iMouse: Eye Gesture Control System

By

NC Daud Bin Habib | NC Muhammad Abdullah Bin Ashfaq | NC Amna Nasir | NC Hina Aslam

Submitted to the Faculty of Computer Science, Military College of Signals
National University of Sciences and Technology, Rawalpindi in partial fulfillment for the requirements of a B.E Degree in Telecom Engineering
JULY 2018

ABSTRACT

The project is aimed towards the design of an open-source generic eye-gesture control system that can effectively track eye-movements and enable the user to perform actions mapped to specific eye movements/gestures. It detects the pupil from the user's face and then tracks its movements. It needs to be accurate and real-time, so that the user is able to use it like other every-day devices with comfort. The Hardware for this project has to be cheap so everyone can afford and use it, thus a Microsoft LifeCam HD-6000 is used. The project is designed such that it can Control the mouse cursor of a computer for patients suffering from diseases that render movements from the neck down ineffective. It should not be difficult to operate and must be user friendly and allow for easy calibration so that everyone, despite their limited technical knowledge or disability, is able to use it easily. The project works on computers with Linux as an operating System (OS).

ACKNOWLEDGMENTS

We would like to offer our thanks to Brig. Khan Minhas for his supportive attitude, and complete guidance throughout the successful completion of the project, to the well-respected faculty members of MCS for helping by sharing their experience and steering the project in the right path.

TABLE OF CONTENTS

TABLE OF FIGURES

1. INTRODUCTION

1.1 Background/Motivation

Because of exponential progress in the computer technologies in the past decades , the applications of the computers have increased a lot and almost everyone uses computer on daily basis.. With all the possibilities a computer can offer, humans and their interaction with computers are now a limiting factor. This caused a lot of research in the field of human computer interaction (HCI) aiming to make Human-Computer interaction more intuitive, easier, and more efficient. Interaction with computers is not limited to keyboards and printers anymore. Different types of touch-sensitive surfaces, pointing devices, microphones, high-resolution displays, and speakers are normal devices for Human-computer interaction in modern era. There are new modalities for Human-computer interaction like input by gestures or by tangible objects with sensors, speech interaction, the industry is also noting a trend in visual reality goggles the challenge that bounds of HCI.

The aim of our project was to explore new and improve upon existing avenues in eye gaze tracking, particularly those which could help the physically disabled and enable them to use computers and programmable control systems. Thus, such individuals could still take on their responsibilities, improve the quality of their lives and continue with their day to day tasks, often without the need for a helping hand. In present times, most eye tracking systems employ the use of real-time video based tracking of the pupil. We have adopted the same technique as technological improvements have made the high-definition, small, portable cameras readily available at low costs. These cameras can easily be attached to a home-brewed head mount while keeping the overall weight of the mount and the camera to several hundred grams.

Moore’s law has proved to be implementable to date as standard desktop and laptop processors become more powerful with each passing day. A typical laptop processor today is more powerful than the processor used by the super-computer of the past.

These computers today come with built-in high definition cameras which can be exploited to present a field of view or world view against which the user's eye tracking movements can be compared. Modern powerful processors have allowed us to process the real-time video stream for eye tracking and forward the necessary outputs to the mouse or external machinery, on nothing more than a simple laptop computer. The head based mounts, which are necessary to provide freedom of movement to the user, are also video based. They make use of a second camera, carefully positioned to track the user's pupil movements in real time and forward this information to the laptop where after complex comparative calculations, useful outputs are produced.

Employing such techniques as mentioned above allowed us to exponentially reduce the overall project costs and make our prototype cheaper than most commercially available eye tracking devices and equipment. A considerable portion of people affected with Amyotrophic lateral Sclerosis (ALS) or those paralyzed by injury, cannot use their computers for basic tasks such as sending or receiving messages, use the internet or watching their favorite TV show or movies. Even when it comes to eating, they need the help of another individual to feed them. Through previous research, it was concluded that eyes are an excellent candidate for ubiquitous computing since they move anyway while interacting with computing machinery. Using this underlying information in the eye movements could allow bringing the use of computers back to such patients. Not only that, such information could be used to produce necessary outputs for controlling a computer, moving commercially available robotic machinery such as the robotic arm or wheelchairs to enable these patients to feed themselves, to physically enable them and make them, once more, positively contributing members of the society.

1.2 Project Description

This project was undertaken to fulfill the requirements for our Bachelor's degree at the National University of Sciences and Technology (NUST), Military College of Signals (MCS). It falls in the field of Human Computer Interaction (HCI) and shows that by improving upon existing open source frameworks utilized for the purpose of Computer Vision and HCI, a cheap eye tracking solution can be massively produced for the benefit of disabled patients. Figure 1 gives an overview of the system model of the system.

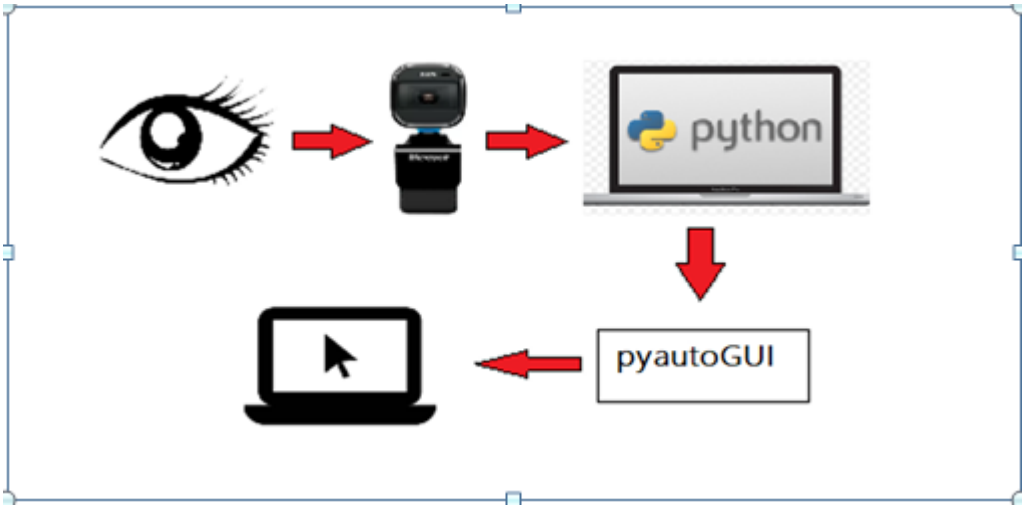

Figure 1: Block Overview of the Designed System

The project prototype takes input from a camera and recognizes the user's pupil and tracks it in real-time. This "tracking" information can then be used by computers or micro-controllers to perform various tasks, some of these tasks that the project aims to achieve is to track the pupil-movement and then store that tracked eye movement to control the mouse cursor of a computer, so that someone with a disability like say Amyotrophic Lateral Sclerosis can use to communicate with others.

On software-end, a program based on OpenCV and PyautoGUI has been developed which consists of get-eye-ball, get-eye, detect-eye-blink and move-mouse functions coded in python that will track the pupil position of the eyes, and use this information to perform the functions mentioned above.

Figure 2 shows Block Diagram of the project's design. To summarize:

1. First the input from a webcam or external camera is taken in real time at a rate of 30 frames per second.
2. From the Input the face of the user is detected using face-cascade from the Haar-cascade. The detected image is then cropped and the eyes of the user are detected using eye-cascade. The detected image is again cropped.

3. From the cropped image, the edges of the pupil are detected and then a circle is drawn around them using Hough Transform. Then the movements of that pupil are tracked and stored in form of numerical values.

4. These variables are then given access to control the mouse cursor of a computer through a module of python named PyautoGUI.
5. Further processing is done and algorithm is formed to control the mouse cursor in real time using these variables.

