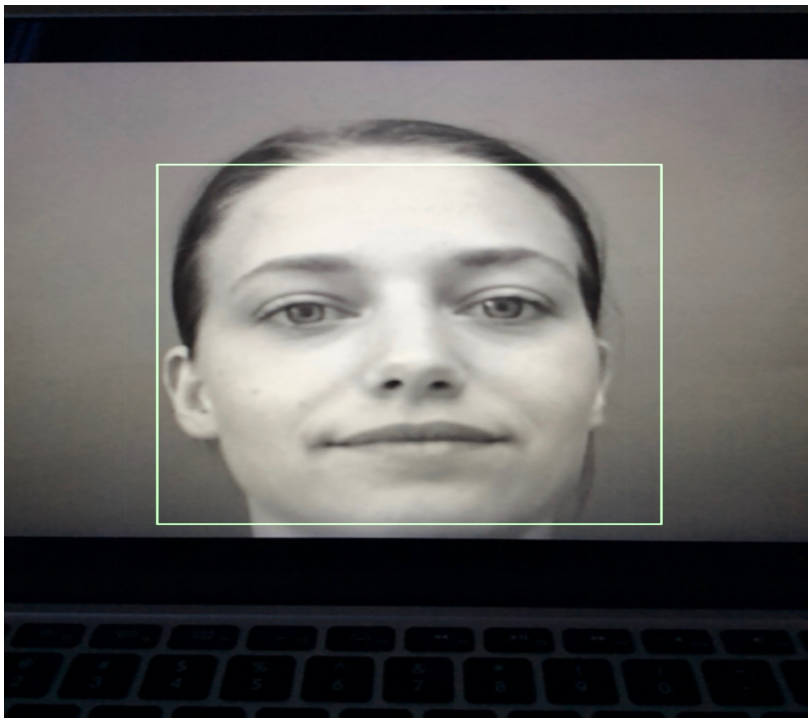
```
- (cv::Point) detectPupilwithGradient: (Mat&)eyeROI withRect: (cv::Rect)eye{
    cv::Mat gradientX = computeGradient(eyeROI);
    // Get gradient in X cordinate
    cv::Mat gradientY = computeGradient(eyeROI.t()).t();
    // Get gradient in Y cordinate
    cv::Mat magnitudes = matrixMagnitude(gradientX, gradientY);
    // Compute the magnitudes
    double gradientThreshold = computeThreshold(magnitudes,
kGradientThreshold);
    // Compute the threshold

    // Normalize the gradient with threshold
    for (int i = 0; i < eyeROI.rows; ++i) {
        double *Xr = gradientX.ptr<double>(i);
        double *Yr = gradientY.ptr<double>(i);
        const double *Mr = magnitudes.ptr<double>(i);
        for (int x = 0; x < eyeROI.cols; ++x) {
            double gX = Xr[x];
            double gY = Yr[x];
            double magni = Mr[x];

            if (magni <= gradientThreshold) {
            // Remove gradient when magnitude is lower than threshold
                Xr[x] = 0.0;
                Yr[x] = 0.0;
            } else {
                Xr[x] = gX/magni;
                Yr[x] = gY/magni;
            }

        }
    }

    cv::Mat weight;
    // Create a blurred image to get a smoother image
    GaussianBlur( eyeROI, weight, cv::Size( kWeightBlurSize,
kWeightBlurSize ), 0, 0 );

    // Invert the image
    for (int y = 0; y < weight.rows; ++y) {
        unsigned char *row = weight.ptr<unsigned char>(y);
        for (int x = 0; x < weight.cols; ++x) {
            row[x] = (255 - row[x]);
        }
    }
```

