Controlling Android Device with Eye Tracking

Supervisor

Professor Md. Haider Ali

Chairperson

Department of Computer Science and Engineering

Sakib Hasan 12301003

Md. Tahsin Alam Tamim 12301022

S. M. Tarequl Hasan Robin 12301013

Sadia Tasnim 14101265

Shah Tanvir Mahmud 12321054

Declaration

This thesis is submitted to the School of Computer Science and Engineering, BRAC University, Dhaka. In partial fulfillment of the requirements for Bachelor's degree in Computer Science and Engineering.

Signatures of Author

Sakib Hasan (12301003)

Md. Tahsin Alam Tamim (12301022)

S. M. Tarequl Hasan Robin (12301013)

Sadia Tasnim (14101265)

Shah Tanvir Mahmud (12321054)

Signature of Supervisor

Professor Md. Haider Ali

Abstract

For people with mobility disabilities, communicating via smartphone is often frustrating, and require constant assistance. Among the diseases, only Parkinsons disease alone affects up to 10 million people worldwide[1]. Especially, people having hand tremor, have major issues using mobile phones. To solve this, we want to implement a mobile application that will enable users to operate the mobile device touch-free. Most of the current eye tracking solutions use expensive hardware, but our proposed system will not require any additional hardware, it will use mobile device's front camera. To achieve this, we will use combination of few detection methods including: face, eye, pupil, corner detection and calibration of the data.

Acknowledgment

After a rigorous period of three semesters, today is the day; writing this note of appreciation is the finishing touch on our thesis. It has been a period of vigorous learning for us, not only in the scientific arena, but also on a personal level. On this very day, we would like to reflect on the people who have supported and helped us so much throughout this period.

First of all, we would like to thank the Almighty for making a way for us through the rough world. Without His help we would never have existed in this universe.

Secondly, we would like to express our sincere gratitude to our adviser Prof. Dr. Md. Haider Ali for believing in us with the project, supporting us endlessly, for his patience, motivation, and in-depth knowledge which have guided us towards our destination. He has definitely provided us with the correct path that we needed to choose for the right direction and successfully complete our thesis. His doors were always opened for us whenever we needed him.

We would also like to thank our parents who were always there for us and helping us for achieving this tremendous goal. Their constant and tireless support, both morally and intellectually; have been one of the key factors of our achievement. Without all their help, we would not be on the verge of graduation.

Contents

List of Figures

1 Introduction

1.1 Overview

With the boom of technological advancement people are using mobile devices ever than before. Almost everybody has a smart device with them in this modern age. The mobile phones were introduced not more than 45 years and now it is being used in every aspect of human life. However almost all of these devices still need human touch or press command to operate itself. This creates a huge problem for the people with disabilities that can range from war veterans to Parkinson's disease victims that makes the person unable to do work by themselves. All the smartphones from the market will be no use for them. Because all of the phones assume the user can interact with the device with either press or touch command. To solve this, we have implemented a system that enables the device to understand what the user's intent is and help him establish a bridge with smart devices with just tracking their eyes.

The localisation of centres has significant importance in many computer vision applications such as human-computer interaction, face recognition, face matching, user attention or gaze estimation. That being said, according to Gao, the technology is still in the "embryonic stages"[2] of development and is not ready to be used in daily life. But we intend to explore the options and come up with a usable solution that can improve a disable person's life. We will assume for this project that the user owns an Android smartphone and will only interact with eyes. No special hardware will be needed. Many of the previous commercial products required head mounted or invasive gear like infrared camera. The need of extra hardware is a big problem for the consumer since they do not want to spend extra money for additional hardware.

However, to actually implement our idea we must acquire in depth knowledge about oculometry which is basically biometric measurement of the condition and movements of the eye. Its prime focus is to measure the position and orientation of the eye of a subject as the person looks at a screen of a computer. As we are proposing a model that is convenient for people with mobility disabilities, our main motto is to remove the necessity of human touch for operating phones and all other complexities. For that, we have done some research. Our researches have led us towards some effective methods for eye-tracking or gaze tracking. Traditional gaze tracking systems rely on either contact and invasive hardware, or expensive and non-standard hardware. To address this problem research has been done into systems that use only simple

hardware to create gaze tracking systems. Inspired by this thought, we were able to find a way to use device's front camera as the device for eye-tracking.

1.2 Problem Statement and Proposed Model

Helping one another is a natural instinct of human being. Seeing disabled person struggling to communicate with other people is heartbreaking. The problem that we tried to solve with our thesis is that, disabled persons are not able to use smartphone by touch or keypress. To help them we proposed a smartphone application that can track their eye movement and use it to perform basic communication tasks. The conducted research had few specific goals in mind:

- Design and implement a functional eye tracking application that can control the smartphone.
- The proposed model must not use any expensive hardware to perform the eye tracking. It should just use the front camera of the smartphone.
- The application should be very user-friendly and easy-to-use.
- The eye tracking process should be accurate and very fast.

1.3 Thesis Outline

The rest of this report is structured as stated below:

- **Chapter 2** covers the pre-requisite knowledge required to understand how eye functions, how eye tracking works and methods of eye tracking.
- **Chapter 3** focuses on overall system design along with design decisions.
- **Chapter 4** covers the implementation of the system and describes all the steps of eye tracking in details.
- **Chapter 5** explains testing process and discuss about the know limitations of the device.
- **Chapter 6** summerize the whole thesis and discuss the effectiveness of the system.

