Gaze Tracking on Android

Brian Maguire
B.A.I. Engineering
Final Year Project April 2014
Supervisor: Fergal Shevlin

School of Computer Science and Statistics

O'Reilly Institute, Trinity College, Dublin 2, Ireland
5.3 Security concern ................................................................. 32
5.4 Pay per Gaze implementation .......................................... 33
Conclusions .............................................................................. 33
1 Introduction

This project looked at a possible method for predicting the gaze of a smartphone user on the device's screen. The aim was to have reliable screen coordinates for where the user had been looking at any time while using their phone. The device's front-facing camera would be used to capture images, which would be processed using the image processing library OpenCV.

This method could be put into use in a number of different applications. The data that it produces would be of great value to developers of new smartphone application, and advertisers designing ads for the small screen.

Having accurate data for where the user is looking can tell you what parts of an app the user’s attention is being drawn to. What visuals in a banner ad attract the most looks, or what menu design allows the users the quickest glance before finding what they were looking for.

As an added tool to advertisers it could be used as a new systems for payment. Many banner ads on apps take up a large portion of the screen and may never be touched by a user, other than by mistake. The advertiser may be spending money on a useless advert. With data on where the user looks, an ad could be charged on a pay per gaze system, in that they only pay if someone has actually looked at their ad. This means an advertiser can be confident that they’re only spending money on advertising on those that actually have an interest.

As an alternative use, the method could be used as an input to applications. It could be used in situations where users can’t use the touch screen, while wearing gloves, or when cooking or painting. Users with physical disabilities could use gaze input if using touch is difficult or impossible for them.
2 Related Work

Collecting data on what a person is looking at is not a new problem and has been solved by many. Most methods involve the use of special equipment, such as eye mounted infrared cameras. These devices normally come with a great deal of precision, and a large price.

2.1 Light vs Dark pupil tracking

There are two main approaches when it comes to using infrared light and cameras to track a pupil. The light, and dark method.

2.1.1 Light Pupil tracking
This method involves shining an infrared light directly into the eye, where it is reflected back by the retina. This reflection is the same phenomenon that causes red eye in flash photography and can be picked up by an infrared camera as a bright spot in the middle of the eye. The method gives a good contrast between the iris and the pupil, and is generally unaffected by shadow.

The technique is mainly used in dark circumstances, and works best with blue eyed people. It can be affected by the size of the pupils changing due to any number of reasons including emotional stimulus to the participant.

2.1.2 Dark Pupil tracking
Dark pupil tracking still uses an infrared light source, but rather than shine it directly into the retina, the light is shined at the face and iris, making the pupils appear very dark in comparison. This dark spot is then tracked.

The technique is more susceptible to shadow and noise, and is best used in bright environments and can be used without a controlled infrared light source. It also works best with a dark coloured iris as the IR light is reflected better off the surface of dark iris creating better contrast with the pupil.
2.2 SMI Eye Tracking Glasses

The SMI eye tracking glasses are designed to collect gaze data for researchers or product developers.

The glasses have a camera pointed at each eye and a single camera pointing outward. The device makes use of a light pupil method by emitting an infrared source that is reflected off the cornea and then picked up by the eye cameras. The glasses can calculate the gaze of the wearer to an accuracy of 0.5 degrees. The glasses also make use of a customized Samsung Galaxy S4 to control the glasses.

2.3 Tobii glasses

Tobii glasses are similar to the SMI glasses, only they use a Dark pupil detection technique instead of light. Like the SMI they use an infrared light source, but it is not directed into the eye, but on the face and iris. The Tobii glasses have been used in a host of market research and scientific studies that required gaze tracking.
**Calibration**

Tobii glasses need to be calibrated before they can be used effectively. The calibration technique comprises the wearer looking at a grid on a monitor and following on-screen instructions. Once calibrated, the glasses must be kept still on the head. If there is any movement on the head of the wearer, the calibration has to be repeated.

