DESIGN OF AN EYE GESTURE CONTROL SYSTEM

By

NC Hasnain Raza | NC Shaheer Cheema | NC Syed Ali Shehryar | NC Sheharyar Naseer

Submitted to the Faculty of Electrical Engineering, Military College of Signals, National University of Sciences and Technology, Islamabad in partial fulfillment for the requirements of a B.E Degree in Telecommunications Engineering

JUNE 2014

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 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 an Arduino Due Board along with a modified Microsoft LifeCam HD-6000 is used. The project is designed such that it can be 'hooked up' to any compatible software or machinery – for example it is able to control the mouse cursor on a computer or operate a Robotic Arm, and can be extended by enthusiasts to operate hospital on-call equipment 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.

ACKNOWLEDGMENTS

We would like to offer our thanks to Lec. Moiz Ahmed Pirkani 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

Due to the tremendous progress in computer technology in the last decades, the capabilities of computers increased enormously and working with a computer became a normal activity for nearly everybody. With all the possibilities a computer can offer, humans and their interaction with computers are now a limiting factor. This gave rise to a lot of research in the field of HCI (human computer interaction) aiming to make interaction easier, more intuitive, and more efficient. Interaction with computers is not limited to keyboards and printers anymore. Different kinds of pointing devices, touch-sensitive surfaces, high-resolution displays, microphones, and speakers are normal devices for computer interaction nowadays. There are new modalities for computer interaction like speech interaction, input by gestures or by tangible objects with sensors, the industry is also noting a trend in visual reality goggles that challenge that bounds of HCI. Such systems typically use eye gaze as the sole input, but outside the field of accessibility eye gaze can be combined with any other input modality. Therefore, eye gaze could serve as an interaction method beyond the field of accessibility [2], [3], [11].

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 while employing infra-red principles for detection. 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, convert and forward the necessary serial port control outputs to a microcontroller to be used with robotic 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) [13],[15] or those paralyzed by injury, are unable to use their computers for basic tasks such as sending/receiving messages, browsing the internet or watching their favourite Netflix show. 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 moving commercially available robotic machinery such as the robotic arm 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 fulfil the requirements for our Bachelor's degree at the National University of Sciences & Technology, Military College of Signals. It belongs to the field of HCI (Human Computer Interaction) 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 and modules of the system.

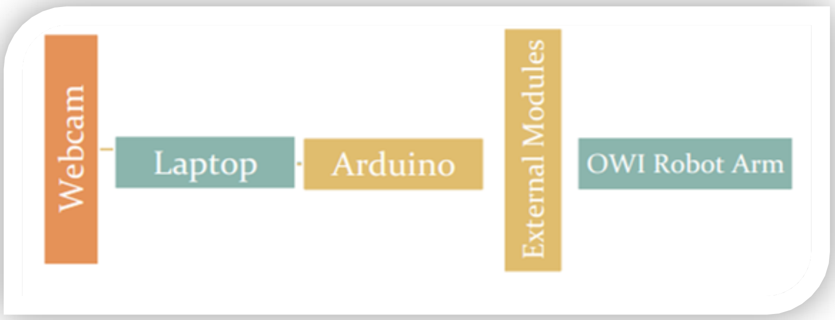


Figure 1: Block Overview of the Designed System

The project prototype 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 the use of the eyes as a graphical drawing tool [14] and a means to communicate with robots so that someone with a disability like say Amyotrophic Lateral Sclerosis can use to communicate with hospital staff or "helper" robots.

On software-end, a program based on OpenCV [11] and Pupil Frameworks [16] has been developed which consists of both front-end and back-end (low-level detection of pupil) modules coded in python that will, with the help of infra-red illumination of the eyes, track the pupil position of the eyes, and use this information to perform some of the functions mentioned above.

Also, an Arduino Due board will be connected to the computer's USB port (serial over USB), where the output generated by the PC will be sent, the Arduino would interpret the data and produce outputs accordingly on its Digital Pins, where a connected robot like an arm or hospital machinery (bed switches, nurse alarms etc.) can be interfaced and driven essentially via eye control. Controlling simple robots via pupil movements is the proof of concept we're trying to present as well as the underlying motto of our final year project.