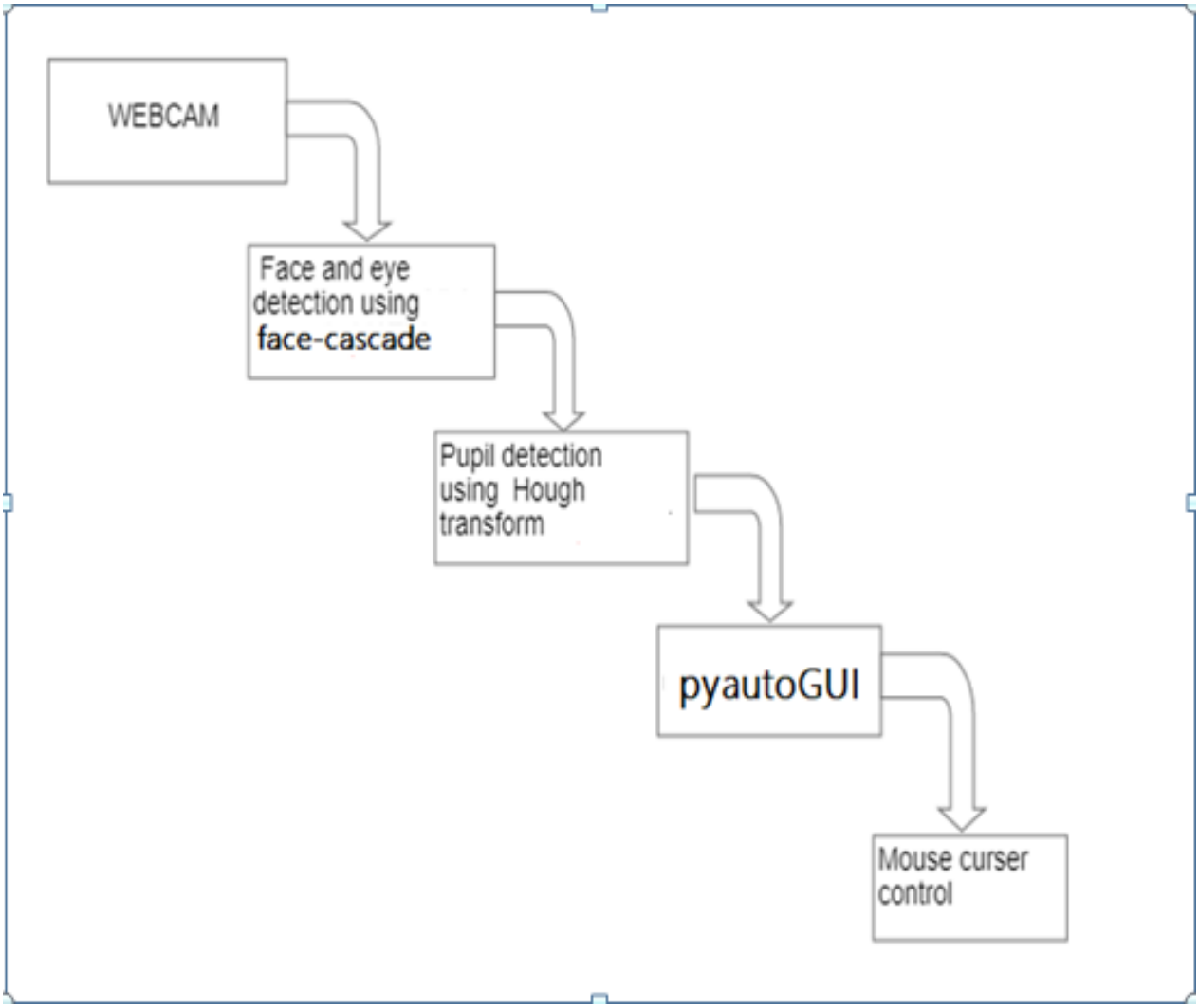

Figure 2: The complete system overview

2. SCOPE, SPECIFICATIONS & DELIVERABLES OF THE PROJECT

The complete project design, as shown in the project description section as well as the prototype, includes a headset or mount with a carefully positioned high definition web camera, an open-platform and easy to install software module compatible with all linux/free-bsd operating systems and easily installed on all modern laptops or desktop personal computers.

This project can be thought of as a holistic transition from the concept to design to proof of concept. It consists of part research paper implementation and part working with open-source community on designing and then building the prototype, all the while ensuring that only open-source and cheap, readily available, commercially off the shelf (COTS) products are used.

The design consists of the following major components:

1. A Microsoft Lifecam HD-6000 Camera with a visible light filter
2. A laptop with Linux as its OS
3. A python code for eye-tracking and mouse-movement.

Microsoft Lifecam HD-6000 is used for input from the user as it has a frame rate of 30 frames per second that meets well with requirements of our project. Also, it has a rotating lens that will help the user to adjust it according to his ease and obtain the best results. Code is written in python and following modules of python are imported for working of the code for this project.

1. Numpy
2. Scipy
3. Opencv-3.4.0
4. Opencv_contrib-3.4.0
5. PyautoGUI

3. LITERATURE REVIEW/BACKGROUND STUDY

3.1 Literature Review

Quite a lot lot of work has been done on this topic in the history

.

A team over at "**Rensselaer Polytechnic Institute**" have done major research in gaze position detection under IR illumination. Their method uses multiple reference points on the entire human face to detect gaze position changes relative to the other mapping points on the image, this technique is also investigated under variable lighting conditions and the team found out that if a light intensity gradient is sufficiently constant in the room, gaze positions are relatively accurately detected under IR illumination and the data set for tracking does not vary too highly.

An open-source software called "**Gaze Tracker" built by the ITU** is available online, work was discontinued on it in 2005 but the project is still maintained by the community. This software will be used as a starting point for this project, since it provides a very basic functional eye gaze detection system. It also operates with the help of OpenCV libraries.

A group over at MIT has also created a project titled "The sixth sense", the project aims to enhance human computer interaction by using gestures from the hands and eyes, along with the entire system being mountable on the user's head, so that it can be projected on to smooth surfaces (like walls), and used anywhere in the world. But it doesn't provide enhanced assistance and accessibility to the disabled nor does it produce a system that can interact with other compatible machinery like the project discussed here can. More details can be found at: *http://www.pranavmistry.com/projects/sixthsense/*

A team over at the Princeton University has also created a general system that only detects gestures and shows that what eye gesture was performed. The system was designed only as a test-bed to find out how gaze gesture detection works. The report serves as a means of knowing how to get started with the basics of eye gesture tracking.

http://www.cs.princeton.edu/courses/archive/fall08/cos436/FinalReports/Eye_Gesture_Recognition.pdf

Though Drewes, Heiko (2010) presents a comprehensive overview, it was noted that most algorithms needed further refinement as they took tedious and long-winded approaches to calibration. A no-nonsense, agile approach was defined Schmidt, Jochen in using structure from motion algorithm for human computer interaction. This was exploited by Kassner, Moritz Philipp, and William Rhoades Patera in using the same sfm (structure from motion) algorithm, optimizing it and extending its use as an efficient algorithm for pupil tracking. In order to achieve this goal they developed a framework by the name of PUPIL, to critically inquire the relationship between a human subject and a space, to visualize this unique spatial experience, and to enable its usage for gaze gesture tracking.

To understand the differences between various gaze gestures and to conclude upon the use of pupil as the most effective part of the eye for tracking, Ashwash, Issa, Hu and Marcotte was extensively studied and together with Drewes, Heiko (2010), provided an immense knowledge-base of the basics to help us get started with eye gesture tracking.

It should be noted that eye tracking in highly lit environments lead to inaccurate results and to overcome these inaccuracies, use of IR (infrared) illumination was stressed by Zhu and Ji to overcome these inaccuracies in variable lighting conditions. They do this by analysing the color gradients of the ambient light via photo-sensors. The hardware for this technique is expensive and is only set up within a single environment, the system is not as flexible and mobile as a robust eye detection module should be, but they do present really interesting conclusions, one being that if the luminance inside your environment is relatively constant on a gradient of light intensity, then eye tracking can be performed efficiently without stutter.

3.2 Background Study

3.2.1 The Human Eye

The eye of a human being works on a two lens system in a fluid, called the vitreous humor, to project the rays of light from different objects in the world onto the retinal surface. Figure 3 gives the structure of the human eye with the various aspects labelled. There is an area on the retinal surface of the eye that is densely packed with cones in proportion to rods. Once the rays pass through the lens, they cast on the

retinal surface after traveling through the vitreous humour. The fovea is measured as 1/4000th of the retinal surface area or an angular diameter of between 0.3 degrees and 2 degrees. This area is called the fovea, and appears as a small yellow spot on the retinal surface. The

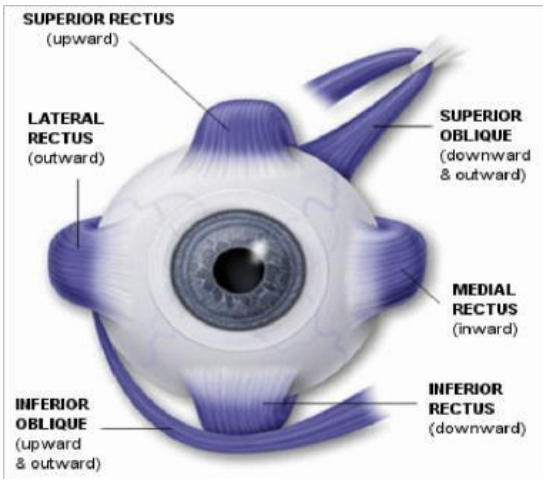

Figure 3: Structure of the Eye

Retinal surface contains two types of photo receptors, rods and cones. The fovea is densely packed with cones, approximately 161,900 per square millimeter, allowing for high resolution colour vision. The physiology of the retinal surface shows us that here is only a small portion of our visual field that we can resolve in high resolution. The remainder of the retina is not blind, as the distance increases from the fovea the density of cones and optical acuity is greatly reduced.