*Figure 1 a part of the Tobii calibration*

The Tobii glasses can be found on online retailers for $15,000.
2.4 Cheaper alternatives

Not all gaze tracking techniques make use of expensive devices. The ITU Gaze Tracker, an open source program developed at the IT University of Copenhagen. It makes use of normal webcams which have IR night vision capabilities. Relying on either natural lighting conditions or head mounted IR LEDs, it uses a dark pupil finding technique.

ITU research group recently produced a paper on an experiment they conducted, directly comparing their cheap system with the two other devices above. The test involved participants using the various gaze tracking systems to type words as fast as possible. The results showed the cheap solution had very similar performance. For the test the ITU system was set up with IR LEDs and a webcam, that were mounted into a wooden block that was being held in the participants mouths.
3 Implementation

3.1 Collecting data

My first step towards building this program was collecting sample data from a front facing camera. I wanted to get a collection of images that would be a typical representation of the data that the final app would receive. I had hoped that some conclusions could be draw about this data that would help in the development of the vision filter.

To get useful pictures I designed an app that would show on the phones screen a marker that would traverse from the middle to the four corners of the screen, taking a picture at each corner.

![Figure 3: Screen shot of the data collection app](image)

I collected 120 images from several people and in a variety of different lighting conditions. I went through these images looking for possible problems and opportunities as to where the user’s eyes were.
3.2 Collected data conclusions

3.2.1 Image reduction
In the interest of saving computation time and memory, the image that is processed should be minimised. Reviewing the sample data I collected I found that due to the angle that most people hold a smart phone relative to their faces, the users heads and face only occupy the lower two thirds of the screen.

For finding the pupils I make the assumption that they’re in the bottom two thirds and discard the remaining image. This cuts the work done by the program by a third.

3.2.2 Face position
To track the users gaze, the eyes and pupils need to be found. With this in mind It is important to be able to make assumptions about the orientation of the faces that will be present in the image. In all images collected, the face is level, and facing the camera straight on

3.2.3 Blurred images and movement
A surprising revelation from the sample data I collected was the amount of movement that that was expressed in the images. While the app was taking their picture, participants were asked to hold the phone as still as possible. Even with the phone in careful hands, there was considerable movement in some images, as the participant adjusted the phone. This is an issue that had to be addressed to in order to get usable data.
3.3 Locating the eyes

As most state of the art systems for tracking gaze make use of infrared images, and a night vision equipped smartphone is not something that can be relied on I looked into several other possible methods of finding the eyes.

3.3.1 Hough circles
Hough circles is a feature detection method for looking for circles in an image. The pupils of the eye should be a good circle shape in an image that shouldn’t contain many other circles.

Hough circles in OpenCV works with the output from an edge detection function canny. Canny uses a combination of the images intensity gradient and the gradient direction to return an image with just the edges.

![Image](image_url)
*Figure 6 canny edge detection, colour (middle) and grayscale (right)*

Above is the original sample image, the grayscale canny output, and in the middle is the RGB canny output. To use the colour image in canny, it must be split into the three component parts of red, green and blue and then after canny has been applied, merged back together. An RGB edge image can be useful to find edges only visible in one colour channel, helping to define the edge between skin and the white of the eye.
3.3.2 Hough circles implemented

Figure 7 Hough circles output (left), canny input (right)

The image above shows the canny input beside the results from the Hough circles, it has found one of the two pupils in the image. The input sample image shows the full pupil and has favourable lighting. The image below shows the output from canny on a different sample image. In this sample the pupils are partly obscured and thus the Hough circles has only found noise. After some time spent adjusting the settings for Hough circles, I decided that it was simply not reliable enough for the job.
3.3.3 Skin segmentation
An alternative method for finding the eyes in the image is finding the skin, and then looking for the large gaps within the skin. The eyes shouldn’t come up as skin, if the skin detection method is accurate enough, providing two holes where the eyes are.