Figure 2 shows general flow of the project's design. To summarize:

1. Pupil movements are detected and recorded by the modified camera at a rate of 24 frames per second.
2. The low-level algorithm called the Structure from Motion (structure from motion algorithm) is based on research done by Patera and Kassner in MIT as well as by Drewes & Heiko who have concluded that the **SFM**

algorithm presents by far, one of the most efficient means of pupil tracking.

3. The software we have programmed is based on OpenCV (standard framework used for computer vision as well as human computer interaction) and Pupil® framework. All in all, this program compares frame by frame the position of the pupil and forms the basis for efficient tracking.
4. The pupil positions are streamed by creating a local server on port 5000 via universal socket connections supported by the python programming language.
5. Output is shown on the computer via mouse based-movements and click gestures.
6. Furthermore, the screen is divided into imaginary quadrants. Each quadrant corresponds to a different serial output. For example, if the output in quadrant 1 is 1000, the output in quadrant 2 could be 1001. This of course is a lay men's example just for the sake of explanation. The actual implementation is a lot more complex and discussed in detail in the thesis.
7. These serial port outputs are exploited by the Arduino due microcontroller and used to produce three-dimensional robotic arm motion.

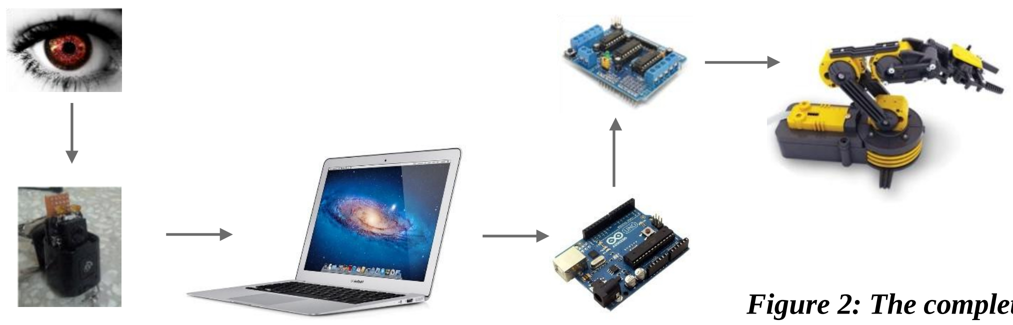


Figure 2: The complete system overview

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 and modified 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. The output modules include an Arduino due microcontroller that acts as an interface between digital outputs from the computer and the intelligible outputs to be transmitted to the robotic arm.

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. 2x SMD Infra-red LEDs at 840nm emission with peak output power of 0.2mw/sr
3. A pair of glass frames
4. A laptop
5. An Arduino Due board (ARM Cortex M3)
6. The eye-tracking software
7. Transistors (to form an H-bridge)
8. A high current battery
9. 5x DC motors
10. A motor driver board
11. Steel frames and 3D Printed Camera Mount (Our own proprietary design).
12. Connecting wires

Items 1 to 6 comprise of the main eye tracking product, the rest of the components were used to build the external robotic modules (a robotic arm that is operated with eye gestures).

3. LITERATURE REVIEW/BACKGROUND STUDY

3.1 Literature Review

With the invention of the computer in the middle of the last century there was also the need of an interface for users. In the beginning experts used teletype to interface with the computer. Due to the tremendous progress in computer technology in the last decades, the capabilities of computers increased enormously and working with a computer became a normal activity for nearly everybody. With all the possibilities a computer can offer, humans and their interaction with computers are now a limiting factor. This gave rise to a lot of research in the field of HCI (human computer interaction) aiming to make interaction easier, more intuitive, and more efficient. Interaction with computers is not limited to keyboards and printers anymore. Different kinds of pointing devices, touch-sensitive surfaces, high-resolution displays, microphones, and speakers are normal devices for computer interaction nowadays. There are new modalities for computer interaction like speech interaction, input by gestures or by tangible objects with sensors. A further input modality is eye gaze which nowadays finds its application in accessibility systems. Such systems typically use eye gaze as the sole input, but outside the field of accessibility eye gaze can be combined with any other input modality.