The eye lens can change in shape, or deform in order to focus on light coming from different depths. The retinal surface is covered with photoreceptor cells. So the density of the light should be such that it is not too high that it affects the eye and cause the eye lens to contract and also it should not b too low that the camera becomes unable to track the movements of the pupil.

4. DESIGN AND DEVELOPMENT

Considerable work has been done and all major project milestones have been crossed. To report most of it, we have divided our work into Hardware & Software sections, the bottlenecks and issues faced as well as the achieved results are also mentioned below:

4.1 Hardware

4.1.1 The Camera

The camera used is Microsoft Lifecam HD-6000. This is used because of the 3 main reasons.

1. Webcam of most laptops does not have a good result, as one has to extract a small eye from the whole image and then crop it. Due to cropping size of the image is reduced and its quality also decreases due to which processing this image does not produce good results. Hence an Hd-camera should be used, so one can achieve accurate results even if the quality of the image is decreased.

2. Frame rate of webcam of laptops is very low. As for tracking of eyes a high frame rate is required because eye moves in a very short interval of time and every movement of eye should be recorded for better results. Microsoft Lifecam HD-6000 has a frame rate of 30 frames per second that meets well with the requirement for this project.

3. Webcam of a laptop is fixed at the top of the screen and the user have to adjust his position every time he has to use the computer according to the webcam. Microsoft Lifecam Hd-6000 has a rotatable lens that allows the user to easily adjust the camera position for according to his position.

4.2 Software

This part was done on two softwares.

1. MATLAB
2. Python

4.2.1 Work done on MATLAB

As most of the users use Windows OS, this meant that the software based on Linux would not have been accessible for the masses. Hence first we decided to make this project on Matlab as it provides the best image processing tools on Windows.
In this approach, a program (MATLAB code) is going to be interfaced with an internal or an external webcam, to track the pupil's position of the eyes, and use this information to perform cursory movements. It consists of 2 parts.

1. Eye-detection
2. Cursor-movement

4.2.1.1 Eye-detection

In the eye-detection part, 2 algorithms are used:

1-Viola Jones Algorithm for face and eye detection.

2-Hough transform for drawing a circle around the pupil.

It takes input from the camera in the form of the frames. From one frame it detects the face of the user using Viola Jones algorithm. It detects the face of a person from an image excluding all other objects. But what if there are more than one faces in front of the computer? The program should select the face with the largest dimensions. The reason for that is because when an object is closer to the camera it seems to be bigger as compared to the objects lying at a greater distance. This program considers the person most close to the camera as the user and then detects its face and uses that face for further processing. When it detects the face of the person from the Image it crops the detected image and then detects the eyes of the user using Viola Jones Algorithm. After the detection of the eyes Hough transform is applied on the eyes to detect the edges of the pupil and then it draws a circle around it for further processing. The coordinates of the circle are stored in variables. The program then compares the coordinates of the circle from the previous frame and tracks the movements of the eye. If there is no change in the coordinates of circle from two frames then the eyes are considered still, If there is change in horizontal coordinates then eyes are considered to be moving in horizontal coordinates, if there is change in vertical coordinates then eyes are considered to be moving in vertical coordinates and if there is a change in both horizontal and vertical coordinates then eyes are considered to be moving diagonally.

4.2.1.2 Cursor-movement

The numerical values obtained from eye-detection part are then used to control mouse cursor using JROBOT. JROBOT is a class of MATLAB that gives access to mouse and keyboard. But as our aim is to control the mouse so we will only control the mouse through this. In this screen of laptop is mapped and given the x-axis a value of 1400 and y axis a value of 770 because in most cases the width of the screen of a laptop is more as compared to the length. Then the program takes values from the eye-detection part. If there is no change in the coordinates of circle from two frames then the cursor remains still, If there is change in horizontal coordinates then cursor is moved in the horizontal direction, if there is change in vertical coordinates then cursor is moved in the vertical direction and if there is a change in both horizontal and vertical coordinates then cursor is moved diagonally. Figure shows the output.

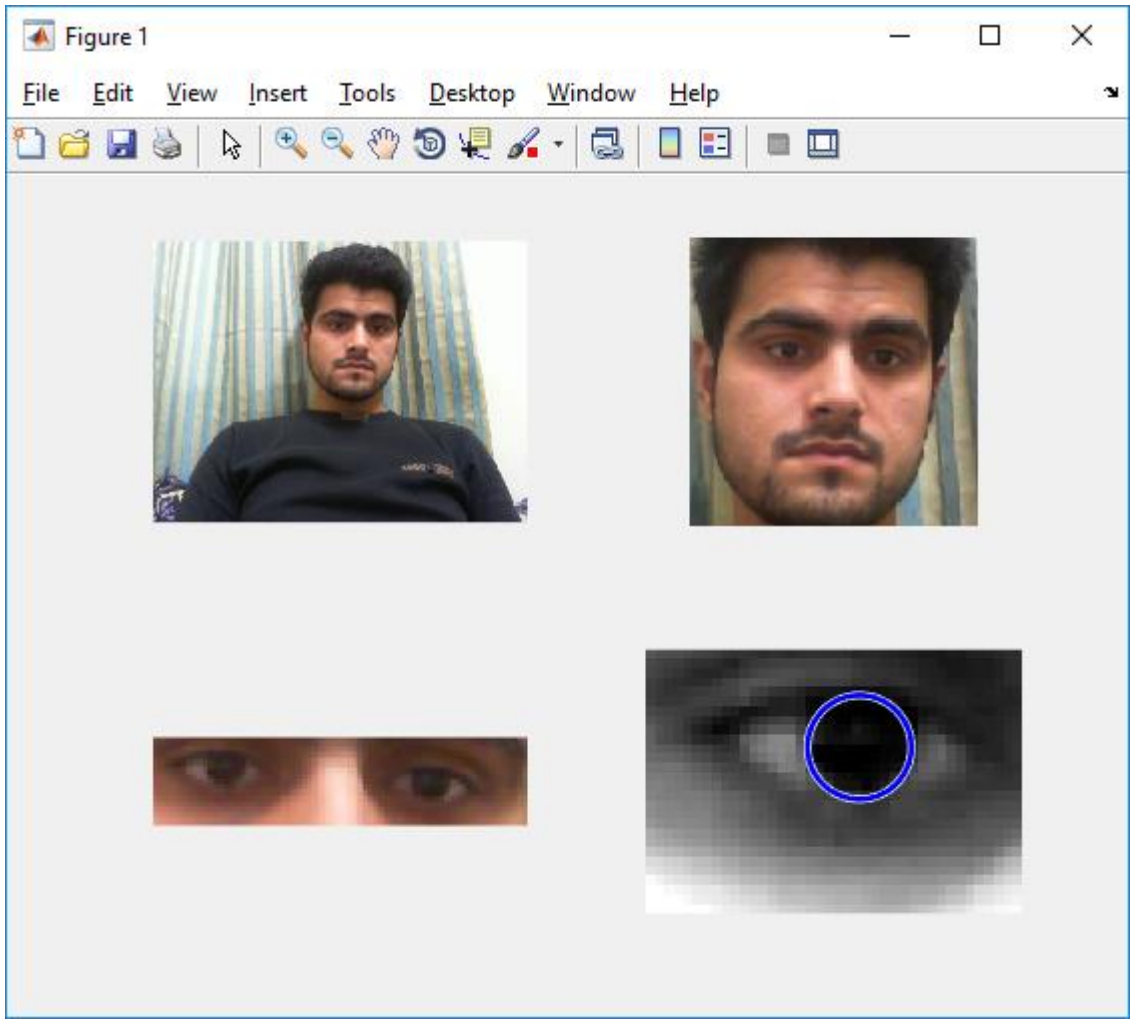

Figure 4: MATLAb OUTPUT

4.2.1.3 Problems faced

The output obtained from this approach was not accurate and hence it could not be implemented in real life. The reasons for failure of this approach are.