Skin in an image is simply a range of skin like colours. A problem with detecting skin is the amount by which skin can vary depending on lighting, skin reddening and a person’s skin tone.

There are different techniques for finding the skin range, but most are essentially defining a range of values that the pixels can be. Other techniques use different colour spaces, other than RGB, such as HLS(Hue, Luminescence, Saturation).

HLS was designed as a colour space that would fit human’s understating of colour better. Hue is the colour, Luminescence how bright the colour and Saturation the intensity of the colour.

Using both of these spaces I used the ranges that were used in Nadira Nordin’s project for skin segmentation and implemented a skin detector. Before detecting I normalized the image to attempt to reduce the effect of lighting conditions. Normalizing the image is diving each colour intensity by the average of all three, eg red normalized = red/(red + green + blue).
3.3.4 Skin Segmentation implemented

The image below shows the output of the skin segmentation program. The technique picks up a lot of noise from the background. Most objects that are cream or light brown are also found as skin. The eyes can come out well, given favourable conditions.

![Figure 9 skin pixels are shown in white](image1.png)

I found that this method was unreliable over the sample images that I collected. Often the eyes would be unrecognizable from the noise in the image, and even when the eyes were definable, the white of the eye would alternate from being skin to not, making it difficult to design any detection method from it.

![Figure 10 the gaps of the eyes are smaller than many noise spots](image2.png)
3.3.5 Haar Cascade Classifier
The Haar cascade classifier is a method implemented in OpenCV based on a paper by Paul Viola and Michael Jones. The paper describes a trained feature detection method, and uses facial features as an example.

The feature detection is based on splitting a grayscale image into rectangles of interest and calculating the difference in the sums of the values of adjacent and diagonal rectangles.

![Figure 11 feature calculation](image)

The image above shows the four different features that are calculated for each section of an image. A is calculating the difference in the dark rectangle adjacent to the white. B is the difference of the rectangle above and below. C is the difference in sum to the middle rectangle to the exterior rectangles and D is the dark rectangles sum minus the white diagonal pairs. To get the features for the entire image the sums of the rectangles must be calculated. To speed up this expensive process, Integral images are calculated.

An integral image is one where each pixel is the value of the sum of all the pixels above and to the left of it. This approach allows for the sums of all the regions of an image be calculated on a single pass. To find the sum of any rectangle in the image, find the difference between its top left corner and bottom right.

To save calculations the detection method makes use of a cascade structure. Each region must pass each feature comparison one after the next. If a region fails a feature comparison, the region is dropped and the program will move on to the next section.
To train the program, thousands of images need to be fed into the feature classifiers, the paper used 4916 positive images, where the feature being searched for is present, and 10000 negative images.

The results of this is fed into an adaptive boost algorithm that calculates a weighting system for the classifiers.

The program can be trained with any feature, and in the OpenCV library there are trained data files for facial features such as eyes and face.
3.3.6 Using eye cascade classifiers

The OpenCV library include a left and right eye trained cascade classifier that has been trained on the front facing images of the respective eyes. From my sample data I assumed the user’s eye would always be front facing towards the camera, and so I used this classifier in a program that ran through some of the sample images.

The program performed the feature detection on multiple scales and through the whole image. The program ran at slow speeds of 1 or 2 frames per second, and couldn’t be used every frame for a usable gaze tracker. The OpenCV function that implements the cascade classifier returns all rectangles that have passed as left and right eyes. To be useful, these would have to be reduced to two.

Using some simple assumptions the best pair can be found.

- The eyes should be the same size
- They should be on the same level
- Not more than two eye widths apart
- Not less than one eye width apart

Figure 12 green squares drawn around the expected eye areas

Figure 13 after screening for best pairs
3.4 Finding the pupils

The cascade classifier only finds the region that the eye is in, to track the gaze the user’s pupils have to be found in the image. To find them, the assumption is made that the pupils will be the darkest points within the eye region, and so finding the darkest point, that is large enough, should uncover the pupil.