2 Background Study

2.1 Introduction

To have a clear idea, we have read some previous papers, projects about eye, gaze tracking and its implementations on an Android device. There are few papers published for eye/gaze tracking, for example: [3], [4], [5]. These papers discuss about various techniques to detect eye position and movement. A brief overview for these techniques will be discussed in later sections. As for implementation on an actual device, there is no useable application available on Android Play Store for using Android device with eye. So, we see a lot of possibilities in this side of technology. But before the actual implementation, we are going to cover the basics of human eye structure, eye movement, algorithms for eye tracking available to use.

2.2 Structure of Human Eye

Human eye is one of the most sophisticated part of human body. Before designing and implementing an eye tracking system that can accurately find where the person looking at on their cellphone, we must understand the structure and behaviour of the eye.

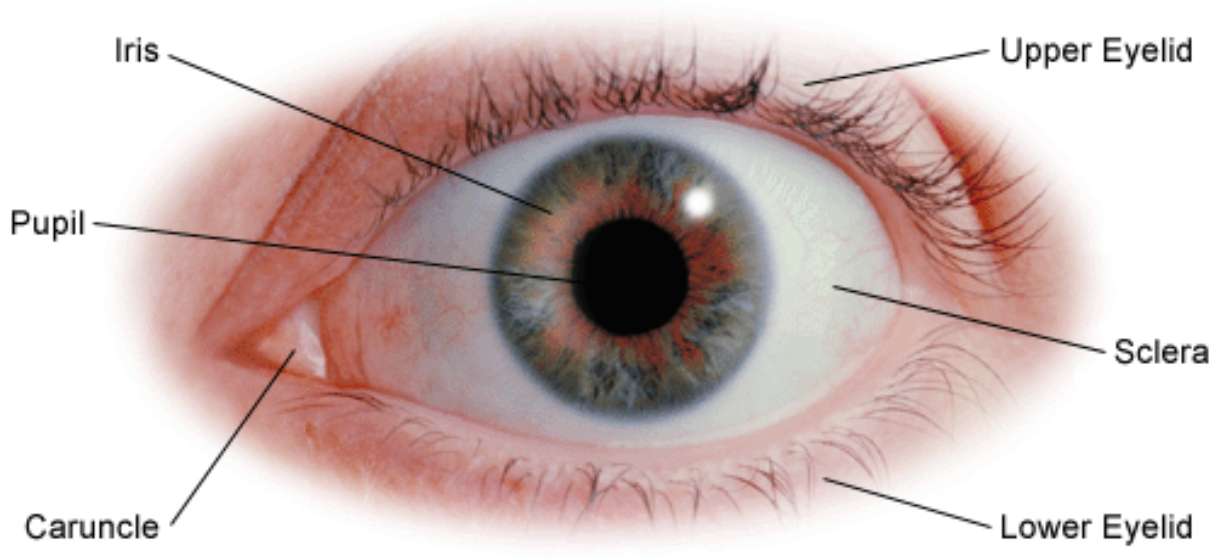

Figure 1: Anatomy of an eye

As seen on the figure[1], human eye have a perfect circle combining Iris and Pupil. It is surrounded by a white background. The Pupil is the centre of Iris and it represents where the person is looking at or the gaze of the person. The eye can rotate in a very fast motion in

almost all directions. The eye also have eyelids, that covers up the whole eye time to time. This process takes place around 10 times per minute and it is called the blinking process. The purpose of blinking have different useful biological factors but blinking can also be used as a sign for useful indication. For this research, we have used both pupil and blink detection.

2.3 Eye Movement Methodology

It is very important to understand the eye movement a person do both consciously and sub-consciously. Because of the eye moves so fast, its a major issue for the eye tracking system to reduce potential calculation errors and lags. Our system have a limit of 30 frames per second image recording. Since we process each and every frames from input video, its a big challenge to maintain accuracy when detecting eye movements.

Normally, a major issue for eye movement tracking is a reference point while the eye moves. It is harder to track gaze, If we consider the Caruncle as the reference point, because the calculation delay will be high and the target person must be standing directly perpendicular to the camera in order to produce an accurate reading. For a user, keeping his head straight throughout the eye tracking process will be very uncomfortable and annoying.

When a person moves his eye to a specific direction, the eye make sudden jump in that direction. Even though the eye take some short pauses along the path, this process takes place instantaneously and beyond the limit of image processing. So, while taking input coordinates of the pupil, we will not consider gradual increase of the values. We will take the major displacement and run the algorithms to take decisions where the person is looking at.

2.4 Methods of Eye Tracking

The most accurate methods for finding the eye accurately requires very expensive equipment. Moreover, many of the methods might not be comfortable for a disabled person, like using a contact lens or mounting camera on head. Since we are using the mobile device's front camera, we can not expect the input image will be high quality. To solve both implementation and low-resolution problem, we will look into Haar Cascade method to detect gaze.

Object Detection using Haar feature based cascade classifiers is an effective object detection method. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. Once its is trained, it can be used to detect objects in other

images. Haar-like features can be computed at any scale or location in constant time using the integral image representation for images. Already trained open source cascade classifiers for eye and pupil detection is available on the internet. Detailed explanation of this method will be covered in this section.

2.4.1 Image Detection using Haar Classifiers

Object Detection from image using Haar feature based cascade classifiers is a very effective object detection method. It is proposed by Paul Viola and Michael Jones on their paper [6]. We choose this method because it is simple to implement yet feature-rich. Also, this method doesn't require a lot of post-processing. Most of the image processing and training of data is done before the implementation, so its very fast and responsive on runtime.