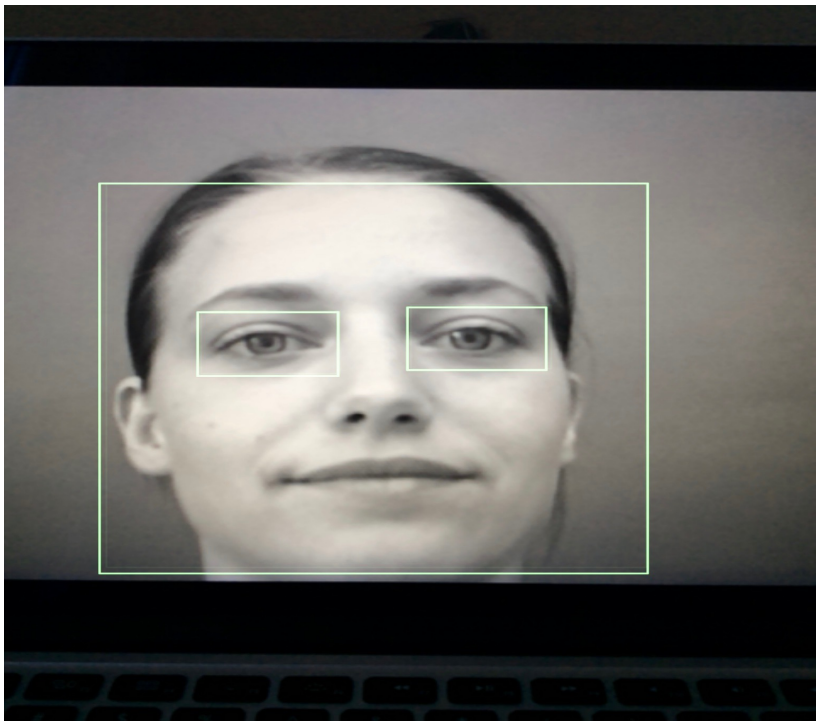
```
44.
45.    cv::Mat outSum = cv::Mat::zeros(eyeROI.rows,eyeROI.cols,CV_64F);
46.
47.    // Evaluates every possible center for each gradient location
48.    for (int j = 0; j < weight.rows; ++j) {
49.       // For every possible gradient location (non-zero)
50.       const double *Xr = gradientX.ptr<double>(j);
51.       const double *Yr = gradientY.ptr<double>(j);
52.       for (int i = 0; i < weight.cols; ++i) {
53.          double gX = Xr[i];
54.          double gY = Yr[i];
55.          if (gX == 0.0 && gY == 0.0) {
56.             continue;
57.          }
58.          testPossiblePoints(i, j, weight, gX, gY, outSum);
59.          // Test gradient vector and displacement vector
60.       }
61.    }
62.
63.    double numGradients = weight.rows*weight.cols;
64.    cv::Mat outarray;
65.    outSum.convertTo(outarray, CV_32F,1.0/numGradients);
66.    // Scale all the values down
67.
68.    // Find the maximum point among all the possible points
69.    // c* = arg max_c* { 1/N Σ_{i=1}^{N} ( d_i^T g_i )^2 }
70.    cv::Point maxP;
71.    double maxVal;
72.    cv::minMaxLoc(outarray, NULL, &maxVal, NULL, &maxP);
73.    // Finds the global maximum point and value in outarray
74.
75.    return unscalePoint(maxP,eye);
76. }
77.
78. void testPossiblePoints (int x, int y, const cv::Mat& weight, double gx, double
    gy, cv::Mat& output) {
79.    for (int cy = 0; cy < output.rows; ++cy) {
80.       double *Or = output.ptr<double>(cy);
81.       const unsigned char *Wr = weight.ptr<unsigned char>(cy);
82.       for (int cx = 0; cx < output.cols; ++cx) {
83.          if (x == cx && y == cy) {
84.             continue;
85.          }
86.          // Create a vector from the possible center to the gradient origin
87.          double dx = x - cx;
```

```
88.          double dy = y - cy;
89.          double magnitude = sqrt((dx * dx) + (dy * dy));
90.          // Normalised displacement vector di to unit length
91.          dx = dx / magnitude;                              // Normalize dx
92.          dy = dy / magnitude;                              // Normalize dy
93.
94.          double dotProduct = dx*gx + dy*gy;
95.          // Compute dot product of di and gi
96.          if(dotProduct<0){                          // Turn negative value into zero
97.             dotProduct = 0;                         // Avoid opposite direction
98.          }
99.
100.        // compute ∑_{i=1}^{N} w_c (d_i^T g_i)^2
101.         Or[cx] += pow(dotProduct,2) * Wr[cx];
101.      }
102.   }
103. }
```