As the pupil can contain reflections of light at its centre, to get a good position for the pupil centre the colour will need to be clustered together. The clustering of colours will smooth out any noise and give the pupil as a single dark region.

3.4.1 K means Clustering

K means clustering is an iterative clustering technique that chooses data points as random starting centres, and then moves this centre in order to minimise the distance from all closest data points.

An RGB image can be thought of as a set of 3 dimensional data points, as shown in the image above, if Dim 1-3 were the red, green and blue channels of the image. To cluster the colours of the image, we want to collect all the pixels that are close to each other, and change them to the sets average. The average colour values should be those that minimise change that each pixel must make. To find a good average centre, K means picks a number of random centre points. Each pixel is then assigned to which ever centre they are closest to, using the Euclidian distance.
With all pixels assigned, the centres are recalculated as the average value of all pixels assigned to it. This moves the centre, and so a new set of pixels are assigned. The process is repeated either a set number of times, or until there is an acceptable average distance to the centres.

The algorithm requires significant computational time for each added pixel making it a very slow process. It is improved by the fact that it is only being performed on the small regions that are found by the cascade classifier, but for a usable application the process is too slow to perform every frame.

### 3.4.2 K means implemented

The image below shows the left eye region, picked out by the cascade classifier and then run through the OpenCV implementation of K means clustering. The process is then finding the darkest centre. The centre that has the lowest total of red, green and blue values is found and all pixels that are within this cluster are set as one, all other pixels are set as zero.

![Figure 15 before (left) and after k means clustering](image)

The results are a binary image, where the only white should be the pupil. However, to take into account the possibility of the eye brows being included, as well as shadows, a verification procedure is needed.
### 3.5 Verifying pupils

The binary image returned by the cascade and k means processes may contain many different white regions. Noise, shadows and dark eyebrows could be present. To remove these possibilities I make use of some properties of eyes, discussed in this paper\textsuperscript{15}. To compare the two eyes, I produce pairs out of all white regions found in the binary image, and perform the following checks.

- Each region in the pair has an area at least as large as 1/8 of the eye region
- Each pair is on the same level.
- Within one and three eye region widths of each other.
- If more than one pair remain, the pair with the lowest average position should be the eyes. This should remove the possibility of picking up the eye brows.

![Figure 16 a valid pair of eyes must meet all assumptions](image-url)
3.6 Tracking the pupils

The implementation of finding the users pupils so far has been extremely processor intensive, K means and the Haar cascade classifier are both processor intensive, and when used sequentially the frame rate becomes unusable.

Rather than finding the pupils in every frame I track them from frame to frame. Starting the search from the last known position and refining it for any movement in-between.

I select an eye region around the last known position of the pupils centre. The region selected is five times the size of the pupil, allowing for differences in scale, from a close up head to one further away. Finding the dark spot of the pupil is done through a binary thresholding operation.

Binary thresholding is a simple operation on a grayscale image. Any pixel whose value is below a set threshold is set to 0, and any above is set to 255, or white. The key to finding the pupils through this method is first finding an appropriate threshold to use. Using a fixed threshold would make the app dependant on lighting conditions being the same as the ones that the threshold relate to.

To find an appropriate threshold, I use the last frames pupil position. For each pupil that is found, the average grayscale value is found. To do this all the pixels outside the pupil are set to 255, and the average is calculated by running through the whole region, summing all pixels that are below 255.
The thresholding returns a binary image, which is treated just like the results of k means. The objects found in the image are made into pairs, and validated. If a valid pair of pupils are not found in a frame, then the eye finding function is called again until it is found.
3.7 Flow chart

Below is a flow chart illustrating the final implementation of the eye tracking program. It starts at the top where video is captured from an android device and the transferred to the PC to be processed. In the chart, the fast binary track is shown in green, and the slower initial track in yellow.