First, lets understand what is a classifier in our context. Classifier is the outcome of training with hundreds of positive and negative images. Suppose we want to detect a car from a given image. For the positive images, we take thousands of images of different cars of different models and makers. Then we manually tag where the car is in each picture. As for the negative image, we take almost same amount of positive images or more. And those image must not contain any cars. Then we can train the classifier using a specified algorithm. After the classifier is trained, it can be applied to an input image. The classifier will give a Boolean result whether or not there is any car on it. Since the input image will be possibly much larger than the trained image, the search window will be moved over the input image to detect possible multiple occurrence. The classifier is designed so that it can be easily "resized" in order to be able to find the objects of interest at different sizes, which is more efficient than resizing the image itself. So, to find an object of an unknown size in the image, the scan procedure should be done several times at different scales.

To use this method for face detection and eye tracking, initially, in order to train our classifier, we need a lot of positive and negative images of both face and eye. Then, we need to tag the features on each image. Each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle.

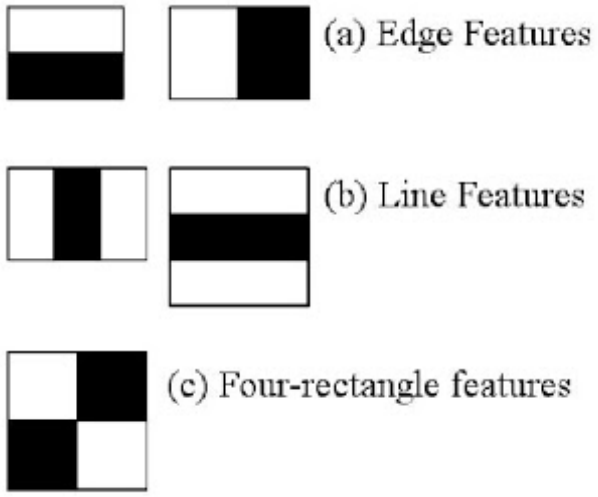

Figure 2: Haar Features

Now all possible sizes and locations of each image is used to calculate plenty of features. That will take huge amount of computational power, even if its for a small window with large amount of features. For each feature calculation, we need to find sum of pixels under white and black rectangles. To solve this, P. Viola and M. Jones introduced the integral images. It simplifies calculation of sum of pixels, how large may be the number of pixels, to an operation involving just four pixels. This improves the speed of calculation.

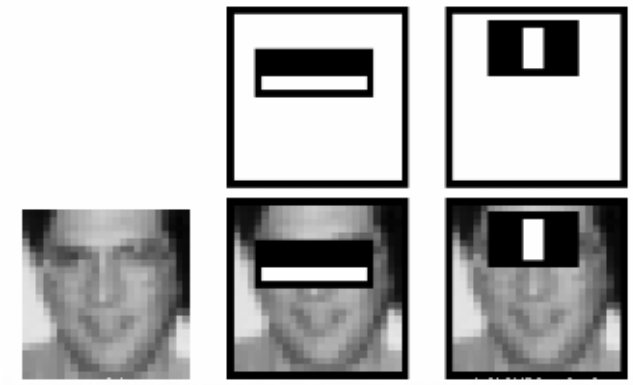

Figure 3: Eye detection using haar feature

But among all these features we calculated, most of them are irrelevant. For example, consider the image of figure: 3. Top row shows two good features. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. But the same windows applying on cheeks or any other place is irrelevant. So we use Adaboost to select the best features from large number of features.

For this, we apply each and every feature on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. But, there might be errors or misclassifications. We select the features with minimum error rate, which means they are the features that best classifies the face and non-face images. Final classifier is a weighted sum of these weak classifiers. It is called weak because it alone can't classify the image, but together with others can form a strong classifier. The paper says[6] even 200 features provide detection with 95% accuracy.

Now, if we input a sample image to the system, it takes each 24x24 window, and apply 6000 features to it, then check If its a face or not. It seems very inefficient and time consuming. But in practice, since most of the part of the image will non-face region, the algorithm first checks if the window is a face region. If it is not, discard it in a single phase. And the algorithm will not process it again. Instead focus on region where there can be a face. This way, we can find more time to check a possible face region.

So, here comes the concept of Cascade of Classifiers. Instead of applying all the 6000 features on a window, group the features into different stages of classifiers and apply one-by-one. If a window fails the first stage, discard it. We don't consider remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region.

2.5 Algorithm

A Gaze tracking system is a combination of several different techniques, of which Eye tracking is only a part. There are a great many methods of combining these techniques to create algorithms. Some examples of Algorithms are given below in order to show the differences, similarities and general structure of gaze tracking algorithms.

2.5.1 RTAC Algorithm

RTAC stands for Real-time gaze Tracking Algorithm with head movement Compensation. As the name suggest, the algorithm will track down the pupil even though there are head movements. According to the paper [7], the core of this algorithm is "Pupil Center Corneal Reflection (PCCR)". To calculate the direction of gaze, first the system takes the pupil-glint vector and compute a gaze mapping function. The glint is the reflection of the infrared light source in the eye. The pupil-glint vector is then the 2D vector between the glint and the pupil. The

algorithm then uses a mapping function to map this vector to the 3D gaze direction. The mapping function is given below:

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

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

Here, θ_h is the angle between the gaze direction and the horizontal direction and likewise θ_v is the angle between the gaze direction and the vertical direction. The coefficients - a and b, are estimated using the sets of pairs of pupil-glint vectors.

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

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

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

Then the next step is fixing the user's head during calibration in order to get the initial parameters required. After this calibration the user can move their head freely and the parameters are compared to do the calculations.

According to the authors of [7], it was found that this algorithm had an accuracy rate of approximately one degree, and that there was little difference between different subjects. There was also little difference between different positions of the head, but the error did increase when the head was moved near the boundaries of the cameras vision.

2.6 Summary

At the end of this chapter we can clearly state that we now have the necessary knowledge and algorithms to design and implement a effective eye tracking system. So the next chapter will cover the design part of this project.