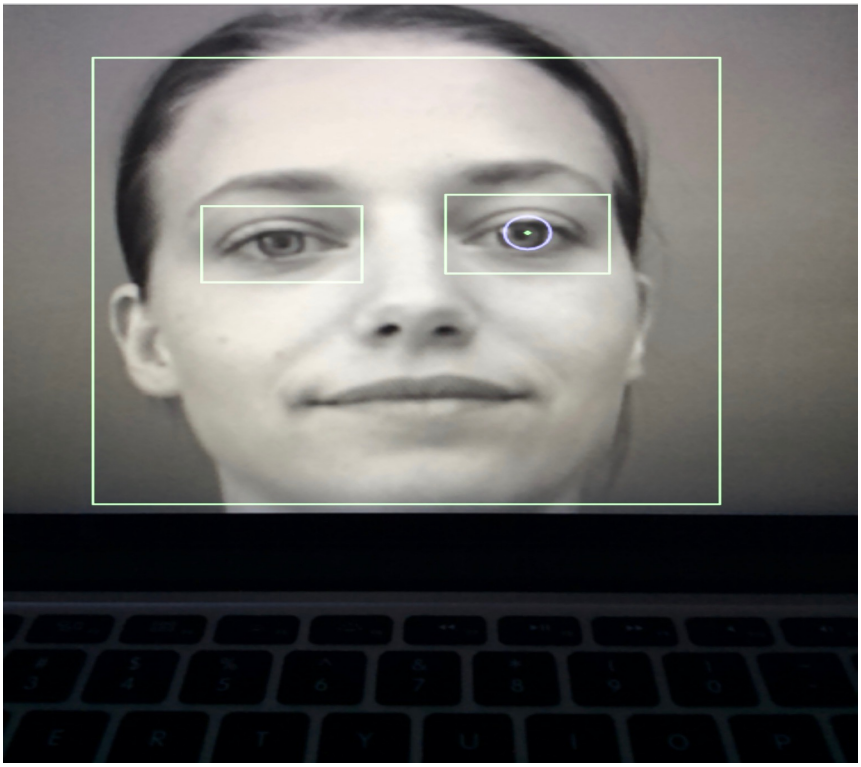
Appendix C

List of Figures

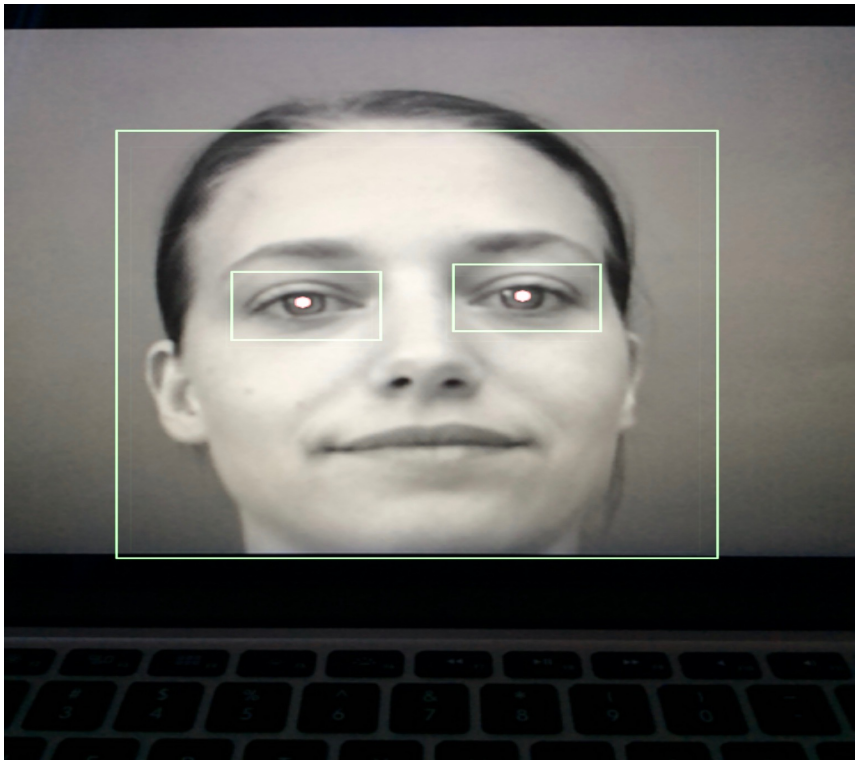




- Figure C.3 Pupil Detection with Circle Hough Transform



- Figure C.4 Pupil Detection with Means of Gradient



Appendix D

Test Case Result

Scenario	Test Step	Expected Result	Actual Result
Verify that tabbing snap button when pupil is detected, data is saved.	Apply the detection algorithm to the frame image and tab the snap button when pupil location is detected.	Data of pupil location is saved.	Data of pupil location is saved.
Verify that tabbing snap button when pupil is not detected, data will not be saved.	Apply the detection algorithm to the frame image and tab the snap button when pupil location is not detected.	Data of pupil location is not saved.	Data of pupil location is not saved.
Verify that tabbing snap button when pupil is detected, the data being saved is indeed the data being returned.	Apply the detection algorithm to the frame image and tab the snap button when pupil location is detected. Compare the saved data to the data printed on screen.	The two data are identical.	The two data are identical.