1. The processing of MATLAB is too slow. As the aim for this project is to track every action of the eye and then move the mouse according to it. For usage in real life, the software should be fast enough that it should get the input and after all the processing move the mouse cursor in less than a second. But on m MATLAB, it took more around 4 or 5 seconds to move the mouse cursor in response to a given action of eyes. Thus this approach ended in a failure.

2. After researching with faculty of MCS, It was found that MATLAB converts the code into C Language first and then start processing it due to which it was slow. To speed the process a little the code was needed to be converted into C Language or write the project code on linux in python.

4.2.2 Work done on LINUX

After failure of the approach based on MATLAB a new approach was needed. After researching a new approach was selected that could provide a lot better results than the previous approach. In this approach, Linux is used as OS and coding is done on python with opencv. Opencv is a library of python that is mainly aimed at real-time computer vision. Following steps explain the work done in this approach.

1. Installation of the required packages

2. Work on code

4.2.2.1 Installation of the required packages

First of all install Ubuntu on the Laptop. If laptop is already having a windows as its OS then Dual boot the Ubuntu with windows. This will allow the laptop to use both of the OS at the same.

Open the Linux and open the terminal using commands. (CTRL+ALT+T)
Update the Linux using the command.

```
Sudo apt-get update
```

After this step , upgrade linux usig command.

```
sudo apt-get upgrade
```

Install the packages of python that are essential for working of the project and installing opencv by typing commands.

```
sudo apt-get install build-essential
sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev libswscale-dev
sudo apt-get install python-dev python-numpy libtbb2 libtbb-dev libjpeg-dev libpng-dev libtiff-dev libjasper-dev libdc1394-22-dev
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
sudo apt-get install libxvidcore-dev libx264-dev
sudo apt-get install python-scipy
```

Download opencv-3.4.0 and opencv-contrib-3.4.0 and install using commands.

```
sudo apt-get install libgtk-3-dev
cd  opencv-3.4.0
mkdir build
cd build
cmake-
DCMAKE_BUILD_TYPE=ReleaseDCMAKE_INSTALL_PREFIX=/usr/local -
DOPENCV_EXTRA_MODULES_PATH=../../opencv_contrib-3.4.0/modules -
DOPENCV_ENABLE_NONFREE=True ..
make
sudo make install
```

Download PyautoGUI and install it with commands.

```
sudo apt-get install  python-xlib        #pyautogui
python setup.py install
```

This will install Python and all of its packages required for the working of this project.
Now open another terminal and import opencv in python by command .

```
Import cv2
```

4.2.2.2 Work On Code:

In this approach a code is developed on python named imouse.py. Firstly it will open the camera and start capturing video, the video consists of the frames, It will select a frame and then convert it to a grayscale image as it converts images to binary form and it becomes easy to detect objects from that image. It will then detect the face from it using Haar-cascade. Haar-cascade is a cascade function which is trained from a lot of positive and negative images and then it detects objects from other images. It will detect the face from the given frame and then crop that frame and will pass on for further processing. The program selects the face with the largest dimensions. The reason for that is because when an object is closer to the camera it seems to be bigger as compared to the objects lying at a greater distance. This program considers the person most close to the camera as the user and then detects its face and uses that face for further processing. Then from the obtained frame it will detect eyes using Haar-cascade and crop that from. Haar-cascade that detects eye is called eye-cascade. A 4 variable array (i.e spported by numpy) is defined that will give us four values at a time x, y, w and h. x and y are for the starting point from where the camera detects eyes, w is for the width of eye detected and h is gives us the height. By using these variables we draw rectangle around the eye and then crop that image. The rectangle starts from the point x and ends at ends at x+w in the horizontal direction, and start from y and ends at y+h in the vertical direction. The Output is shown in the figure below.

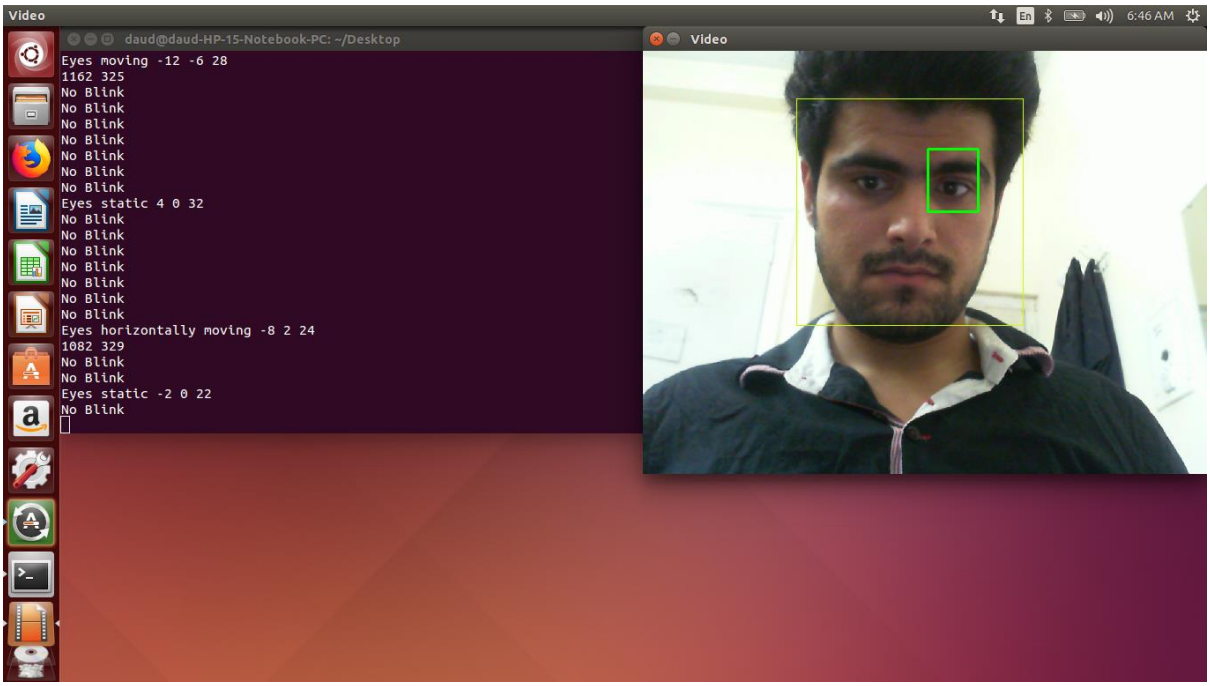

Figure 5: Output of python code

The resulting image will be of very small size the program will blur it to make it clear for our further processing. After that the it will detect the edges of the pupil with the help of the Hough Transform and then draw a circle around the pupil of the eye. Sometime the camera recognizes dark circles near the eyes as pupil. To overcome this problem, detect the dark spots that are only circular in shape and are found in the centre of the rectangle drawn. This way the camera will only detect the pupil of the eye and track its movements.

Define two variables x and y using pyautogui for cursor movement. Y is for the vertical movement, x is for horizontal movement of the mouse. At start both of them are given a random value so when the code will run the mouse will start moving from that point. To detect the movement of the eye, note the position of pupil from two different frames and check if the distance is increased. Human eye is not still, a little bit of movement is always there. To overcome this problem a check is performed that if the difference in the position of the eye is less than 5 pixels then consider eye is

still. If difference in the horizontal position of the eye is greater than 5 pixels and in vertical direction less than 5 pixels then this means the eye is moving in the horizontal direction, If difference in the vertical position of the eye is greater than 5 pixels and in horizontal direction less than 5 pixels then this means the eye is moving in the horizontal direction and If difference in both horizontal and vertical position of the eye is greater than 5 pixels then this means the eye is moving diagonally. When we move the eye vertical, the mouse cursor will move in the upper/down direction and when we will move our eye horizontal then the cursor will move right/left and when we move our eye in diagonal then the cursor will move in diagonal.

5. BOTTLENECKS AND ISSUES

A major issue was that of Auto-Focus. Today all cameras are bundled with Hardware-based Auto-Focus, so during the eye-tracking procedure the camera tried to adjust the auto-focus itself and disrupted our operation. We had to de-solder the autofocus joints in the camera manually. Right now, we adjust the camera focus using the in-app controls.

Another major issue was that of the Operating System to use for our project. Most of the existing Eye-Tracking Projects were aimed at Microsoft Windows and Apple MacOS X, and hence were closed, project-wise. We wanted to work on the Linux Platform so that eye-tracking could be performed easily on devices of all architectures. Finding the Pupil Framework was our answer to this issue because it was originally aimed at the Linux Platform with the same ideologies.

Another major issue was that the project detects the dark circles near the eye as the pupil . To overcome this problem, detect the dark spots that are only circular in shape and are found in the centre of the rectangle drawn. This ways the camera will only detect the pupil of the eye and track its movements.