3 Design

After covering all the necessary knowledge of algorithm and eye tracking methods, we will now look into how we designed our proposed model. Designing the system was a very crucial part of this project, because there are lot of things to consider.

3.1 Design Decisions

Design decisions depends on various factors. The considerations we made are listed below.

- **No extra hardware.** Since, the application will be used by disabled person, we can not use any extra hardware. Extra hardware might be a burden for the disabled person.
- **Enable basic smartphone options via eye tracking.** Smartphones nowadays have enormous amount of functions and most of them require more complex interactions than normal touch. We will not cover all possible interaction via eye tracking. User will only be able to perform the basic functionalities of smartphone.
- **Simple yet effective.** The mobile application must be simple enough to be able to easily interact with it.

3.2 Smartphone

We do not need any extra hardware other than an Android Smartphone, but we do need some minimum hardware requirements to be able to run our application smoothly. To choose our prototype smartphone, we considered the following criteria.

- A phone with good front camera. Since the main input image will be taken using this, it's vital to get a good quality picture.
- High performance smartphone with fast processing chip. Without the processing power we can not get a good quality reading from the input image.
- A phone with large display, because it will give more space for the eye to travel to an element on screen to tap on it.

Considering all this criteria, we choose the OnePlus X smartphone from the market since it met all the points above and its a budget phone, which is perfect for a prototype. Below are some of the specification of the phone:

- Front Camera: 8 MP, f/2.4
- Display size: 5.0 inches
- Resolution: 1080 x 1920 pixels
- CPU: Quad-core 2.3 GHz Krait 400
- GPU: Adreno 330
- RAM: 3 GB

Figure 4: Front Facing Camera of OnePlus X

3.3 Software and Libraries

Lets focus on what software and libraries should we choose to design this application. Since we are going to make the application exclusive to Android smartphones, Android Studio by Google is the first name that came to our mind. Even though there are some other 3rd party tools that offer to build mobile application easily like Ionic Framework, React Native, Cordova, Onsen UI, Intel XDK, Sencha Touch, Kendo UI, Framework 7, JQuery Mobile, Mobile Angular UI, Monaca etc. But we need very raw and core power of mobile phones, so we stick to the poplar and well supported Android Studio for mobile application programming.

Figure 5: Android Studio Logo

The main library of this project is OpenCV. OpenCV stands for Open Source Computer Vision. OpenCV is free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform. Adopted all around the world, OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 9 million. Usage ranges from interactive art, to mines inspection, stitching maps on the web or through advanced robotics.

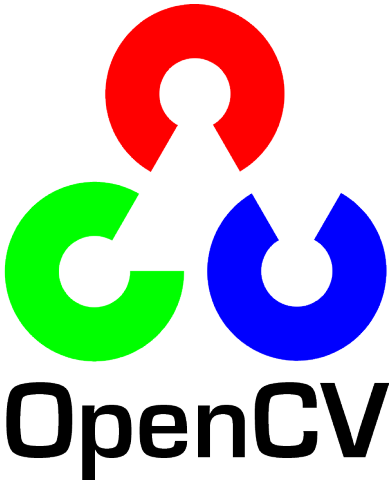

Figure 6: OpenCV Logo

OpenCV have full support for all the functions we have discussed on previous chapters like Haar Classifiers, Cascade Training etc.

3.4 Programming Language

For the mobile application, we are only focusing on the Android Operating System. So the natural choice is Java because it is the native programming language of Android Devices. The main library, OpenCV supports Java language, so we choose Java as our main programming language.

3.5 Algorithm

To get a brief idea about our project, lets see a flow chart that explain the steps of the Eye Tracking and controlling the Android device.

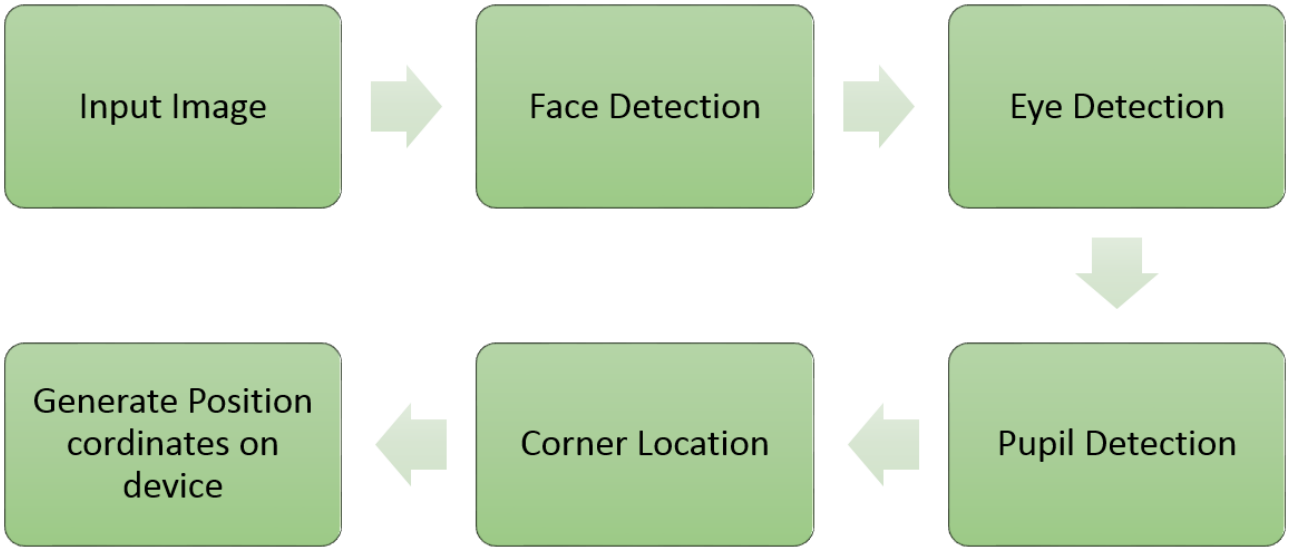

Figure 7: General Algorithm

The first step is to get a single frame from the camera. Once the frame is retrieved, our code will first check if there is any face on the frame, otherwise it will be ignored. Face detection is done before anything else because it will ignore most of the false positive images for eye or pupil tracking. But on other side, a requirement of this is that the face detection needs to be a quick process in order to not add extra delay to the system.