Verify that user can snap more than one time in all directions	Select each direction and snap 3 to 5 times	New result overwrites old ones every time user tabs snap button	New result overwrites old ones every time user tabs snap button
Verify that data being retrieved is successfully sent to presenting page	Apply the detection algorithm to the frame image and tab the snap button. Then go to presenting page	Result is shown on presenting page	Result is shown on presenting page

Verify the data being retrieved is saved and presented successfully according to the eye direction	Apply the detection algorithm to the frame image and tab the snap button. Repeat the step in different eye directions. Then go to presenting page	Result is shown on presenting page according to the eye direction	Result is shown on presenting page according to the eye direction

Appendix E

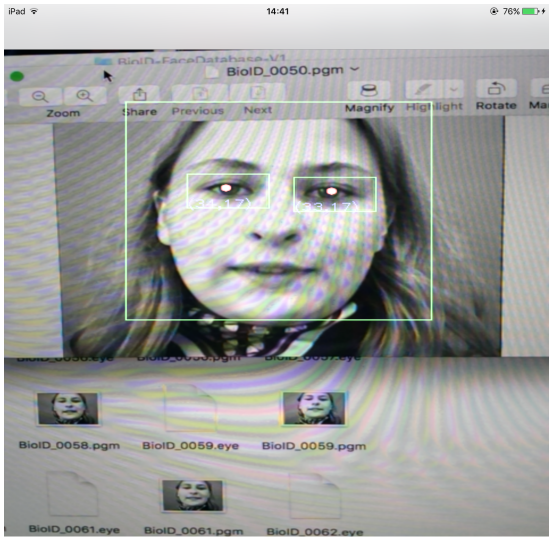
User Guide

- Hardware Requirement
  An iPad is needed.

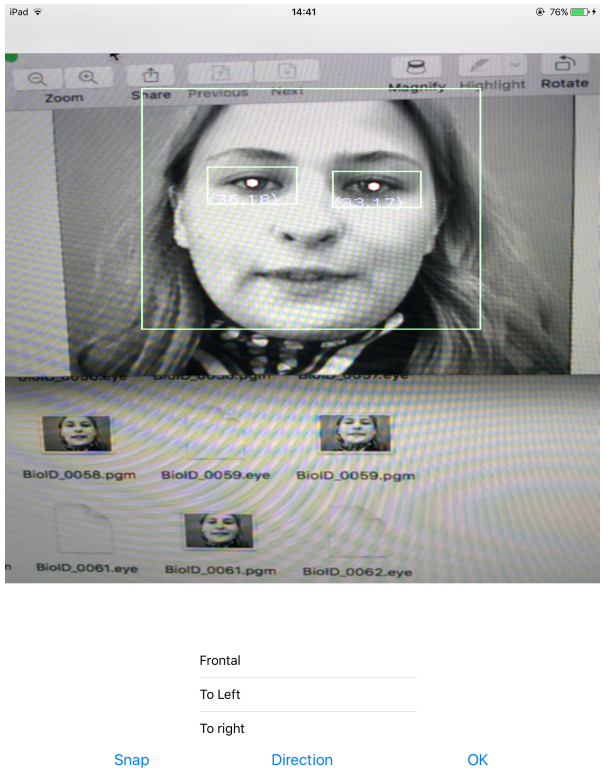
- Guide of Use
  1. The first screen you will see is as below. This is just a plain page before entering the actual detector. You can tab the button start to start detecting.

