Gaze Tracking for Screen Navigation

Rupert Christie
B.A.I. Engineering
Final Year Project April 2014
Supervisor: Dr. Gavin Doherty

School of Computer Science and Statistics
O’Reilly Institute, Trinity College, Dublin 2, Ireland
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.

___________________________________
Rupert Christie

2/4/2014
Acknowledgements

I would like to thank my supervisor, Dr. Gavin Doherty, for all of his advice throughout this project. I would also like to thank Akira Hayakawa for the encouragement and use of his office for testing throughout this project.

Thanks also goes to my friends, who gave me their time to test the usability of my system.

Finally, I would like to thank my family for their continued support.
Abstract

Gaze tracking is a growing area due to the interest from consumer electronics companies as well as companies looking to develop systems to aid people with motor or dexterity impairment. There are two systems currently being researched, display mounted trackers and head mounted trackers.

The aim of this project is to develop a navigation system using a head mounted eye tracker. The system should not be affected by head movement.
# Table of Contents

Declaration.........................................................................................................................i

Acknowledgements..............................................................................................................ii

Abstract..................................................................................................................................iii

Table of Contents..................................................................................................................iv

1. Introduction.....................................................................................................................1
   1.1. Motivation..................................................................................................................3
   1.2. Overview of Problem.................................................................................................4
       1.2.1. Getting coordinate data in relation to user’s head.................................4
       1.2.2. Finding coordinates of the screen in relation to the field of view.............................................................4
       1.2.3. Finding where the user’s gaze is in relation to the screen.................5
       1.2.4. Using the screen coordinate for navigation.................................5

2. Background.....................................................................................................................6
   2.1. Similar Work.............................................................................................................6
       2.1.1. Gaze Data as an Input Device.................................................................6
       2.1.2. Using Gaze Data for Selection...............................................................9
       2.1.3. Gaze Gestures for Computer Actions.............................................10
   2.2. OpenCV ..................................................................................................................11
   2.3. SMI Eye Tracking Glasses & iViewNG SDK.............................................13

3. Execution.......................................................................................................................15
   3.1. Initialisation of Server and Subscribing to Data Stream.............................16
   3.2. Finding the Corners of the Screen using OpenCV...............................18
3.3. Using Screen Corners to Find Screen Coordinates of Gaze...........21
3.4. Using Screen Areas for User Input...........................................22

4. Evaluation......................................................................................23
  4.1. Successes..................................................................................23
  4.2. Example of the System in Use..................................................24
  4.3. User Tests..................................................................................26
    4.3.1. Accuracy Test.................................................................26
    4.3.2. Usability Test...................................................................27
  4.4. Difficulties................................................................................29
    4.4.1. OpenCV.................................................................29
    4.4.2. Server Connections..........................................................30
    4.4.3. Noisy Gaze Data..............................................................31
  4.5. Future Work..............................................................................31

5. Conclusion....................................................................................33

6. Bibliography..................................................................................34

7. Appendices....................................................................................35
1. Introduction

Gaze tracking is an area of interest for many different groups which include web designers, advertisement companies, medical researchers, electronic device manufacturers and people with motor or dexterity impairment. These groups are either using gaze tracking as a research tool to understand how our vision system works and what stimulates it or as part of a device that will improve their everyday lives.

Web designers have used gaze tracking studies to understand what catches a user’s eye and use this data to improve their website. They often use directional cues, such as arrows, to direct users through their website. A recent study by KISSmetrics (Ciotti) showed that by simply having a gaze drawing object, such as a face, pointing towards another object, a person’s gaze would move in that direction. This is illustrated in figure 1.

![Figure 1](http://blog.kissmetrics.com/eye-tracking-studies/) (27/3/14)
Advertisement companies have also used studies like this to learn how they can draw people’s attention to important information in their ads. They also use gaze tracking to study how their advertisements are performing against other ads. Figure 2 shows a study done by Tobii, a leading gaze tracker manufacturer, on advertisements within pubs.