Figure 18 flow chart of the final eye tracking program
3.8 Calibration

To track the gaze of a phone user, there has to be a way to correlate the position of a pupil in an image, and the part of the screen the user is looking at. This is done by way of a calibration period, where the position of the pupils are measured as the user follows a pointer around the extremes of the screen. This gives pupils positions to screen locations for the current position of the phone.

After the app has been calibrated, the pupil positions discovered can be given as a ratio of the calibrated values. If XCmin, and XCmax are the x pupil positions for the minimum and maximum screen locations, then a recorded pupil point is XR. XR is compared to XCmin and XCmax, if it is between the two, then the user is looking at the screen. I get a position on the screen by getting XR – Xcmin as a percentage of XCmax – XCmin.

![Figure 19 screen capture of calibration process, y-max and x-max (left), y-min and x-min (right)](image-url)
4 Results and discussion

The program is not running on an android device, but instead using an android app to collect data and then processing it on a PC. This is for two reasons. First is complications in porting the OpenCV library over to the android environment. The second is performance constraints. The android device available ran at such low frame rates when just at the eye feature detection level, it is likely that the full eye finding method would be so slow that any eye position information would be out of date by the time it had acquired it.

As a result some features were not practical to implement, such a detection of camera movement. To address the constant movement of the camera while in use, I had planned on using the phones in built accelerometer, to detect when the phone had been moved violently. This feature would have invalidated any calibration attempt where a major shake was detected, and the calibration could be run again. As the program was running on a PC this was not implemented, and as a result the program is reliant on a stable calibration period.

The data used for the program comes from a simple android app that captures video from the front facing camera, while displaying a pointer on screen. The pointer moves around the screen as the user follows it with their eyes. The results from the gaze tracking can be set beside a screen shot of the app.

I decided to consider the implemented program in two sections, first as an eye detecting method, and second as a method for calculating a person’s gaze.
4.1 Eye tracking

Out of 550 frames of test data that the program claimed to have found the pupils in, it had found a point on both pupils in 430 of those frames, and was wrong, meaning the point it had selected as the pupil was not at any point on the pupil 120 frames. This measurement is based on the point just being on the pupil, and not necessarily at the centre of the pupil. This makes it easier to collect data on large scale runs.

To calculate metrics for these results I’ll take true positives (TP) to be a frame where the program successfully found a point on the pupil. False positives (FP) as a frame where the points found were not on the pupil. True Negatives, correctly identifying instances in which the object being looked for is not present, are irrelevant for this test, as all the frames contained a pair of eyes. False negatives, where in no object was found when there is one present will be the those points that the program could not get a location for the eyes.

![Figure 20 positions of the pupils centres shown by yellow circles](image)

**Recall**: is the percentage of the objects being searched for that were located. In this instance, frames that had eyes, in which the program found eyes.

\[
recall = \frac{TP}{TP + FN} \quad \frac{430}{430+0} = 100\%
\]

**Precision**: The percentage of those claims that are correct.

\[
precision = \frac{TP}{TP + FP} \quad \frac{430}{430+120} = 78\%
\]
**Accuracy**: The percentage of total samples that are correct. All pupil position given that were correct.

\[
\text{Accuracy} = \frac{TP + TN}{Total \ Samples} = \frac{430 + 0}{530} = 78\%
\]

**Fβ**: A weighted measure of precision and recall, with weight \(\beta = 1\).

\[
F\beta = (1 + \beta^2). \frac{(Precision \cdot Recall)}{((\beta^2 \cdot Precision) + Recall)}
\]

\[
(1 + 1). \frac{78.100}{(1.78) + 100} = 87.6\%
\]

The most important figure for this eye tracking method is the Precision. The program could produce false negatives without producing inaccurate data, it simply wouldn’t produce any data for those instances. The false positives are the readings that cause a real problem for any application of this eye tracking, as they do produce inaccurate data.