6. PROJECT ANALYSIS & EVALUATION

We start by presenting results of the working contour detecting algorithm that works by using Haar-cascade functions and Hough Transform. Figures below show the algorithm performing in our framework.

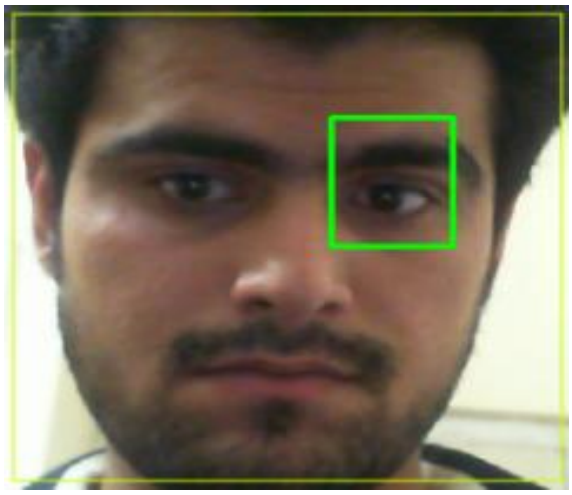

Figure 6: Face-cascade.

It first detects the face of the user from an image using face-cascade. It draws a box around the face and crops the Image for further processing. After we extract the face from the image it detects the eyes of the user from that image. As shown in figure below.

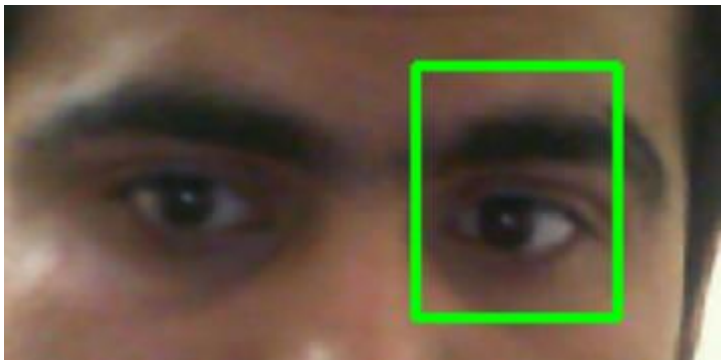

Figure 7: Eye-cascade

As both eyes move at the same time, so the tracking can be done by analyzing the movements of only one eye. This program picks left eye (right eye in image as the image flips in a camera) of the user and track its movement. It draws a box around it crops this image.

Then pupil is detected from this cropped image and a circle is drawn around it to track its movements.

As you can see in the figure below that this project detects the pupil and ignores other dark spots and tracks only the pupil of the eye.

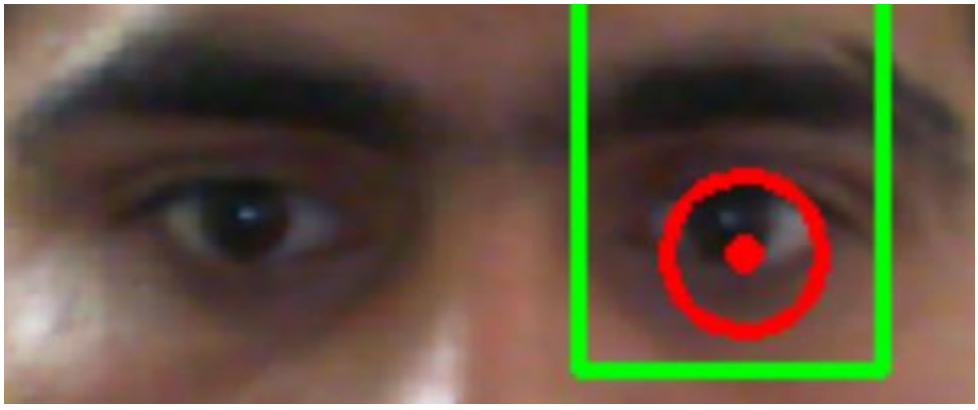

Figure 8: Hough-Transform

Eyes coordinates are then located in each frame. If there is a change in the coordinates of the eyes then the eye is considered to be moving otherwise the eye is considered still. The delay of all this processing is less than one second. Which means this project can be used in real life situations. The histograms for mouse vs. eye control are shown below, please note that the readings were taken for in ideal lighting conditions and may vary in other situations if pupil detection is erroneous. This has been illustrated in figure 9.

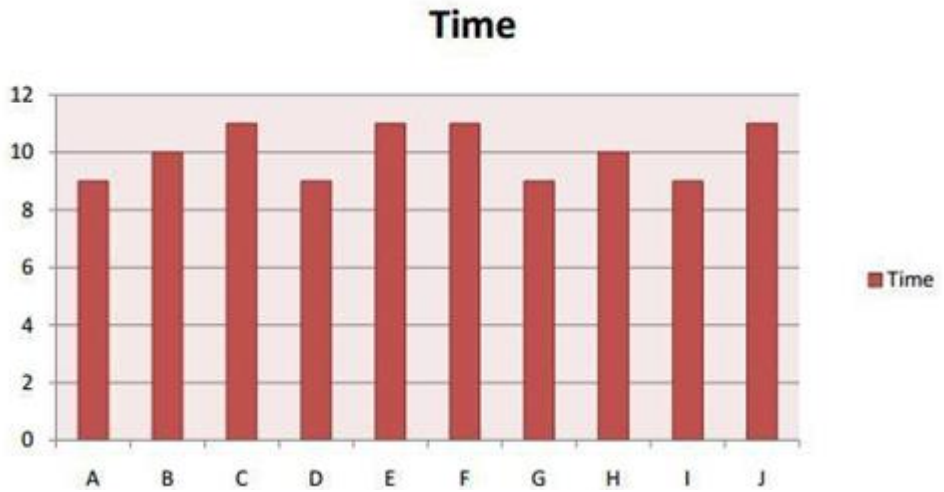

Figure 9: Timing of action A through J.

After tracking of the pupil the coordinates are stored in the variables and mouse cursor is moved according to change in these values and blink is detected. The data transmitted can be seen below:

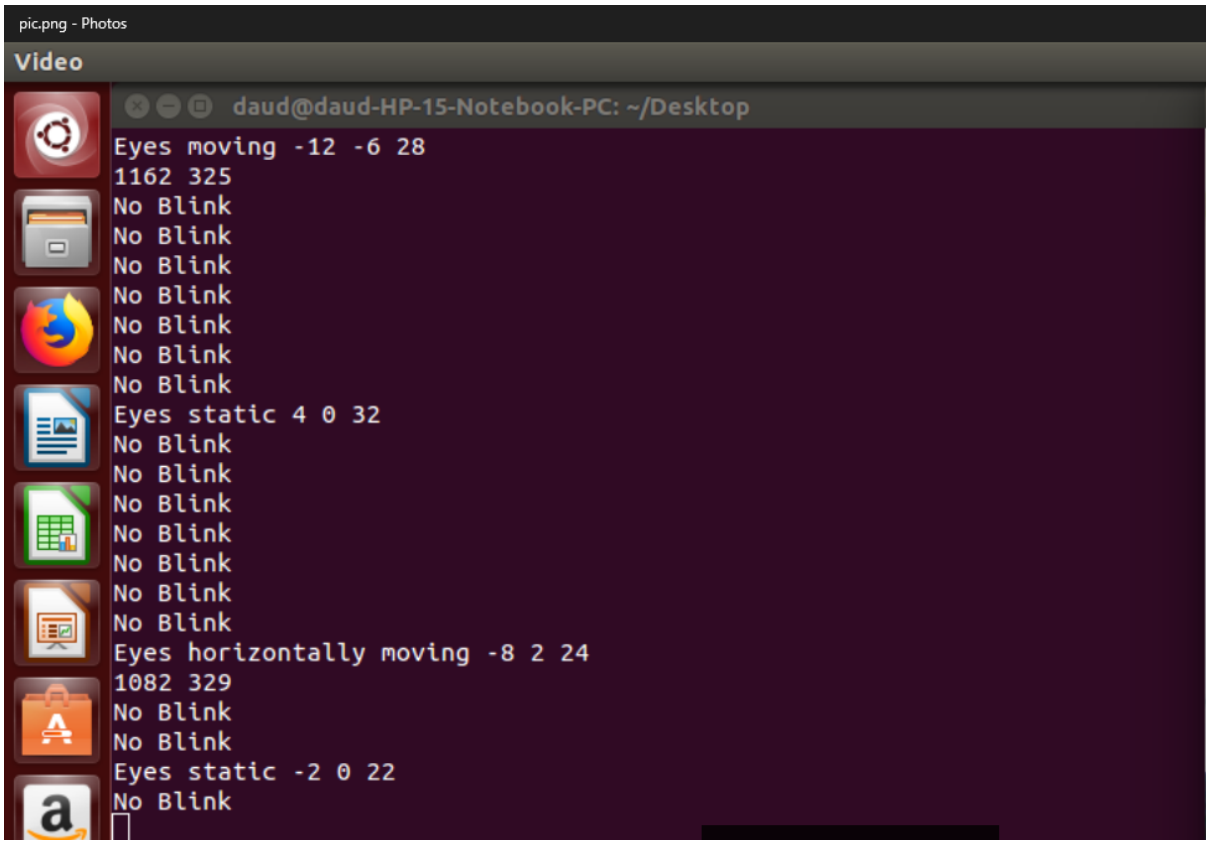

Figure 10: Pupil coordinate values and blink detection.