Eye tracing is also starting to be used in medical research and diagnoses. A study by University of Texas and University of North Carolina (Sasson & Elison, 2012) has shown that early signs of autism can be detected using eye tracking techniques. For example visual fixation on social objects can be mapped to determine how many objects are focussed on per second. It is also used to study vision deterioration.

Electronic device manufacturers have started implementing eye tracking within their devices. Samsung have introduced a “smart scrolling” system within the

---

2 http://www.tobii.com/ImageVaultFiles/id_977/cf_2/st_edited/PxxpX289BPHsS7FDyram.jpg (27/3/14)
Galaxy S4 which allows you to scroll up and down a page using your eyes. This is quite useful for people who would have reduced dexterity in their thumbs making it hard to scroll.

1.1. Motivation

This project was proposed as a way of aiding people with motor or dexterity impairment. As computers become more and more part of our everyday lives, these people are in danger of being left behind. A control system needs to be developed for these people. There are some aids currently on the market using, for example, chin controlled joy sticks or other motor controls. However these devices aren’t suitable for everyone. The project will allow computer control with a person’s eyes and more specifically their gaze. This should help these people navigate on a computer with ease.

There is also an augmented reality element to this project. Many video game designers are now looking to add different control systems into their games. Such a gaze tracking system could be used within a video game as a unique selling point, especially in games using a first-person camera to control the player’s field of view.
1.2. Overview of Problem

The original problem is tracking the gaze of the users. There needs to be some way to get coordinates of a person’s gaze in relation to the screen they’re looking at. In order to do this a piece of hardware, SMI Eye Tracking glasses, will be used. Now the problem can be broken down into four areas.

1.2.1. Getting coordinate data in relation to user’s head

A server programme, provided by the manufacturer, must be initialised and connected to. The eye facing cameras’ video streams are passed through the server to the API where the system calculates the user’s gaze coordinates. It will then return the gaze in XY coordinates in relation to the user’s head.

1.2.2. Finding coordinates of the screen in relation to the field of view

The front facing scene camera is streamed through the server similar to the eye cameras. A computer vision library, OpenCV, will then be used to find the corners of the computer screen. From this an estimated coordinate value of each corner of the screen in relation to the user’s head can be calculated.
1.2.3. Finding where the user’s gaze is in relation to the screen

The relationship between the screen corner coordinates and the gaze coordinates need to be found. From this a screen coordinate should be able to be found.

1.2.4. Using the screen coordinate for navigation

The screen coordinate will then be used to navigate the screen. User gestures will need to be developed so that it is easy to navigate the screen and the user can do what they want.
2. Background

The aim of this chapter is to give a brief background to the project, including similar work that has been conducted and the approaches taken. This will include research done on using gaze data as an input device. It will also provide a brief introduction to some of the computer vision techniques and the hardware used in this project.

2.1 Similar Work

There has been quite a lot of research done on using gaze data as an input device. These have mostly focused on using gaze as a selection device and comparing it to traditional input methods. There has also been work done on control gestures and which ones are the most efficient and natural to the user.

2.1.1. Gaze Data as an Input Device

An analogy used by Edward Tufte (Tufte, 1989) is that the interaction between a human and a computer is a situation where two powerful information processors are trying to communicate with each other over a narrow bandwidth and highly constrained interface. Robert Jacob (Ware & Mikaelian, 1987) then leads on from this to say that “the fundamental goal of research into human-
computer interface is, therefore, to increase the useful bandwidth across that interface."

Research from Ware and Mikaelian (Ware & Mikaelian, 1987) showed that using gaze control as input is faster than traditional pointing devices such as a mouse. They showed that Fitts’ Law, which is a way of predicting how long it will take to rapidly move from one target to another, holds true. The increased speed of selection was due to how rapidly the eyes could move and that there is actually no need for physical movement of a pointer.