The next step is the eye detection. This step takes the area the face was found in and searches that area for the eyes. This step finds each eye separately. Both eyes are used in order to increase the accuracy of the system. But we can turn off one eye to achieve less processing delay. Once the eyes are obtained, the pupil and corners are needed.

The exact method of finding the pupil within the eye differs between algorithms but there are some consistencies. A particular technique is to detect circles in the image but this requires cleaning of the image first. Since no special lighting was used, some form of brightening of the image must occur in order to accentuate the differences between the pupil and the white of the eyes. It is also necessary to reduce the noise in the image. A threshold technique is useful to reduce the noise in the image. Edge detection is also usually a good method of accentuating the border around the iris. Once you have included enough noise reduction then it is possible to search the image for circles and find the circle of the Iris. The center of the pupil can be approximated as the center of this discovered circle.

The inner corner of the eye was chosen as the second feature to check. The reason a second

feature is required is to create a vector to track the changes in the eye when it moves within the eye socket. As such, this feature needs to be fixed relative to the movement in the eye. The nose can be used too, but the inner corner was decided on in order to have separate points within the reduced region for each eye. Once the coordinates of both the eye and the corner are obtained for both eyes, then the loop steps are complete.

Moreover, we can see from figure 7, our proposed model have no calibration stage. This will enable us to build a fast and real time use interaction.

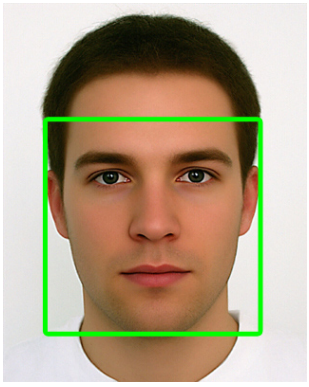

Figure 8: Face Detection

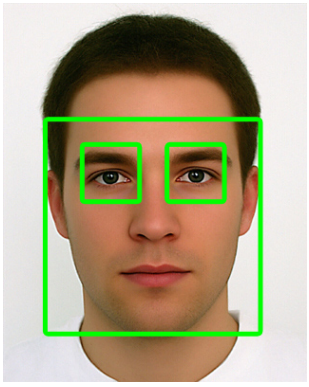

Figure 9: Eye Detection

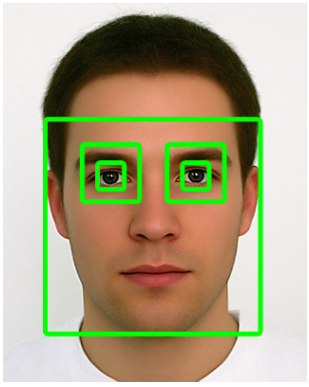

Figure 10: Pupil Detection

4 Implementation

4.1 Introduction

On this chapter of the report, we are going to highlight some of the most important algorithm, codes, and brief overview of each of the elements we implemented on our Android application. We will walk you through the transformation from design to a working system.

4.2 Application Structure

Firstly, we created a new Android Studio project and designed a dummy structure for our application. This dummy structure will help us to add functionality to the buttons and camera view.