The adverse effects of these false positives can be seen later during the calibration process, where one inaccurate reading can ruin the entire process.

With that said, looking at the process in a more general way, getting accurate data 78% of the time can still be useful for some applications. The recall of 100% could make this method useful if finding the eyes was the only goal, and not tracking them precisely. A program that kept track of whether a person was behind a computer screen, or absent could make use of it.

With finer tuning, it could be possible to bring down the number of false positives, at the expense of the false negatives. The constraints for valid pupils could be heightened. Simple tweaking of these constraints returned varied results, but often produced far higher false negative readings, and more precision.
4.2 Gaze tracking

A problem with attempting to test a method for gaze tracking is that it is difficult to verify if the tracker’s claims are correct or not, as reliable gaze tracking data can be difficult to get. To test the gaze tracker’s ability, I collected recordings of eyes tracking a point on a phone screen. The gaze tracker should return a position similar to that of the real pointer position.

4.2.1 Calibration issues
To relate the position of the pupils in an image to the phone screen, a period of calibration is needed. If this process is not completed perfectly, the resulting gaze information is very poor. As seen above the eye tracking’s 78% precision becomes a big problem in calibration.

4.2.2 Graphs
The following graphs are a plot of the gaze tracking screen coordinates, and the app’s screen pointer coordinates, on a frame by frame bases. The y axes represents the value of the x or y coordinate. For this discussion, x coordinate refers to the width of the screen, and y the height.
In blue is the x coordinate on screen of the pointer the participant was looking at. In orange is the x coordinate that the gaze tracking program thinks the participant is looking at. Both coordinates are plotted next to the frame number on the x axes.

In blue is the y coordinate of the screen pointer, plotted on the y axes against the frame number on the x. In orange is the gaze tracking y coordinate against the frame number.

The negative numbers around frame 275 are likely caused by an error in calibration, making it seem that the end of the phone screen is closer than it is. The overall higher values of the gaze tracking coordinates are likely caused by bad calibration also.
The stepped motion of the gaze tracking positions is due to the use of an integer value for the pupil position, leading to a truncation during the division necessary to get the pupil positions as a screen percentage.

The performance of this program as a method for tracking the users gaze is poor. Although there are several points in which the program appears to track the motion the screen pointer has, these points are when the pointer is stopped. From frame 60 to 110 for example. There are large portions of these graphs in which the gaze position and screen pointer don’t appear to have any correlation. At frame 225 on the y coordinate, the screen pointer is on the very bottom, while the gaze tracker has several high jumps.

The precision recorded for the eye tracking method is backed up by these results, as there are the occasional jumps and dives for no discernable reason other than the eye tracking has found something that is not the eye pupil.
5 Further Work

5.1 Working Camera app

Obvious further work would be to get the method implemented on an android app. The speed issues could be helped by using a more modern smartphone. The phone used for this project had a 2 core 1 gigahertz processor, compare that with the Quad core 2.5 gigahertz snapdragon found in the new Samsung galaxy S5. If the S5 is an example of the next generation of smartphone, the app should become quite usable on those devices.

Further work could be done to separate the work onto different threads. Although most of the image processing would have to be done in sequence, capturing the images and handling calibration could be done concurrently on separate threads, improving performance.

5.2 self-calibration

An obvious flaw in the design of this program for any practical use is the calibration period. Every time the phone is moved significantly the calibration would have to be performed again. It would be unrealistic to expect people to use any service that would stop constantly to calibrate. Considering the expectations of modern smartphone users, if the app was to be used at all, it would have to run seamlessly.

A possible solution to this is a calibration that used the normal inputs to the phone as the calibration data. The app could take the touch inputs and use them to get a bases for where the user is looking at the screen at the time. As a person scrolls down a page of text, the assumption can be made that they are looking at the bottom of the screen. If a button has just been pressed, the app can assume that the user is looking at or very close to the position of the button on screen.