However gaze tracking data is very noisy. This is due to the nature of the movement of eyes. There are 7 types of eye movements (Hyrskykari, 1997):

- Saccade – these are sudden rapid eye movements that move the eyes to a new place in the scene. Once this movement begins its trajectory or destination can’t be altered.
- Fixation – this is the stable state of the eye.
- Microsaccade – these are small involuntary movements of the eyes during a fixation. This keeps the information the cells of the retina changing otherwise the image would disappear.
• Pursuit – a saccade movement to follow a moving object.
• Nystagmus – movements of the eye in response to motion of the head.
• Vergence – when the eyes move relative to each other, for example going “cross-eyed”.
• Torsional – movements of the eye in response to motion of the neck.

The noisy data caused by these movements can be seen in figure 3 below (Jacob, 1991). In this figure x-coordinates are plotted against 3 seconds of time.

![Figure 3](image)

The user was asked to look at 3 points. As can be seen the raw data is extremely noisy. This gives a false sense of movement.
around the user’s point of focus even though the user was asked to look at only 3 points.

2.1.2. Using Gaze Data for Selection

Robert Jacob proposed two selection methods for use with gaze input (Jacob, 1991). The first one is an aggregate method where the users looks at an object and then presses a button to confirm selection. The second method uses dwell time meaning that is the user holds their gaze on a certain object for a desired amount of time, the object is then selected. Jacob suggests that the second method is the better and more convenient option. He says that the selection time should be long enough so not to cause false selections but short enough so that the system feels responsive.

Glenstrup and Engell-Nielsen then lead on from this and suggest there should be some feedback for the user when dwell time selection is being used (Glenstrup & Engell-Nielsen, 1995). They used Eyecon, an eye controlled button system used in Eyetracker software, as an example. As illustrated in figure 4, as the user holds their gaze on a button, the pictures of an eye appear. It slowly closes indicating how long the user has held their gaze. When the eye closes, the button is pressed.
2.1.3. Gaze Gestures for Computer Actions

Milekic (Milekic, 2003) suggested a set of eye gestures, shown in figure 5, which could be used as the basis for all computer actions.

These gestures could be used, for example, to turn the page of an ebook. They could also be used as gestures to pan objects on screen.

http://www.diku.dk/~panic/eyegaze/node21.html#SECTION000820000000000000000

(30/3/14)
2.2 OpenCV

OpenCV (open source computer vision library) is a library of functions developed by Intel for real time image processing. Its interface is available in C and C++ with wrappers for other programming languages.

Referenced throughout the project, an IplImage and a Mat are data structures used in OpenCV to store an image and its meta data. IplImage is the older structure which uses a C interface. This means that memory has to allocated and released manually. This memory structure is used in the API provided by the manufacturers of the eye-tracking glasses used. This caused problems with memory overflow which will be mentioned later on in this report. Mat uses a C++ interface so memory is allocated and released automatically.

Images are saved in RGB (Red, Green, and Blue) space in OpenCV by default. However colour processing is very difficult and inaccurate in this image space. HSV (Hue, Saturation, and Value) space is much more accurate. As shown in figure 6, HSV space is arranged in a cylindrical shape with hue ranging from 0-179 and both saturation and value ranging from 0-255.
The function inRange is a thresholding function. This function when provided with a range of values returns a binary image. A binary image is an image with only a single bit per pixel (i.e. black or white). So in the image where the pixel value is within the range, the function returns a 1 (white). Everywhere else it returns a 0 (black).

Erode is a technique of shrinking objects, in a binary image, by removing pixels from the shape’s boundary. Dilation is the opposite, where it expands the number of pixels in all directions simultaneously. An erosion followed by a dilation can remove noise from a binary image as the noise pixels should be set to zero in the erosion and when the dilation happens they will not return to their original value.
Contours are the curve joining all the points, of same colour and intensity, along the boundary of an enclosed shape. findContours function, when a binary image is used as a input, finds the edges of the shapes in the image and saves them to a variable. The drawContours function then draws along these contours to make them viewable. From this you can find the centre of the shape by taking the moment of the shape and dividing by the area of the shape.