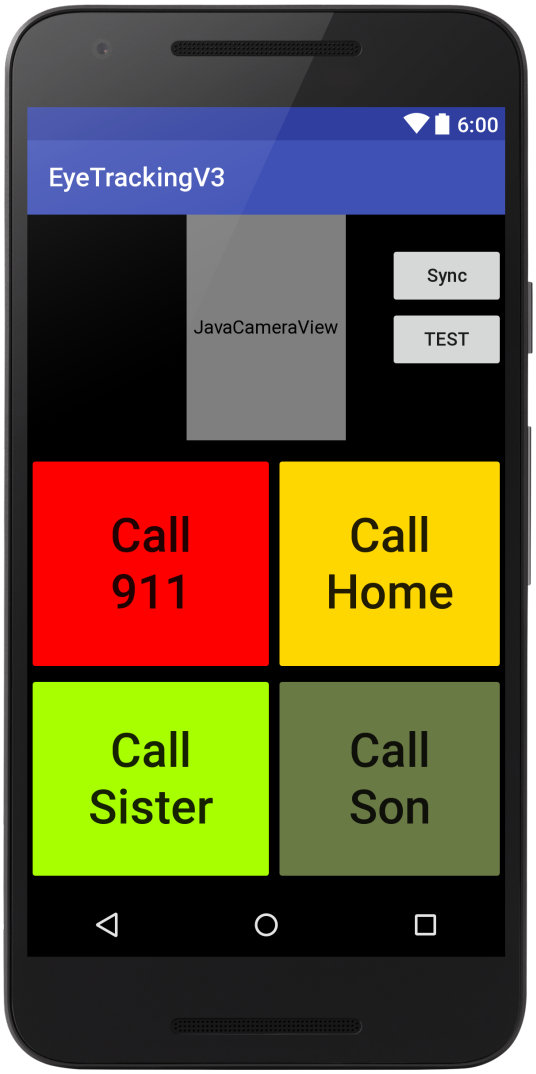

Figure 11: Pupil Detection

4.3 Camera Input

The very fast step to eye tracking is taking input image from the camera. Since the display needs to in front of the person, we will be using the front camera of the device. OpenCV have a built-in camera module for Java called JavaCameraView. This enables us get the user's input image without any native coding at all. The below code is used to insert a JavaCameraView into our Android Main Activity.

```
// activity_main.xml
<org.opencv.android.JavaCameraView
   android:layout_width="120dp"
   android:layout_height="170dp"
   android:id="@+id/fd_activity_surface_view"
   android:layout_alignParentTop="true"
   android:layout_centerHorizontal="true"
/>

// MainActivity.java
public void onCameraViewStarted(int width, int height) {
    mGray = new Mat();
     mRgba = new Mat();
     mRgbaF = new Mat(height, width, CvType.CV_8UC4);
     mRgbaT = new Mat(width, width, CvType.CV_8UC4);
     mGrayF = new Mat(height, width, CvType.CV_8UC4);
     mGrayT = new Mat(width, width, CvType.CV_8UC4);
 }
```

Using these two blocks of code, we can now access the input frame on `onCameraFrame()` method. We can manipulate and take any reading from this method.

There was a slightly difficult problem to solve on this part of the implementation. OpenCV always gives the input image on landscape mode, so we can not use the camera in portrait mode. Since, portrait mode is the natural position for all the smartphone users, we had to solve this issue by taking the input image and rotating the image by 90 degrees using Matrix Transpose and flip.

```
// Taking the RGBA and Grey of input frames
  mRgba = inputFrame.rgba();
```

```
mGray = inputFrame.gray();

// Rotate mRgba 90 degrees
Core.transpose(mRgba, mRgbaT);
Imgproc.resize(mRgbaT, mRgbaF, mRgba.size(), 0, 0, 0);
Core.flip(mRgbaF, mRgba, -1);

Core.transpose(mGray, mGrayT);
Imgproc.resize(mGrayT, mGrayF, mGray.size(), 0, 0, 0);
Core.flip(mGrayF, mGray, -1);
// Image rotated.
```

Now we are ready to process the input frames to track use's eye.

4.4 Face Detection

To use eye tracking using OpenCV we need to perform few steps as a pre-requisite. Firstly, we need the trained classifier for face detection. This training requires a lot of facial images and need large amount of time. For the purpose of this prototype, we are going to use a pre-trained front face cascade classifier from internet. To use this file, we need to first import it to OpenCV library, since its doesnt come with the library.

```
File cascadeDir = getDir("cascade", Context.MODE_PRIVATE);
mCascadeFile = new File(cascadeDir, "lbpcascade_frontalface.xml");
mJavaDetector = new CascadeClassifier(mCascadeFile.getAbsolutePath());
if (mJavaDetector.empty()) {
   Log.e(TAG, "Failed to load cascade classifier");
   mJavaDetector = null;
}else{
Log.i(TAG, "Loaded cascade classifier from " +
    mCascadeFile.getAbsolutePath());
}
```

After we have successfully loaded cascade classifier for Front Face detection, we can now use it to detect any face from input frames. So, on the `onCameraFrame()` method, we are going to check if there is any possible match using the following code.

```
// Initialize an array to store detected faces on input frame
MatOfRect faces = new MatOfRect();
if (mDetectorType == JAVA_DETECTOR) {
   if (mJavaDetector != null)
      mJavaDetector.detectMultiScale(mGray, faces, 1.1, 2, 2, new
          Size(mAbsoluteFaceSize, mAbsoluteFaceSize), new Size());
}else{
   Log.e(TAG, "Detection method is not selected!");
}
```

After running this code, we will get the position of face on the `faces` array. We draw a green square around the face to let the user know we have successfully detected his face. To draw the square, we use this code:

```
// Draw Green rectangle over face
Imgproc.rectangle(mRgba, facesArray[i].tl(), facesArray[i].br(),
    FACE_RECT_COLOR, 3);
```

This concludes the face detection part, now we move into eye detection phase of our project.

4.5 Eye Detection

Now we have at least one face detected on our input image, we can try to find the eye area of the face. Detecting the face before searching for eyes gave us one big advantage. That is, we have a limited search area for eye detecting. So we will have less errors and false positives.

We take the eye area from the detected face by using this equation:

```
Rect eyearea = new Rect(r.x + r.width / 8, (int)(r.y + (r.height / 4.5)),
    r.width - 2 * r.width / 8, (int)(r.height / 3.0));
```

Then we split the area into two halves.

```
Rect eyearea_right = new Rect(r.x + r.width / 16, (int)(r.y + (r.height /
    4.5)), (r.width - 2 * r.width / 16) / 2, (int)(r.height / 3.0));
Rect eyearea_left = new Rect(r.x + r.width / 16 + (r.width - 2 * r.width /
    16) / 2, (int)(r.y + (r.height / 4.5)), (r.width - 2 * r.width / 16) /
    2, (int)(r.height / 3.0));
```

And finally, we put another green square box around eye area and each zone.

```
        Imgproc.rectangle(mRgba, eyearea_left.tl(), eyearea_left.br(), new
            Scalar(255, 0, 0, 255), 2);
```

4.6 Pupil Detection

Pupil detection is not as simple as previous steps. This requires custom training images from every input frame of user. This is because training individual pupil detection will take long time and will be very inefficient. So, what we are going to do is, we will take 5 input frames from sequentially, and then run an algorithm to detect circle object from each of the images and finally constructing the detection based on some decisions.

```
        if (learn_frames < 5) {
           // Learn from frames
           teplateR = get_template(mJavaDetectorEye, eyearea_right, 24);
           teplateL = get_template(mJavaDetectorEye, eyearea_left, 24);
           learn_frames++;
        }else{
           // Learning finished, use the new templates for template matching
           match_eye(eyearea_right, teplateR, 5);
           match_eye(eyearea_left, teplateL, 5);
        }
```