This would allow for constant calibration, and for verification of the positions found. If the app is reporting a position at the top of the screen, and the user presses a button at the bottom, the position data up till the last verification could be marked as unreliable.
5.3 Security concern

Any app, program or service that makes use of, or collects large quantities of user’s data will have its security come under scrutiny. For example, when Apple released the IPhone 5s with a finger print scanner, a major concern among consumers was the security of their finger print information. There were justified fears of finger print data being stolen by either a criminal organisation, that could use it to help steal an identity, or more realistically, by a government organisation such as the NSA. If an app that tracks users gaze is going to be accepted, it would have to be proven to responsibly handle any data it collects. In the IPhone, the solution was keeping all sensitive data on separate hardware, and only ever sending a key produced by this hardware xvii.

Similarly, the app could be designed so that the images captured from the camera are at no point saved to memory. The image could be processed and only the eye position data kept. The eye position data, and complimentary data on what app it was collected from could still be sensitive information though. On-board encryption using an asynchronous key could minimise this concern. The data could be encrypted with an algorithm such as RSA, where the data is saved on the phone in encrypted form. Only when the data has been collected can it be decrypted using the private key of the app owner.

![Diagram of encryption process]

RSA is an asynchronous encryption algorithm, used often in banking systems and to pass certificates of websites. It works off the bases that encrypting data with a widely known, public key produces a trap door function which can only be undone correctly with access to the closely guarded private key. This algorithm is perfect for an app communicating with a
server, as the app doesn’t need to store any important key information. The public key can also be made short, speeding up the encryption process at the expense of slowing down the decryption process.

Another potential security concern would be any collection of touch data, for calibration during use of the key pad. The app would need to detect the use of the key pad and stop collecting data. If not, a user’s bank details and personal information could be needlessly collected and put at risk.

5.4 Pay per Gaze implementation

As part of the implementation as an android app, I would like to include the ability to collect gaze statistics on a specific advert. If the gaze tracking was implemented alongside a working app, preferably one that was popular and cost money. The gaze tracking could be used to give data on how many times and for how long the users looked at specific ad bars, and this could be used to bill the advertisers per look. The app could be offered free to those willing to take part in the gaze data collection.

Just weeks before this project’s completion, Google was issued a patent for a “Pay per Gaze” advertising system, for use with their Google glass technology xviii. The patent may change the way that mobile adverts are paid for, and make the use of an app such as this one viable.

Conclusions

The goal of this project was to track a smartphone user’s gaze using a smartphone app. Most of the essential building blocks for that app are now complete. Although the precision of the gaze tracking is not to a level in which it could be used to pick out individual buttons of lines of text, the strength of data acquisition from apps is always in the scale. The gaze tracking only needs to function correctly one time in a hundred, and with millions of people using the app, information on thousands of advertisements can be collected.
DECLARATION

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university

______________________________________________ ________________________
Name Date
http://www.eyetracking-glasses.com/
http://www.ebay.com/itm/NEW-TOBII-GLASSES-EYE-TRACKER-only-hardware-no-manual-/120934451200
http://www.gazegroup.org/downloads/23-gazetracker
http://www.had2know.com/technology/hsl-rbg-color-converter.html
https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf (Rapid Object Detection using a Boosted Cascade of Simple Features)
http://hal-ujm.ccsd.cnrs.fr/docs/00/37/43/66/PDF/article_ivcnz2006.pdf (A simple and efficient eye detection method in colour images)
http://en.wikipedia.org/wiki/File:Iris_Flowers_Clustering_kMeans.svg
A simple and efficient eye detection method in color Images by D. Sidibe, P. Montesinos, S. Janaqi
http://hal-ujm.ccsd.cnrs.fr/docs/00/37/43/66/PDF/article_ivcnz2006.pdf
http://www.phonearena.com/phones/Samsung-Galaxy-S5_id8202