A paper published by Drewes and Heiko [15] discusses basic methods and approach to eye tracking. It in itself presents by far the most definitive introductory document for students and researchers alike, wanting to experiment in the field of gaze gesture based human computer interaction. Not only does it provide a historical overview, a general introduction but it critically analyses all the algorithms developed to date for eye tracking and which ones should be chosen for effective eye tracking. Amongst the approaches discussed are means of using eye gaze as pointing device in combination with a touch sensor for multimodal input and presents a method using a touch sensitive mouse. The second approach examines people's ability to perform gestures with the eyes for computer input and the separation of gaze gestures from natural eye movements. The third approach deals with the information inherent in the movement of the eyes and its application to assist the user.

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 [16] in using structure from motion algorithm for human computer interaction. This was exploited by Kassner, Moritz Philipp, and William Rhoades Patera [17] 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 [9] 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 [2] 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 human eye uses a two lens system in a fluid called the vitreous humour to project incoming rays of light from the world onto the retinal surface that is located at the back of the eye. Figure 3 gives the structure of the human eye with the various aspects labelled. On the retinal surface there is an area that is densely packed with cones in proportion to rods. Once past the lens, the rays of light travel through the vitreous humour and are cast on the retinal surface. The fovea is measured in angular diameter of between 0.3 degrees and 2 degrees, or 1/4000th of the retinal surface area [8].This area is called the fovea, and appears as a small yellow spot on the retinal surface. The

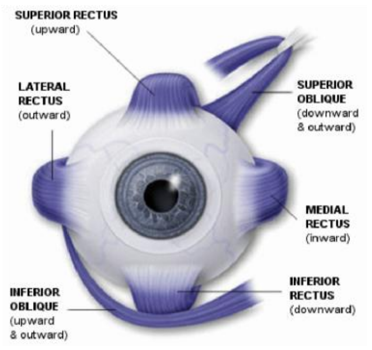

SUPERIOR RECTUS (upward)
LATERAL RECTUS (outward)
SUPERIOR OBLIQUE (downward & outward)
MEDIAL RECTUS (inward)
INFERIOR RECTUS (downward)
INFERIOR OBLIQUE (upward & outward)

Figure 3: Structure of the Eye

Retinal surface contains two types of photo receptors, rods and cones [16]. 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 [18].

The first optical element is the Cornea, to be precise, a thin layer of tear fluid that covers the curved corneal surface. The surrounding area is populated by rods, densely packed around the fovea. One function of the aperture is to control the amount of light that passes into the lens, or camera. This aperture can change in size, growing larger – dilation in low light conditions to allow more light, or smaller – constricting – to allow less light.

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. But all this response is done in face of a visible light stimulus. The project utilizes infra-red illumination, which is invisible to the naked eye, hence the pupil doesn’t expand or contract, neither does the lens focus this IR light on the nerve endings at the back of the eye. Without this response from the eye, is IR illumination safe. A research paper [18] highlights the health and safety risks associated with IR illumination of the eye, and quotes 5mW as the safe upper limit. The SMD IR LEDs we employ use 0.2mW of power, and even that falls off rapidly with distance. Hence the project is safe to operate and the user need not be concerned in light of health and safety.

3.2.2 Field of Vision

The human monocular field of vision, without eye movement, is one hundred and sixty degrees in width and one hundred and seventy five degrees in height. While this may seem like an incredibly large field of vision, this field of vision is not all sharp and in colour. The visual angle foveal vision is only approximately one degree. Figure 4 shows the resolution capabilities of the human eye as a function of foveal cone density. As a practical example, the area that a human eye resolves in colour and high resolution is approximately equivalent to the area of one's thumbnail held out at arm's length. Due to this limitation we have to rotate our eyes in their sockets, positioning the eye such that the area of interest in the world is projected onto the fovea [17]. This movement called a saccade and is usually very fast, up to seven hundred degrees per second and typically lasting for thirty milliseconds. The saccade is a major motif of visual movements.