As we can see here, after taking 5 input frames, it will call `match_eye` method, which will process the pupils position.

```
private void match_eye(Rect area, Mat mTemplate, int type) {
       Point matchLoc;
       Mat mROI = mGray.submat(area);
       int result_cols = mROI.cols() - mTemplate.cols() + 1;
       int result_rows = mROI.rows() - mTemplate.rows() + 1;
       // Check for bad template size
       if (mTemplate.cols() == 0 || mTemplate.rows() == 0) {
           return;
       }
       Mat mResult = new Mat(result_cols, result_rows, CvType.CV_8U);
       switch (type) {
```

```
            case TM_SQDIFF:
                Imgproc.matchTemplate(mROI, mTemplate, mResult, Imgproc.TM_SQDIFF);
                break;
            case TM_SQDIFF_NORMED:
                Imgproc.matchTemplate(mROI, mTemplate, mResult,
                        Imgproc.TM_SQDIFF_NORMED);
                break;
            case TM_CCOEFF:
                Imgproc.matchTemplate(mROI, mTemplate, mResult, Imgproc.TM_CCOEFF);
                break;
            case TM_CCOEFF_NORMED:
                Imgproc.matchTemplate(mROI, mTemplate, mResult,
                        Imgproc.TM_CCOEFF_NORMED);
                break;
            case TM_CCORR:
                Imgproc.matchTemplate(mROI, mTemplate, mResult, Imgproc.TM_CCORR);
                break;
            case TM_CCORR_NORMED:
                Imgproc.matchTemplate(mROI, mTemplate, mResult,
                        Imgproc.TM_CCORR_NORMED);
                break;
        }
        Core.MinMaxLocResult mmres = Core.minMaxLoc(mResult);
        if (type == TM_SQDIFF || type == TM_SQDIFF_NORMED) {
            matchLoc = mmres.minLoc;
        } else {
            matchLoc = mmres.maxLoc;
        }
        matchLoc_tx = new Point(matchLoc.x + area.x, matchLoc.y + area.y);
        matchLoc_ty = new Point(matchLoc.x + mTemplate.cols() + area.x, matchLoc.y
            + mTemplate.rows() + area.y);

        Imgproc.rectangle(mRgba, matchLoc_tx, matchLoc_ty, new Scalar(255, 255, 0,
            255));
        Rect rec = new Rect(matchLoc_tx, matchLoc_ty); // pupil's co-ordinates
    }
```

4.7 Post Processing

The post processing is the crucial part of eye tracking process. This is where we take the decision, where is the user looking into the screen. Any error in the decision will make the application uncomfortable to them. Now that we have the users co-ordinates of face, eye area and pupil, we can do the calculation of gaze. From the figure 12, we can get a brief overview of how the eye gaze is being calculated in a simple way, rather than any complex coding. The pupil is being monitored on every positive frames. We maintain an array of pupils location for 2 seconds, since we get around 20FPS input images, so that is 20 frames per second. So, we need to store about 40 to 50 frames data and get the pupils location on each frame. We check how x or y co ordinate values are changing. Then calculating all 50 frames data, we fill up our array and make a decision based on that. There might be some delay and errors, but it worked well in the end.

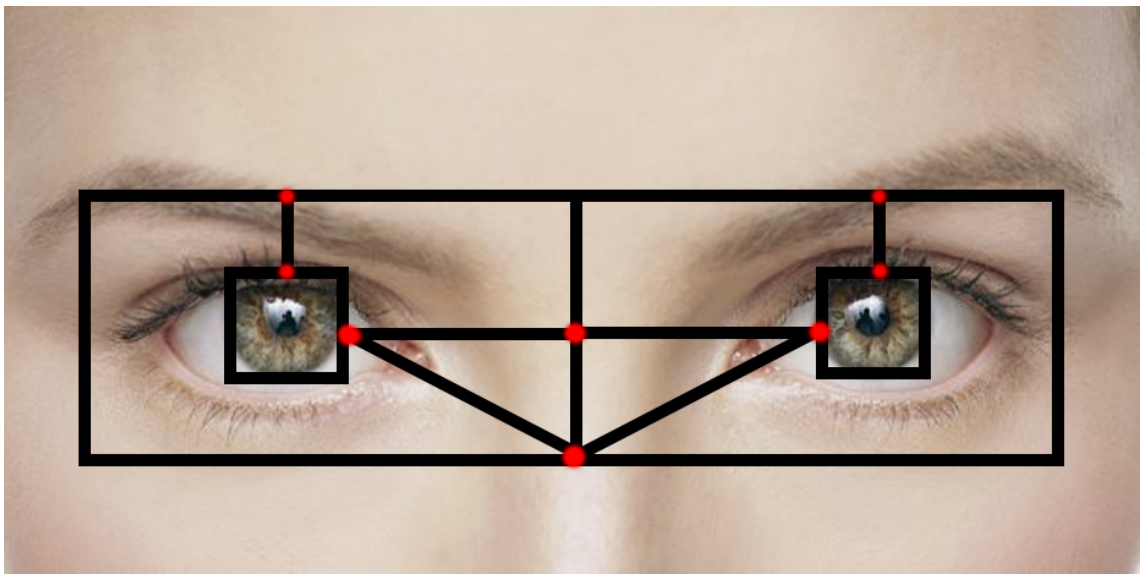

Figure 12: Post Processing

4.8 Translate gaze into mouse movements

To translate the pupils movement into an action on users smartphone, we need to acquire some special permission on the phone itself. Android have a built-in service to manage or help application control the device called `AccessibilityService`. We first get all the necessary permission on the phone by writing the following code in AndroidManifest.xml file:

```
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
```