Note that during a blink, the transmitted co-ordinates are null, signifying the absence of a pupil, this fact is exploited in the click gesture, which has been defined in two ways. One, where the user can perform a click by staring at a point for a couple of seconds, or two, blink for an abnormal duration. The mouse control script (written in python) then interprets these co-ordinates and moves the mouse accordingly whilst operational, the script is given in Appendix B.

7. RECOMMENDATIONS FOR FUTURE WORK

7.1 Present Work

Eye tracking has become an important field of research recently. Sony is working on developing and incorporating eye tracker in their popular Playstation® platform. The technology is being developed by SensoMotoric Instruments, a Berlin-based developer, as part of its RED-oem platform and among other functions, it highlights things you're looking at, from enlarging menu options to simply tracking and responding to your interest.

.Oculus Rift is a virtual reality HCI interface which is the brainchild of Kickstarter.com and relies on heavy user gaze data to reposition the field of view to give the illusion of reality [12].

7.2 Potential Future Areas

7.2.1 Search Engine Optimization

Search engine optimization (SEO) for websites and businesses can be done effectively by employing eye tracking based information and data analytics. This potential area of research is being spear-headed by Microsoft and Google.

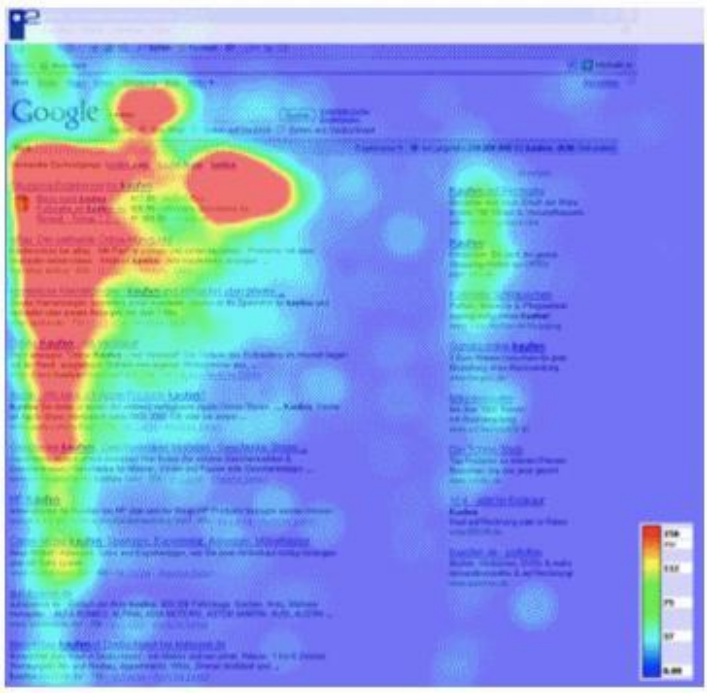

Figure 11: Reults from google of eye-tracking in Transaction.

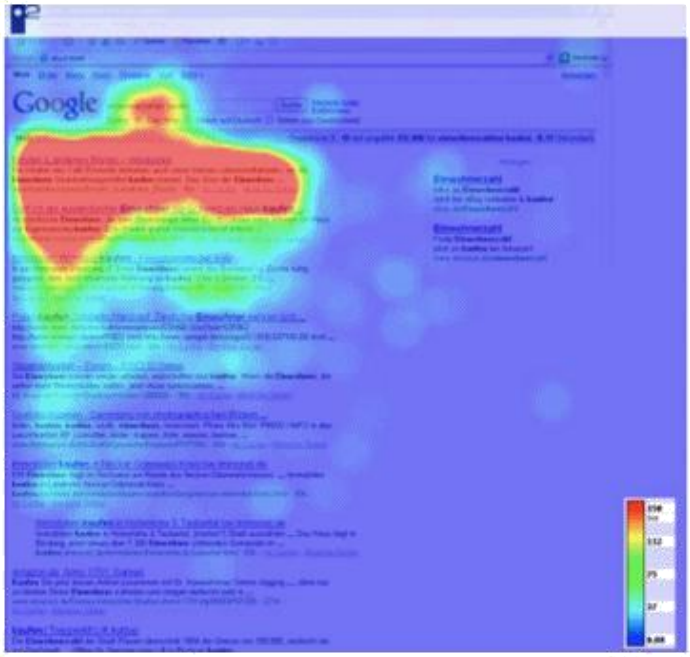

Figure 12: Results from google of eye-tracking in informational query

7.2.2 Market Research and Advertising Testing

Eye-tracker can player a major role in marketing. To understand the demands of the customer eye trackers are placed in marker and they determine for how much time the user is starting at the product and what is its reaction to the qualities or price of the of the product. This will help the manufacturers to make products that meet with the customer's requirements and in return their revenue will increase.

Figure 13: Customer gazing upon a product, points of visual attention marked by eye tracker

A commercial eye tracker scans and zeros in on which products the customer is scanning and where on those products is his gaze concentrating. An example of a dishwasher is shown in figure 18 with blue and pink markers showing where the customer's gaze concentrated and the radius of marker depicts the duration for which the customer's gaze was hooked on to that particular position. We can see the widest markers and most markers in general are concentrated towards the product label and this gives important information to manufacturers and advertisers alike that they should make the labels visually appealing, among other customer behaviour information, for example.

7.2.3 External machinery control.

Eye-tracker can be used to control external machinery such as wheel chairs or robotic arms with user's eyes. Many people are there who are unable to walk , to accommodate these people a eye-tracking wheel chair can be made. This will move the wheel in the direction where the user will see. Also some people have diseases due to which their hands do not remain still. They face difficulty while feeding themselves and hence an eye-tracking robotic arm can be made that will be controlled by the eyes of the user's eyes and will feed the user.

7.2.4 Gaze interaction and car assistant systems

Future applications also include eye gaze interaction. In this a tv can be controlled with the help of eyes. For example when you will look at the tv it will turn on or when no one is looking it will turn off. This eye-tracker system can also help people providing safety. For example If eye-tracker is installed in a car then if somehow the person sleeps or is looking away from the front it will alert the user. This will save the user from a lot of accidents that happens due to sleeping or using mobile phones while driving car.

8. CONCLUSION

As a final review, the project aims to deliver a low cost eye-tracker that will allow the user to control the mouse-cursor of a computer system. The designed system is low cost and efficient, utilizing only a camera (Microsoft Lifecam Hd-6000), software modules coded in C++ and Python Programming language. . If required, spatial field of view history can also be drawn on the world process, showing eye movements and where the user spends most of their time looking at to tweak the interface or to just retrieve spatial attention data for the purposes mentioned in the “future applications” section. Finally, we note that the project is operable in variable environmental conditions, only a few tweaks in the brightness and contrast setting need to be applied for it to maintain its robustness, this is an impressive feat for such a low cost eye tracking system.

APPENDICES

APPENDIX A– GLOSSARY

Edge Detection – A scheme whereby anchors are identified and connected along a continuous pixel line to draw the edges in an image.

Microsoft Lifecam HD-6000 – A USB web camera capable of 720p video transmission and IR light detection with use of different focal length lenses

OpenFrameworks – It is a C++ Toolkit used for coding. It supports many libraries like OpenGL, OpenCV and Quicktime, and can be easily used to build complex GUIs

Hough Transform Algorithm - Hough transform is a technique used to detect the edges of an objrct and then draw the boundary around them. It detects the edges of pupil of the eye and then draw a circle around it.

Viola Jones Algorithm –The Viola–Jones is an object-detection algorithm. It detects the face and eyes of the user from a given input.