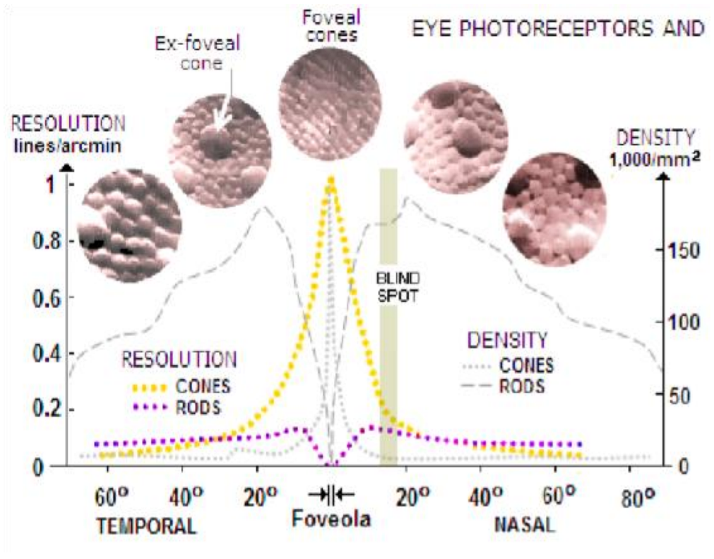


Figure 4: Photo Receptors in the eye and visual acuity

Not only is the retinal area of high acuity small, but the cones are also relatively slow. The step response time for cones is approximately twenty milliseconds. For an image to fully resolve with all its high frequency details, its projection on the fovea need to be motionless. Any movement faster than three degrees per second would result in a blurry image. During fast movements like a saccade the image would be blurred. However, the visual system is blocked during saccades and therefore the brain does not receive this useless information. During a saccade we are effectively blind. This can be easily demonstrated by attempting to look in a mirror and observe your own eyes moving.

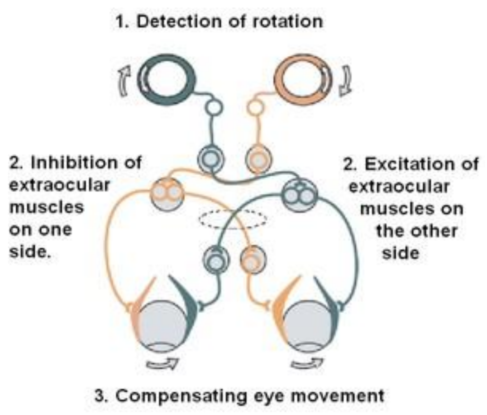

1. Detection of rotation
2. Inhibition of extraocular muscles on one side.
2. Excitation of extraocular muscles on the other side
3. Compensating eye movement

Figure 5: Eye movement compensation

Hence, saccades are undesirable for a tool that uses the eye for accurate and precise pointer or machinery control. To counter saccades, we add dwell time in the click activation of a mouse. This negates any random eye movements and only passes the ones that are desirable for processing.

3.2.3 Eye Masculinization and Stabilization

For successful resolution of a detail, one's gaze needs to rest on the area of interest, this is called fixation another motif of eye movements. Fixations typically last for three hundred milliseconds. During a fixation two powerful image stabilization mechanisms keep the projection in place. Even if one object of interest moves relative to the viewer, the image stays in place.

This stabilization is achieved by the vestibuloocular reflex (VOR) and the optiokinetic reflex (OKR). Both reflexes actuate a three pairs of muscles that rotate the eye around its three axis: the lateral rectus, the medial rectus, the inferior rectus, the superior rectus, the inferior oblique, and the superior oblique. When the muscles contract and their counterpart relaxes accordingly, the resulting torque moves the eye in almost pure rotation. [18]

The eye can be rotated voluntarily to yaw and shift, allowing any part of the field of view to be projected onto the fovea. Movement around the optical axis, rolling, cannot be triggered voluntarily and is not very noticeable due to the rotational symmetry of the eye, but it is nonetheless frequently used by the compensatory reflexes.

VOR is a reflex that compensates for rotation and translation of head movements. These movements are sensed by the vestibular apparatus in the inner ear. Here the inertia of a fluid is sensed by small hairs in the inner ear. Rotational and translational change is then passed on to a short and very fast neuron network called the three neuron arc that stimulates the eye muscles to compensate for the movement.

OKR is triggered by the assessment of change in the foveal image. As the projected image starts to drift the angle and velocity of this change is measured and the eye motor muscles are stimulated to compensate for this change. The processes used to evaluate angle and magnitude of change is analogous to a process in computer vision called optical flow.