This piece of code allows us to run a window overlay over the smartphones window. The purpose of this virtual overlay is to emulate a mouse movement over this. The mouse will be controlled from the pupils location algorithm described in previous section. After processing the frames, our code will generate a movement direction with respect to the gaze, and add 20 pixels to the mouse in the respective direction. We use the following code block to generate a mouse movement over the application.

```
if(null != MainActivity.mUiHandler) {
    Message msgToActivity = new Message();
    msgToActivity.what = 0;
    if(true ==mIsServiceRunning) {
        Log.d(TAG, "mouse move koro akhon => "+msg.obj);

        if(msg.obj.equals("left")){
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    cursorLayout.x -= 40;
                    windowManager.updateViewLayout(cursorView, cursorLayout);
                }
            });
        }else if(msg.obj.equals("right")){
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    cursorLayout.x += 40;
                    windowManager.updateViewLayout(cursorView, cursorLayout);
                }
            });
        }else if(msg.obj.equals("up")){
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    cursorLayout.y -= 40;
```

```
                windowManager.updateViewLayout(cursorView, cursorLayout);
            }
        });
    }else if(msg.obj.equals("down")){
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                cursorLayout.y += 40;
                windowManager.updateViewLayout(cursorView, cursorLayout);
            }
        });
    }else if(msg.obj.equals("click")){
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                AccessibilityNodeInfo nodeInfo = getRootInActiveWindow();
                if (nodeInfo == null) return;
                AccessibilityNodeInfo nearestNodeToMouse =
                    findSmallestNodeAtPoint(nodeInfo, cursorLayout.x,
                    cursorLayout.y + 50);
                if (nearestNodeToMouse != null) {
                    logNodeHierachy(nearestNodeToMouse, 0);
                    nearestNodeToMouse.performAction(AccessibilityNodeInfo.ACTION_CLICK);
                }
                nodeInfo.recycle();
            }
        });
    }
  }
}
break;
```

5 Results

5.1 Testing

After implementing the design, we are now in the testing phase. This is a major step because it will take repetitive process to finish. We have tested the application on the below criteria:

- Same person, same lighting environment.
- Different person, same lighting environment.
- Same person, different lighting environment.
- Different person, different lighting environment.

5.2 Result

Most of the cases, our system passed the test. That means, we could move the mouse cursor to the correct direction. There were some occasional miscalculation due to inconsistency of the lighting or foreign object on the frame that obstruct the face.

5.3 Known Limitations

Right now we have few limitations in the system, first one is, we can not let the user use all of the smartphones functions. We believe this is a big issue for our system. Because we want to let our user feel the same as a normal person would feel. Secondly, we are focusing on one eye for the time being to save processing lag. Even though this method will increase the frames per second, but to get better accuracy, we must take both eyes gaze into account.

Using the Haar detection as the face detection means that any differences from the main cascade in the pixel group intensities causes the system to be unable to find the face. So any user with dark skin will be unable to use the system.

Within the capabilities of the system, the error is still rather large. However the system can still determine the region of the user's focus. This still leaves a great many applications available for the system. Another benefit of gaze tracking systems, and in particular the design of this system, is that the modular nature of the design allows techniques to be replaced with better techniques, which are available, with relative ease. This means the system can be upgraded quickly and effectively.

6 Conclusion

6.1 Effectiveness of the System

We believe this project have enormous opportunities on real life application. There are many disabled people around us who can not communicate like normal people. Even though researches are going on different things, but we feel they are deprived of new technology and research. With this project, we tried to help the disabled people even if it is a small prototype.

6.2 Future Work

The proposed system is challenging and accuracy of the implementation is very crucial. To make the application as user-friendly as possible, it must be very accurate and fast. Our future work plan would be the following:

- Implement the proposed system on Apple smartphones.
- Perform various tests to measure accuracy.
- Test the application with disabled person.
- Write proper documentation for future reference.
- Publish the application for public uses.

7 Reference

[1] Parkinson's Disease Foundation. *Statistics on Parkinson's*. [ONLINE] Available at: `http://www.pdf.org/en/parkinson_statistics` [Accessed 16 August 2016]

[2] Gao, D., Yin, G., Cheng, W., and Feng, X. Non-invasive eye tracking technology based on corneal reflex *Procedia Engineering 29*, 0 (2012), 3608 3612. 2012 International Workshop on Information and Electronics Engineering.

[3] Y. m. Cheung and Q. Peng, "Eye Gaze Tracking With a Web Camera in a Desktop Environment," in *IEEE Transactions on Human-Machine Systems*, vol. 45, no. 4, pp. 419-430, Aug. 2015.

[4] Timm, F. & Barth, E. (2011), Accurate Eye Centre Localisation by Means of Gradients., in *Leonid Mestetskiy & Jos Braz, 'VISAPP'*, SciTePress, pp. 125-130.

[5] Dongheng Li, David Winfield, and Derrick J. Parkhurst. 2005. Starburst: A hybrid algorithm for video-based eye tracking combining feature-based and model-based approaches. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Workshops - Volume 03* (CVPR '05), Vol. 3.

[6] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference*, 2001, pp. I-511-I-518 vol.1.

[7] Huang, Y. , Wang, Z. , Ping, A. (2009). 'Non-contact Gaze Tracking with Head Movement Adaptation based on Single Camera'. *World Academy of Science, Engineering and Technology, International Science Index 35, International Journal of Computer, Electrical, Automation, Control and Information Engineering, 3(11), 2568 - 2571.*