Java Robot - The ability to control mouse pointer position, motion and clicks is not available inMATLAB. To work around this issue, you can use
the Java class java.awt.Robotwhich has this ability

Haar-Cascade - Haar-cascade is a cascade function which is trained from a lot of positive and negative images and then it detects objects from other image.

PyautoGUI - The ability to control mouse pointer position, motion and clicks is not available in Python. PyautoGUI is a module of python that allows the user to programmatically control mouse movement, clicks and keyboard.

Numpy - Numpy ia library of programming language python that supports large arrays that can be multi-dimentional and matrices.

Pupil – The Dark part of your eye which is actually responsible for eyesight

Scipy – Scipy is library of python that can be used for scientific computing and technical computing. It is free and open-source.

Face-Cascade - Face-cascade is a function of Haar-cascade that is used to detect objects in an image that resemble the features if a human face.

Eye-Cascade - Eye-cascade is a function of Haar-cascade that is used to detect .,objects in an image that resemble the features if a human eye.

APPENDIX B – MAIN CODE and SCRIPTS

The approach in which we used MATLA, following code was produced.

Code For MAtlab:

```
close all

clear all

clc

delete(instrfindall);

%deletes all the objects created when the program was run the last time, camera, webcam

x_max=1400;y_max=770;

x_move=x_max/2;

y_move=y_max/2;

jRobot = java.awt.Robot; % Initialize the JAVA robot

jRobot.mouseMove(x_max/2, y_max/2); % y_max=770;x_max 1400

vid=videoinput('winvideo',1,'YUY2_640x480');

%gets video input from the laptop webcam, 1 is for internal webcam, next is resolution
set(vid, 'FramesPerTrigger', inf);

% sets how many frames the camera will capture in a trigger, set to infinity, gives max of
camera

set(vid, 'ReturnedColorspace', 'rgb');

%returns the captured vid in RGB form

vid.FrameGrabInterval = 3;

% grabs the 1st frame, then skips next 3, grabs the next one
```

```
start(vid);

EyeDetector  = vision.CascadeObjectDetector('EyePairSmall');

% built in viola jones algo

%creates detector for eyepair

faceDetector = vision.CascadeObjectDetector();

%built in viola jones algo, by default will detect face


while(vid.FramesAcquired<=200)

% only acquires 200 frames

   temp=getsnapshot(vid);

  %%%%%%% get data frame wise

   flip_temp= flip(temp,2);

%flips the captured image to align it with real

%     temp_gray=rgb2gray(flip_temp);  %%%%%%%%%% convert rgb image to
gray scale

   face_box = step(faceDetector,temp);

%detects the face using the faceDetector, by convolving image and viola jones, the
result will
be area of face and coordinates

%     temp = insertShape(temp, 'Rectangle', face_box);


 subplot(2,2,1);

   imshow(temp);

         % displays frame on 1st subplot

      if(length(face_box)>0)

         biggest_face=1;
         %takes the first face as biggest face
```

```
for i=1:rank(face_box)
   %find the biggest face

   if face_box(i,3)>face_box(biggest_face,3)
      biggest_face=i;

   end
  % if the there is more face, it'll select the biggest one, (i,3) gives area

end

faceImage = imcrop(temp,face_box(biggest_face,:));
  % extract the face from the image, face is found

  %%%%%%%%%% the part of detecting face is
done%%%%%%%%%%

eye_box = step(EyeDetector, faceImage);
   % locations of the eyepair using detector, viola jones on the face-image

%       faceImage = insertShape(faceImage, 'Rectangle', eye_box);

 subplot(2,2,2);

imshow(faceImage)
%prints the image in subplot, of the eyes

 if(length(eye_box)>0)

     biggest_eye=1;

      for i=1:rank(eye_box)

           %find the biggest face

       if eye_box(i,3)>eye_box(biggest_eye,3)

       biggest_eye=i;

       end

      end

          % this finds the biggest eye by first selecting the first one as the
biggest and then comparing it with the remaining ones

        eyeImage = imcrop(faceImage,eye_box(biggest_eye,:));

            % extract the face from the image
```

```
subplot(2,2,3);

imshow(eyeImage)

    % plots only the region of the eyes

single_eye_box=[eye_box( biggest_eye,1),eye_box(
biggest_eye,2),eye_box( biggest_eye,3)/3,eye_box( biggest_eye,4)];
        %resize the eyepair width in half
single_eye = imcrop(faceImage,single_eye_box(1,:));

        %extract the half eyepair from the face image


single_eye=rgb2gray(single_eye);

single_eye = imadjust(single_eye);
        %adjusts the contrast of the single eye image bcz otherwise the
are too zoomed in and would be blown out
subplot(2,2,4);

imshow(single_eye)
  %plots the single eye in 4th plot

                r = single_eye_box(1,4)/4;

        % gives the center point of the single eye image

[centers, radii, metric] = imfindcircles(single_eye, [floor(r-r/4)
floor(r+r/2)],
'ObjectPolarity','dark', 'Sensitivity', 0.95);

        % Hough Transform, finds dark circles in the eye image, i-e
pupils.....floor rounds off to closest integer
[M,I] = sort(radii, 'descend');

        % sorts the circles in descending order, M holds values, I holds
the index of array

 eyesPositions = centers;
hold on;



        % draws a circle around pupil of color blue 'b'
viscircles(centers, radii,'EdgeColor','b');
```

```
            hold off
%               pupil_x=centers(1);
          if ~isempty(centers)
            pupil_x=centers(1);
           title([num2str( pupil_x) num2str(size(single_eye))]);
           if(pupil_x<200)
               x_move= x_move-150
           end
           [c r1]= size(single_eye);
          if(pupil_x>(r1-200))
               x_move= x_move+150
           end
          end
%              disL=abs(0-pupil_x);    %distance from left edge to center point
%              disL=abs(0-pupil_x);    %distance from left edge to center point
%             disR=abs(eye_box(1,3)/3-pupil_x);%distance from right edge to center
point
%
%             if disR>disL+10
%                tt=1
%                 x_move= x_move+150;
%             else if disR>disL
%                tt=2
%                x_move= x_move-150;
%                else
```

```
%                    tt=3
%                   end
%                end
%             end
  if(x_move<0)
     x_move=0;
  end
  if(x_move>770)
     x_move=770;
  end
 jRobot.mouseMove(x_move, y_move); % y_max=770;x_max 1400



        end
     end
%
 pause(.0001);
hold off
                  end
                 end
 pause(.01);
           end
stop(vid)
```