2.3. SMI Eye Tracking Glasses & iViewNG SDK

The Eye Tracking Glasses (Figure 7) contains one front facing scene camera and two eye facing cameras.

The glasses connect to the computer by USB. They are secured to the users head by a pull string behind the head. The licence for the glasses is contained on an encrypted USB drive which must be connected at all times while the glasses are in use.
SMI provide an SDK for the iViewNG software. It contains an API for communication between C/C++ software and the iViewNG server. The API communicates with the iViewNG server using TCP and UDP to provide maximum speed (Figure 8).
3. Execution

This section details the method used to solve the problem and includes explanations of the code and reasons for the methods used.

3.1. Initialisation of Server and Subscribing to Data Stream

Firstly the iViewNG server must be initialised so that the API can subscribe to the data stream from the eye tracking glasses which is passing through the server.

An iView server object is created. The server parameters are then set. This includes setting the server to a local host. The API then connects to the server.

Next the connection between the glasses and the server must be established. The licence from the encrypted USB drive is sent to the server to allow the connection to be opened. The glasses parameters are then sent from the API to the server. This includes camera refresh rate.

Once the parameters are set the data stream is subscribed to. For this project, the gaze information and the scene camera are subscribed to.
Once the first piece of data is received an automatic calibration is called. The API uses the gaze information received to perform a calibration and correctly calculate the user’s gaze.

Once this is completed the scene image is opened (figure 9) on screen. There is also an option to overlay gaze information on the scene image illustrated in figure 10 & 11 with a green dot. Figure 12 shows the images from the eye facing cameras.

Figure 9
3.2. Finding the Corners of the Screen using OpenCV

To find the corners of the screen, red markers are stuck to the corners of the screen. This will allow the system to search for the colour red within the scene camera frame.

First the image is converted to HSV space as this is the best space to do colour thresholding in. The resulting image can be seen in figure 13.
The image is then passed to the inRange function which searches for the colour red. This will return a binary image. Any areas with the red colour will show up in white and the rest will be black. The resulting image can be seen in figure 14.

![Figure 14](image)

As can be seen, this image is quite noisy. To remove the noise the image is passed through two erode filters followed by two dilate filters. This gives a much cleaner image which can be seen in figure 15.
The contours are then found in the image using `findContours`. They are then drawn onto the image using `drawContours`. The resulting image can be seen in figure 16.
When four contours are detected, the centre of these contours are then found and saved in an array. They are then sorted into another global array so that the top left corner is stored in the first slot, top right in the second and so on.

Every time a new scene image is received the same procedure was followed. This gave new coordinates for screen corners every 0.5 seconds.

3.3. Using Screen Corners to Find Screen Coordinates of Gaze

Using the screen corner coordinates found, the system is then able to create a bounding rectangle. Using the gaze coordinates in relation to the scene, it is then determined whether the gaze is within the bounding rectangle. If it is, the gaze is given a value between 1 and 9. The number corresponds to one of the 9 split sections which can be seen in figure 17.
3.4. Using Screen Areas for User Input

When gaze is within one of the areas, a counter starts. If the gaze moves outside of the area the counter resets. The gaze data is being refreshed at 30Hz, this means that 30 on the counter represents 1 second. If the user holds their gaze in the area for 2 seconds (60 on counter) the area will be zoomed in on and displayed full screen. To zoom in, Microsoft’s Magnification API is used. The function MagSetFullscreenTransform zooms in using a scale specified and uses the top left hand corner as the reference point. To zoom back out the user must shut their eyes for two seconds. Then using the same function the screen is reset to its original view.
4. Evaluation

This section will discuss the achievements of the project as well as display some results of testing that was done on a limited sample of users (5 people). The difficulties encountered and future work will also be discussed.