Start

  2. As soon as you enter the detector page, the software will then start to process all the frames captured by the camera. You might see a similar page like the one below.

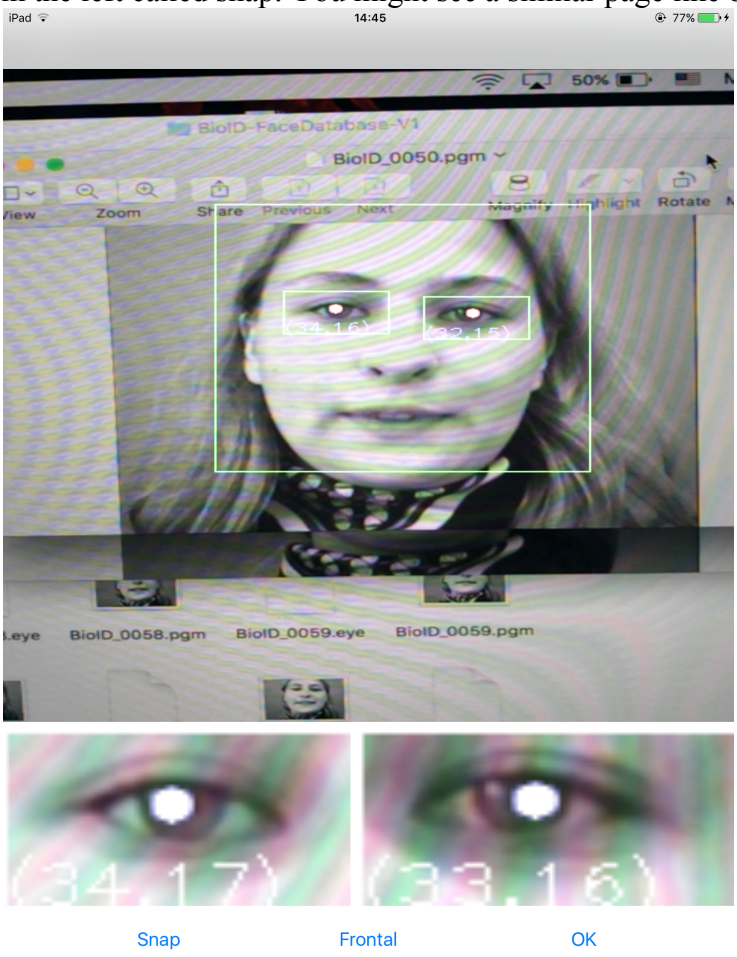


3. Now the detector has calculated the detected pupil location and printed it on screen. You then need to choose which direction you are examining by tabbing the middle button as shown below.



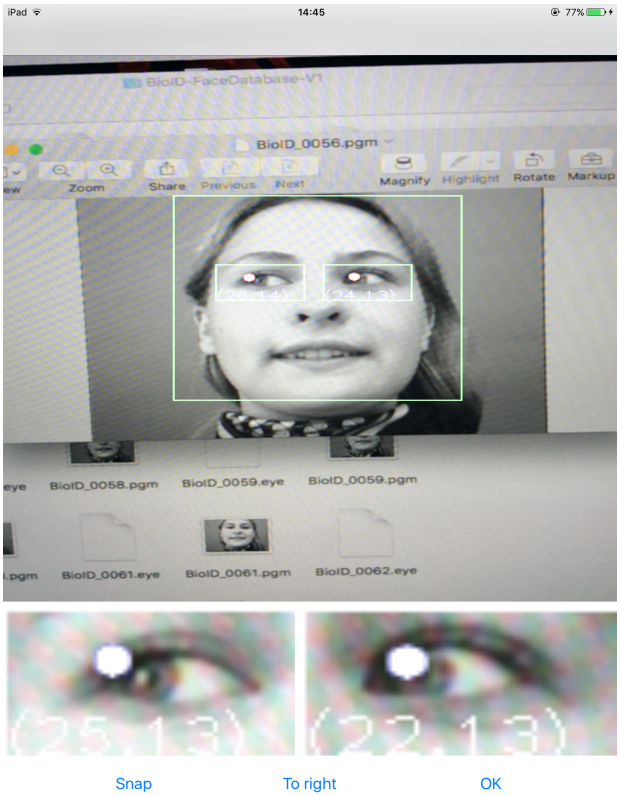
A selection menu will pop up and you need to choose the direction you are examining by selecting one in the menu. In this example above, we are examining eye looking forward. Therefore, Frontal should be selected.