3.2.4 Direction of Visual Attention

Beyond these two reflexes, VOR and OKR, the visual system stimulates the eye muscles to rotate in order to inspect areas of interest. The visual cognitive pathways that control and decide what area in the field of vision to evaluate in greater detail is a subject of ongoing inquiry. Current research suggest that two schemes can be used to describe this behaviour. The first is typically called “bottom up control,” where salient features determine what is to be attended to. The second is called “top down control,” where attention is focused by a predetermined cognitive task [16].

In the bottom up control schemes, the area in our field of vision that is resolved in foveal vision is not only uncontrollable by conscious thought, but it is also completely opaque to it. Salient features of a scene prompt our visual system to saccade the gaze point to this area of interest and fixate on it. An example for this is a sudden movement in the peripheral vision or a contrasting detail like a blossom on a background of green leaves.

Top down control is controlled by high level motivations, like a verbal cue that lets our visual system shift foveal vision towards areas that we deem as potentially informative for the given visual task. This could for example be a search task: “find the red bottle”, or a motor task like opening a drawer. This control scheme is very interesting, but at the same time opaque to us as it implicates the combined motivations of the viewer, their prior experience, training, and a myriad of other influences. While neither the neurological implementation nor more detailed rules of interaction are understood, it appears that both schemes are weighted into the decision what to fixate upon next [16], [18], [19].

Having considered the workings of human-eye and how and why can gaze gestures be used as modes of Human Computer Interaction, it should be noted that eye tracking is not a new field. The emergence of Eye tracker is due to gaze motion research, starting from 1879 [19], when Louis Emile Javal observed that reading does not involve a smooth sweeping of the eyes along the text, as previously assumed, but a series of short stops, called fixations and quick saccades. We visually see our environment only through fixations. Brain actually combines visual images that we receive through fixations. This discovery raised important questions about reading, which were explored during the 1900s.

1930s – Eye tracking research characterize by a more applied research focus, especially for experimental psychology.

1950s — Alfred L. Yarbus wrote about the relation between fixations and interest in an object. Eye movements reflect the human thought processes; so the respondents' thought may be followed to some extent from records of eye movements (the thought accompanying the examination of the particular object). It is easy to determine from these records which elements attract the observer's eye and, consequently, his thought, in what order, and how often.

1970s — Eye tracking expanded rapidly marked by significant improvements in eye movement recording systems facilitating increasingly accurate and easily obtained measurements.

1980s — M. Just and P. Carpenter formulated the influential Strong eye-mind Hypothesis, the hypothesis that here is no appreciable lag between what is fixated and what is processed. The hypothesis is very often today taken for granted by beginning eye tracker researchers.

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

On the hardware side, the Microsoft Lifecam HD-6000 has been disassembled, it has two blue LEDs that light whilst operation, they are an interference to our infra-red light source so these have been removed and will be replaced by SMD (surface mounted devices) LEDs that we ordered. For the time being we have attached a Perf-Board containing low power LEDs with the camera PCB shown in figure 6.

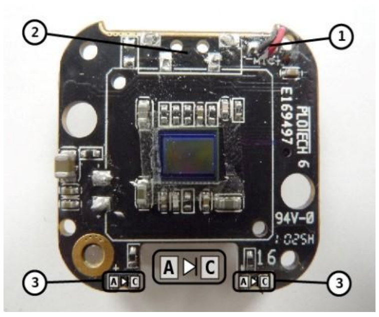
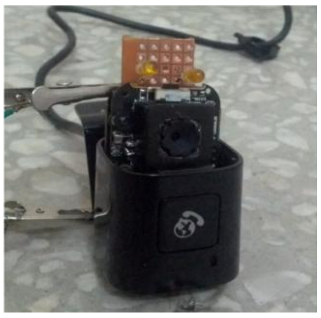


Figure 6: The camera PCB. 1 - Microphone, 2-switch, 3-LEDs

The perf-board modifications are shown in figure 7. The extraneous switches have also been cut off, the lens has been unscrewed and the thin IR filter has been carefully



removed from the lens to allow IR light to pass through into the camera image sensor,