4.1. Successes

The system successfully creates a server and a connection between the eye tracking glasses and the API through this server. Currently both the server and the API are contained within the one machine. However the server and the API can be contained on two different machines and connected via TCP/UDP.

The system is able to find the corners of the screen, which are marked by red markers. It does this by using OpenCV to perform image processing functions on the video stream received from the scene camera of the eye tracking glasses.

The system can determine, from the corners of the screen found from the scene camera, whether the gaze is within the bounding rectangle of the screen. If it is, the system can then determine and associate the gaze with a certain area on the screen. The system can keep track how long the gaze is held continuously in each area, with the use of a counter.
The system implements gaze controlled zooming within a Windows environment. If a user wants to zoom in on an area, they must hold their gaze on that area for 2 seconds.

4.2. Example of the System in Use

A user is shown the London Underground map, seen in figure 18.

Figure 18

The user then holds their gaze in the centre of the map for 2 seconds, and what is seen in figure 19 is shown fullscreen. This allows users to examine the map closer and now the station names are much clearer.
The user then shuts their eyes for 2 seconds. This will return the screen to what is shown in figure 18. If the user then holds their gaze for 2 seconds in the bottom right hand corner, what is shown in figure 20 will be seen. Users can now follow the train line they are on and then zoom in closer on the area in which they are to get off.
4.3 User Tests

The system was tested by 5 users. Each user had a different eye condition:

- User 1 – Perfect eyesight
- User 2 – Short-sighted, no prescription lenses
- User 3 – Long-sighted, no prescription lenses
- User 4 – Short-sighted, contact lenses
- User 5 – Long-sighted, contact lenses

These users were selected in order to test would different eye conditions affect the system. Two tests were then carried out, an accuracy test and usability test. The accuracy test was performed by the tester calling out 30 areas that the user had to look at and zoom in on. The usability test was performed by asking the users to find a route between 2 stations on the London underground map. This would give the user the ability to try out the system for themselves and interact with the map when they needed. Feedback would then be given by the user as to how well they felt the system worked.

4.3.1. Accuracy Test

Users were asked to look at an area and zoom in on it. They would then be asked to zoom back out of that area. This was repeated 30 times. The results are contained in the table, figure 21.
<table>
<thead>
<tr>
<th></th>
<th>Correct</th>
<th>Wrong</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>29</td>
<td>1</td>
<td>96.67</td>
</tr>
<tr>
<td>User 2</td>
<td>30</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>User 3</td>
<td>27</td>
<td>3</td>
<td>90</td>
</tr>
<tr>
<td>User 4</td>
<td>29</td>
<td>1</td>
<td>96.67</td>
</tr>
<tr>
<td>User 5</td>
<td>29</td>
<td>1</td>
<td>96.67</td>
</tr>
</tbody>
</table>

As can be seen from the figures above, the system performed well in all tests. The long sighted user with no prescription glasses or contact lenses reported that he found it difficult to focus on an area as it was always blurry. This may explain why the user had the most incorrect tests. This would also mean that it was not a fault in the system. The other inaccurate tests can be put down to small inaccuracies on the border of the 9 areas of the screen caused by microsaccade movements in the user’s eyes.

4.3.2. Usability Test

All users were asked to find a route from one station which incorporated at least one line change on the London Underground. The users weren’t familiar with the London Underground so they first had to search for the location of the stations. All users used the zoom feature to search for these. After they were finished they were asked for their feedback on the system.
User 1 reported that the zooming seemed smooth and he felt the method of selection felt responsive. His only negative comment was that he wasn’t sure when the screen had zoomed out as his eyes were closed.

User 2 complimented the system but felt user interaction was limited. Suggested a scroll system or some way to move around when zoomed in as he felt zooming out and then zooming in on a different area was quite tedious.

User 3 said that she felt the system was unresponsive. Found it hard to focus on an area for 2 seconds, would prefer for selection time to be 1 second. Closing your eyes for 2 seconds to zoom out was also criticised.