4. After selection the direction, you now need to take a picture to allow the software to save the data it has calculated and printed onscreen. You can do that by tab the button in the left called snap. You might see a similar page like below.

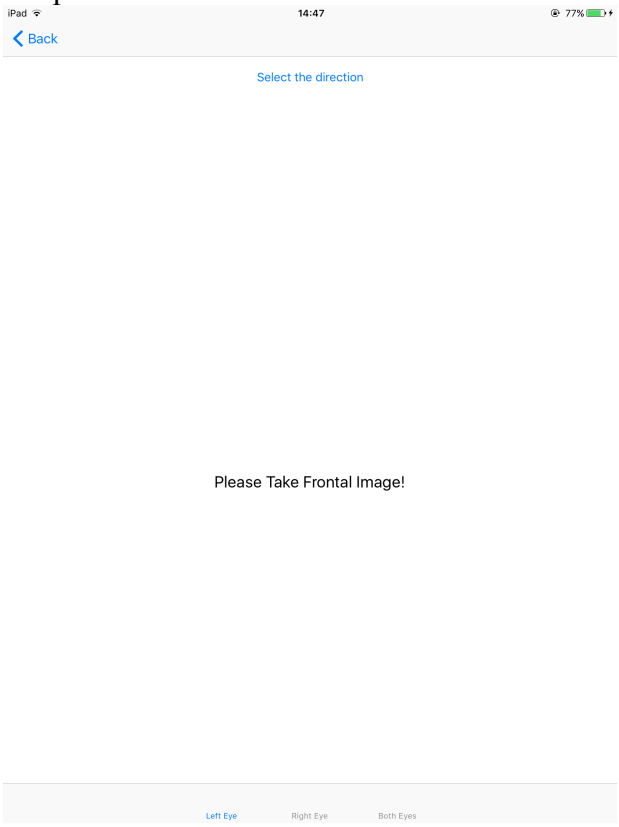


You can see two cropped images containing only eye area is shown. You can check if you are satisfied with the images now. If not, you can take another one simply by tabbing the snap button again. Two new cropped images will appear in the same place.

5. Now you might need to exam another situation, such as eye looking right. Then you need to, same as step 3, choose the direction you are interested in. And then, same as step 4, you need to tab snap button again to save the image and data. A screenshot is shown as below.



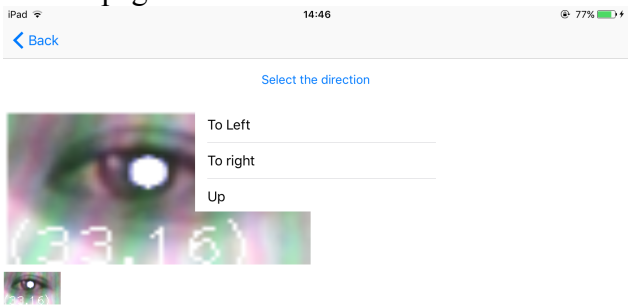
6. You can snap as many times as you want until you think it is good enough. Also, there are five eye directions for you to choose from. You do not need to exam every direction in the selection. You can go to the present page by tabbing OK button in the right. However, if you want to go to the present page to see all the results, you then will at least take image of the frontal eye. If you did not take frontal eye image, this page will show up as shown below.



This is the error message for you asking you to take frontal eye image. If you see this message, this means you did not take image of eye looking forward. You then need to go back to the previous page through back button in the upper left corner and go through step 3 and 4 again. The error message in right eye result page is the same as left eye result page as shown above. The error message for both eye result page is shown as below.