Code Writter in Python:

```
import cv2, sys, signal

from threading import Thread

import numpy as np

import pyautogui

from scipy.spatial import distance

pyautogui.FAILSAFE = False

face_cascade =

cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_default.xml')

eye_cascade = cv2.CascadeClassifier('haarcascades/haarcascade_eye.xml')

mouseX=400

mouseY=400

interrupted = False

def get_eye_ball(eye,circles):

        total = [0]

        value =0

        for x in range(eye.shape[0]):
                for y in range(eye.shape[1]):

                        ind = 0

                        for i in circles[0,:]:

                                cenX= i[0]

                                cenY=i[1]

                                radius = i[2]

                                if distance.euclidean((x,y),(cenX, cenY))<
pow(radius,2):
```

```
                value +=x
                total.insert(ind,value)
                ind += 1
        smallestSum = -99999999
        smallestSumIndex = -1
        for i in range(0, ind):
            if total[i] > smallestSum:
                smallestSum = total[i]
                smallestSumIndex = i
        return circles[0][smallestSumIndex]
def get_left_eye(eyes):
        leftEye = -99999999
        leftIndex = -1
        for i in range(0,len(eyes)):
            if eyes.item(i,0) > leftEye:
                leftEye = eyes.item(i,0)
                leftIndex = i
        if leftIndex >= 0:
            return eyes[leftIndex]
def detect_eye_blink(img):
        img = cv2.flip(img, 1)
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray, 1.3, 5)
        if faces == None:
            return
```

```
        for (x, y, w, h) in faces:
                k = 1
                roi_gray = gray[y:y + h, x:x + w]
                eyes = eye_cascade.detectMultiScale(roi_gray, 1.3, 5)
                if eyes==None:
                        return
                for (ex, ey, ew, eh) in eyes:
                        k = 0
                if k == 1:
                        pyautogui.click(button='right')
                        print "Blink"
                if k == 0:
                        print "No Blink"

def move_mouse():
        global interrupted
        while True:
                pyautogui.moveTo(mouseX, mouseY,0.5)

        if interrupted==True:
                        print "Let's quit"
                        interrupted=False
                        exit()

def main():
```

```
        if face_cascade.empty() or eye_cascade.empty():
                raise(Exception("Check Face/Eye Cascade Folder Path or
Name"))
        camID=1                                 #0 for webcam, 1 for external
        capture = cv2.VideoCapture(camID)
        diffX= diffY= lastX= lastY=  lastEyeCen=0
        global mouseX
        global mouseY
        lx=ly=lw=lh=0
        while True:
                if capture.isOpened():
                        ret, frame = capture.read()
                else:
                        raise(Exception("failed to open camera"))
                if frame is None:
                        continue
                detect_eye_blink(frame)
                gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
                faces =
face_cascade.detectMultiScale(gray,scaleFactor=1.1,minNeighbors=2,minSize=(150,
150),flags=0|cv2.CASCADE_SCALE_IMAGE)
                roi_eyes = None

        for (x, y, w, h) in faces:
                        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 200), 1)
```

```
                roi_gray = gray[y:y+h, x:x+w]

                if roi_gray == None:

                        continue

                roi_eyes = frame[y:y+h, x:x+w]
                eyes =
eye_cascade.detectMultiScale(roi_gray,scaleFactor=1.1,minNeighbors=2,minSize=(3
0, 30),flags=0|cv2.CASCADE_SCALE_IMAGE)

                if len(eyes)>2:

                        continue

                leftEye = get_left_eye(eyes)

                if leftEye == None:

                        continue

                lx =  leftEye.item(0)

                ly =  leftEye.item(1)

                lw =  leftEye.item(2)

                lh =  leftEye.item(3)

        if lx == None:

                continue

        sumX= sumY= count=0

        if roi_eyes == None:

                continue

        cv2.rectangle(roi_eyes,(lx,lw),(lx+lw,ly+lh),(0,255,0),2)

        leftEye = roi_eyes[ly:ly+lh, lx:lx+lw]

        if leftEye== None:

                continue
```

```
blur = cv2.GaussianBlur(leftEye,(9,9),0)
        if blur==None:
            continue

grayEye = cv2.cvtColor(leftEye, cv2.COLOR_BGR2GRAY)
        circles =
cv2.HoughCircles(grayEye,cv2.HOUGH_GRADIENT,1,grayEye.shape[1]/8,
param1=2,param2=10,minRadius=13,maxRadius=2)
        if circles is None:

            continue
        circles = np.uint16(np.around(circles))
        eyeBall = get_eye_ball(leftEye,circles)
        if count < 5:
            sumX+=eyeBall[0]
            sumY+=eyeBall[1]
            count+=1

        if count == 5:
            sumX/=count
            sumY/=count
            count=0
        if sumX > (lx+lw)/8:
            cv2.circle(leftEye,(sumX,sumY),eyeBall[2],(0,0,255),2)
            cv2.circle(leftEye,(sumX,sumY),2,(0,0,255),2)
        diffX = sumX-lastX
```

```
diffY = sumY-lastY

if -5 < diffX < 5 and -5 < diffY < 5:

        print "Eyes static", diffX, diffY, sumX

        diffX=0

        diffY=0

        lastX=diffX+sumX

        lastY=diffY+sumY

elif -5 < diffX < 5 and (diffY> 5 or diffY < -5):


        print "Eyes Vertically moving", diffX, diffY, sumX
        (x,y)=pyautogui.position()

        mouseX= x + diffX*2
        mouseY= y + diffY*10

        print mouseX, mouseY

elif -5 < diffY < 5 and (diffX > 5 or diffX < -5):

        print "Eyes horizontally moving", diffX, diffY, sumX

        (x,y)=pyautogui.position()

        mouseX= x + diffX*10

        mouseY= y + diffY*2

        print mouseX, mouseY


else:
        print "Eyes moving", diffX, diffY, sumX

        (x,y)=pyautogui.position()

        mouseX= x + diffX*2

        mouseY= y + diffX*2

        print mouseX, mouseY
```

```
                lastX=sumX
                lastY=sumY
                cv2.imshow('Video', frame)
                if cv2.waitKey(1) & 0xFF == ord('q'):
                        break
        capture.release()
        cv2.destroyAllWindows()
def signal_handler(signal, frame):
        global interrupted
        interrupted = True

if __name__ == '__main__':
        signal.signal(signal.SIGINT,signal_handler)
        Thread(target = move_mouse).start()
        main()
```

APPENDIX C – SETTING UP THE EQUIPMENT

To operate the equipment successfully and reliably, it must be setup in the following manner:

(Please note that the following instructions are noted down for Ubuntu 13.10).

1. Install the required dependencies via apt-get (numpy, sci-py, numexpr, openexr, opencv, matplotlib)

2. Have the program source code and the scripts in separate directories stored on the user's PC.

3. Connect the world camera (mounted on the robotic arm) to the USB port of the subject's laptop.

4. Connect the eye camera (mounted on the glass frames) to the USB port of the subject's laptop.

5. Wait 1 minute before launching the eye tracking application as camera setup takes time in Ubuntu. Camera status can be checked by typing "lsusb" in terminal.

6. Launch the terminal.

7. Change directory to the tracking app, as an example, the location looks like this "cd pupil/pupil_src/capture/".

8. Launch the code by command Python Projectname.py

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] C. Morimoto and M. Mimica. "Eye gaze tracking techniques for interactive applications." Computer Vision and Image Understanding 2005.

[2] Z. Zhu and Q. Ji. "Robust real-time eye detection and tracking under variable lighting conditions and facial orientations." http://www.ecse.rpi.edu/~cvrl/zhiwei/html/papers/cviueyetracking.pdf (2012)

[3] M. Betke, J. Gips and P. Fleming. "The camera mouse: Visual tracking of body features to provide computer access for people with severe disabliites." IEEE Transactions on Neural Systems and Rehabilitation Engineering, 10:1, pages 1-10, March 2013. http://www.cs.bu.edu/fac/betke/papers/betke-gips-fleming-nsre02.pdf

[4] M. Chau and M. Betke. "Real time eye tracking and blink detection with USB cameras. "Boston University Computer Science Technical Report No. 2005-12. http://www.cs.bu.edu/techreports/pdf/2005-012-blink-detection.pdf

[5] Heiko Drewes and Albrecht Schmidt. "Interacting With Computers Using Gaze Gestures" http://link.springer.com/chapter/10.1007/978-3-540-74800-7_43 *(2009)*

[6] Heinnsmann, J. and Zelinski, A. "3-D facial pose and gaze point estimation using a robust real-time tracking paradigm" (2013) http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=670939&url=http%3A%2F%2Fieeexplor e.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D670939

[7] Morency, L. Quattoni, A. Darell, T. "Latent-Dynamic Discriminative Models for Continuous Gesture recognition" http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=4270324&url=http%3A%2F%2Fieeexplo re.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D4270324

[8] Artifical Intelligence for Human Computing: ICMI 2006 and IJCAI 2007 International Workshops Lecture Notes edited by Thomas S. Huang, Anton Nijholt, Maja Pantic, 2006

[9] Issa Ashwash, Willie Hu & Garrett Marcot, "Eye Gesture Recognition" (2009) http://www.cs.princeton.edu/courses/archive/fall08/cos436/FinalReports/Eye_Gesture_Recogniti on.pdf

[10] Eye Gesture Controlled Wheel Chair – Final Year Project 2013 – Military College of Signals (NUST)

[11] OpenCV http://opencv.org/

[12] OpenFrameworks http://www.openframeworks.cc/

[13] The Eye Writer Project http://www.eyewriter.org (2009-Present)

[14] Drewes, Heiko. "*Eye gaze tracking for human computer interaction*." Diss. lmu, 2010.

[15] Schmidt, Jochsen, Vogt and Nieman *"Calibration–free hand–eye calibration: a structure–from–motion approach." Pattern Recognition. Springer Berlin Heidelberg, 2005. 67-74*.) (2005)

[16] PUPIL: constructing the space of visual attention. Diss. Massachusetts Institute of Technology, 2012.) Kassner, Phillip and Patera (2012)

[17] IEEE paper on "Differences in the infrared bright pupil response of human eyes" by Karlene Nguyen, Cindy Wagner, David Koons, Myron Flickner (2009)

[18] IEEE Paper on "Performance of input devices in FPS target acquisition" by Poika Isokoski, Benoit Martin

[19] "The Mind's Eye: Cognitive and Applied Aspects of Eye Movement Research" by Charlie Ware and Henry Mikaelian (1987)

[20] "An Adaptive Algorithm for Precise Pupil Boundary Detection" by Chihan Topel and Cuneyt Akinlar (2012)

[21] "Eye gaze Patterns while Searching vs. Browsing a Web site" by Sav Shrestha & Kelsi Lenz