User 4 said she liked the system and thought that method of selection was very intuitive. However she didn’t like the method of zooming out. Suggested the method of zooming out should be looking off screen for a certain amount of time.

User 5 really felt the system was very good at selecting areas quickly and was also quite good at the zooming out gesture. User also suggested the system should have more user input developed.
as the system showed promise and could imagine many situations where it would be useful.

4.4. Difficulties

A number of difficulties were encountered during the project. This included problems with the glasses connecting to the server and OpenCV. There were also problems with gaze data being very noisy.

4.4.1. OpenCV

Several problems were encountered while using OpenCV. The first was that an old version of OpenCV (2.2) was built into iViewNG SDK. This version has problems with its memory structures and sometimes crashes for no reason at all. The SDK had to be rebuilt with a new version of OpenCV (2.4.5).

Some of the functions weren’t able to process the scene data in real time. At first, a function called HoughLinesP was being used to find the outline of the screen. However this caused a severe slowdown in the system as images couldn’t be processed fast enough. The system had to be changed to look for the red colour markers as illustrated in the execution section. This system was much faster and was able to process the images in real time.
iViewNG API saves the scene view images to an old format IplImage which need to have their memory manually managed. As the workings of the API are inaccessible the system was unable to manage the memory and the image format couldn't be changed. This meant that memory ran out very quickly, especially when the scene camera is being sampled at 60Hz. To help with this problem sampling was turned down to 2Hz. This helped with the problem but the memory still runs out after a few minutes of running the system.

Sometimes the system has problems finding the red colours due to the changing of natural light in the environment around the screen. This problem has not been resolved yet but a set of scroll bars could be implemented so the user could change the threshold depending on light conditions.

4.4.2. Server Connections

A large stumbling block at the start of the project was that the connection between the eye tracking glasses and the server could not be established. It was later discovered that the glasses were being asked to sample at a rate (100Hz) which the glasses were not capable of. Once the sampling rate was lowered to 60Hz the glasses connected to the server.
The API would occasionally not connect to the server. This problem still occasionally happens. The problem is to do with the UDP/TCP connection but the exact issue has yet to be determined. A workaround for this problem is to close the system and restart it.

4.4.3. Noisy Gaze Data

Originally when the gaze data was sampled, it was very noisy and shaky. This was caused by microsaccade movements and ocular microtremors that occur in everybody's eyes. The sampling of the eye cameras was reduced to 30Hz to fix this problem. At this sample rate the system would miss the tiny, meaningless microsaccade and ocular microtremor movements but was still able to accurately track the gaze movements.

4.5 Future Work

Firstly the user feedback could be implemented. A better method of zooming out could be developed. This could be look off the screen or only close one eye. Also for users that find the system unresponsive a settings menu could be developed where they could specify the amount of time they want for gaze interaction. Scrolling while zoomed in could be implemented quite easily. While zoomed in the user would look off screen
for 2 seconds. The focus would then move in the direction the user is looking off screen.

More user interactions could be implemented. This could include scrolling while zoomed out, button clicks and object dragging. These could be difficult as they will require an extra level of accuracy.

This system could be also be implemented within an augmented reality application. This could be as simple as a balloon popping game. However it could also be used in a game like a first person to move the player’s field of view.
5. Conclusion

The motivation behind this project was to create a system that could be used by people with motor or dexterity impairment. A system has been created to track gaze and has a user input of zooming. This is proof of concept that this kind of system can work.

Further user interactions can be developed to make this a more complete system. Some interactions should be very easy to implement such as scrolling but others such as button clicking and object dragging could need more development as these require an increased amount of accuracy.

In summary, this system using the eye tracking glasses to both track the user’s gaze and the screen’s corners. The system is then able to determine where the user is looking on screen. This information can then be used for user inputs.
6. Bibliography


7. Appendices

Appendix A: Electronic Source

The attached CD contains the source code for the project. It also contains the installation file for the iViewNG Server as well as the iViewNG SDK.