7. Following step 5, now you have taken the image of frontal eye and eye looking to right. You then might want to see the deviation between the two direction. The you need to tab the right button name OK. You will see similar page as shown below.

8. It shows on the page the pupil location in frontal eye image. You then need to choose deviation in which direction you want to see. You should tab the middle button. You will similar page as below.



X 33
Y 16

9. Now we choose to right in the menu. You will see similar page as below.

X 33
Y 16

X 22
Y 13

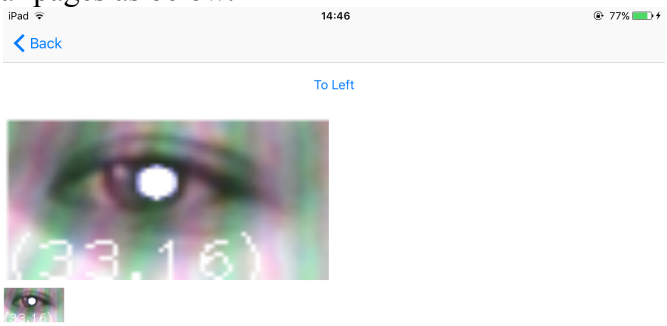
X Deviation 10
Y Deviation 2

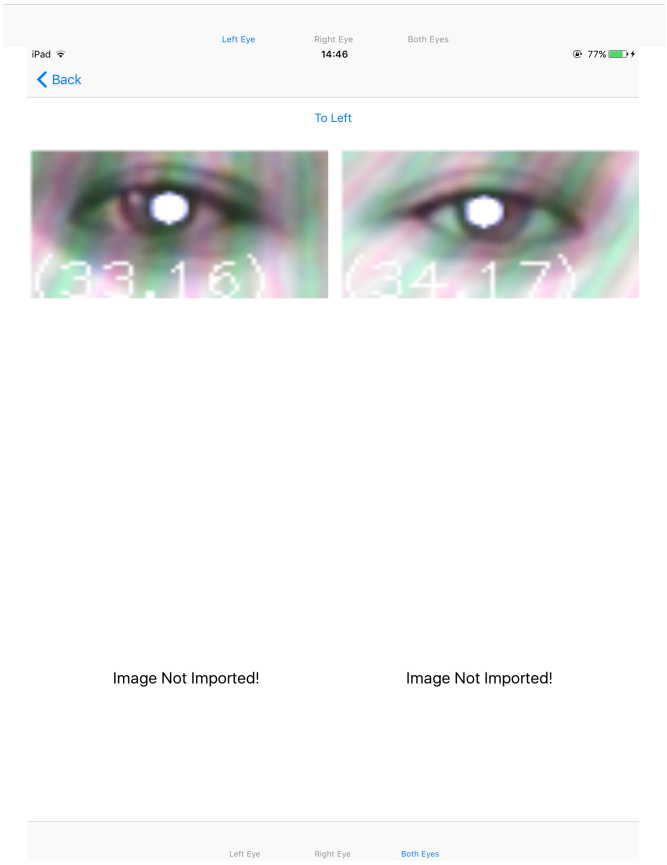
The pupil location in image of eye looking to right is shown. The deviation between the two pupil locations are calculated and shown on screen as well.

If you want to see result of right eye, you can choose the middle tab in the bottom, you will page like below.

Also, results for both eyes are like the screenshot as below.

10. If you try to see the result for the direction in which you did not import image, you might see similar pages as below.





If you see such error message, please go back to detector page and take images in those directions as describe I step 3 and 4. You do not need to retake any images

that you have taken before. Then you can go to this present page to see results in those directions.

Appendix F

Previous Design Document

SUMMARY OF PROPOSAL

This project focuses on developing a component that helps better detection of pupil deviation in eye examination. It will be achieved by implementing several algorithms that process the imported images of intended test takers given instructions. A lot have been progressed by now. More reference papers on dealing with object detection in images have been reviewed. Deeper understanding of the algorithms has been achieved. A detailed design has been proposed and will be discussed as below.

SYSTEM DESIGN

Anticipated Components Overview

This system consists mainly three components. They are two User Interfaces and the main program in the background. First User interface receives data imported from the user. In this case, data are all the pictures which are either being taken by user using iPad backwards camera or pictures imported from the library. After user press the “Start” button, all the images will be sent to the background program. The main background program is mainly responsible for processing the image. It should give a cropped image of the eyes from the original image and also indicate the location of the pupil with a marker. Furthermore, the main program calculates the deviation of the pupil between the initial picture and one picture of the four directions. The second User Interface is responsible for displaying all the processed images and the calculated data.

A data flow diagram will be shown as below.

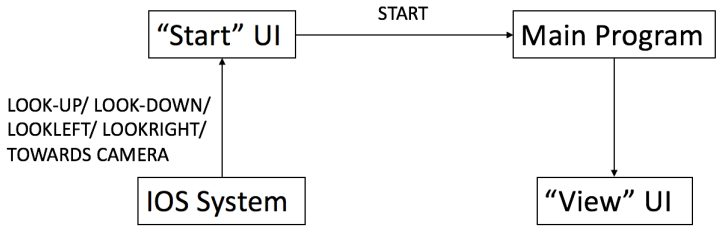
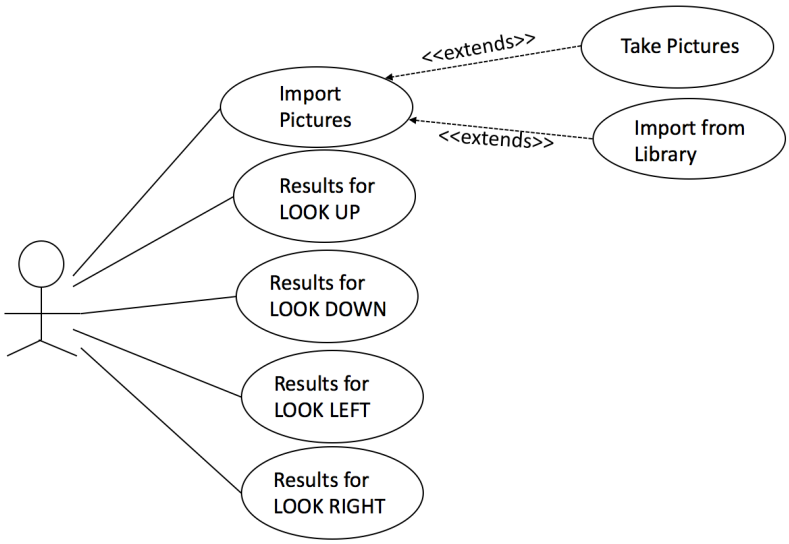
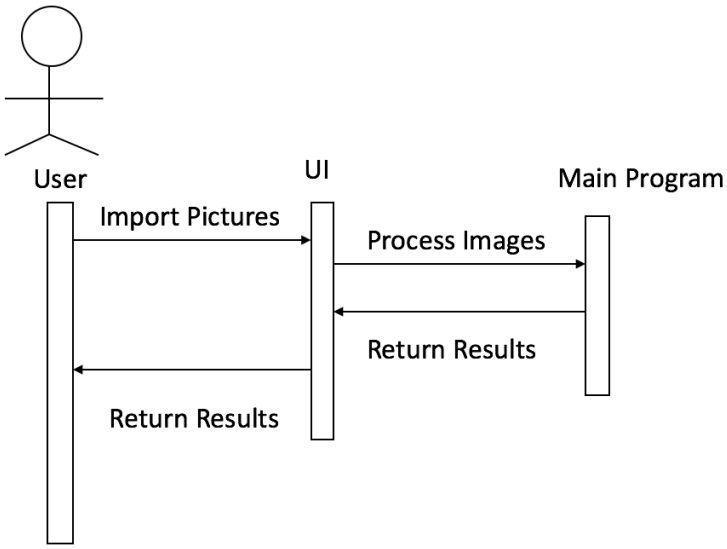


Fig. 4 Data Flow within the System

Use Case Diagram



Interaction Chart



CONCLUSION

In conclusion, this design report discussed about all the design that has been proposed. It includes user interface design, main program design, interface design between modules, and evaluation design. User interface design focuses on making the processed results clear and straightforward for users. In main program design, it focuses on designing the three image processing methods. In testing part, it discussed about the testing criteria and the way of displaying test data.