Figure 7: Camera with IR infra-red LEDs on perf-board

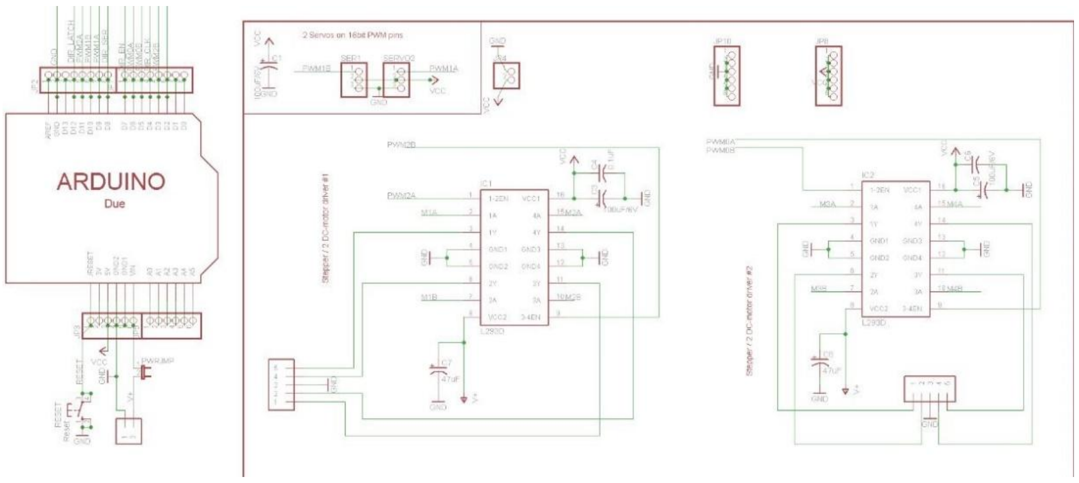
to block visible light, we have added two sheets of cut film negatives as make shift visible light filters where the IR filter used to be and secured it with slight glue. The camera autofocus was also a problem, since we do not want it focusing on random facial features, the focus should ideally be set by physically (by changing the distance between the two lenses, screwing or unscrewing them). So during re-assembly, the metal contacts that give its motor power were rotated 180 degrees so that the motor contacts never touch the power supply, hindering the autofocus from running.

Furthermore, two micro SMD infra-red LEDs Tanta mounting to a total of 0.4mW/Sr of IR energy have been soldered directly on the circuit to serve its purpose as means for redundancy as well to ensure fail-safe tracking of the pupil. To overcome health issues, all the IR LEDs chosen certify ergonomically and are safe for long term use.

An alarming issue that surfaced was the over-heating of camera circuitry from extraneous usage. Since we wanted the camera to be used for pupil tracking consistently, often for longer durations, a custom-made heat sink was ordered through OMNI Engineering INC, specific to the camera circuit dimensions and which was MIL-STD-202 as well as RoHS- Reach compliant. Instead of using glue to attach the heat sink, Thermal Conducting Paste was used. Both these measures helped in dramatically reducing the circuit’s basal temperature and allowed worry-free use for longer durations [11], [16].

4.1.2 Motor Driver

Initially we were using a single servo motor with our Arduino, to check whether the serial library of the Arduino works with enough accuracy so as to allow us to build a robotic arm. We experienced that connecting a servo motor directly to the Arduino is



not suitable, so we designed a motor driver that could efficiently operate multiple

Figure 8: Circuit diagram of L293D and Arduino Due

servo motors by being controlled via an Arduino. The figure below shows the schematic for the L293D motor driver based board, along with the connection for the Arduino Due board.

This ‘Motor Driver’ uses the L293D IC as its base and will act as a medium between the Arduino and the “Servo Motors” of the OWI Robotic Arm. Multiple instances of this circuit will be connected to each Servo Motor to effectively control them via our host machine.

The figure below shows the 74hc595 shift register, whose binary output of 8 bits is what actually optimizes the Arduino interface. Only 3 pins of the Arduino are used to drive the motors and hence 3 pins of the Arduino are converted to an 8 bit output. This is done via the use of serial commands, and latch on and off states. The latch is shut off, the data is written to it in the “write mode”, the latch is locked and the data is then transmitted to the output register where the serial data is represented in 8 bits. The shift register based interface and its pin configuration with figure 4 are shown in figure 9:

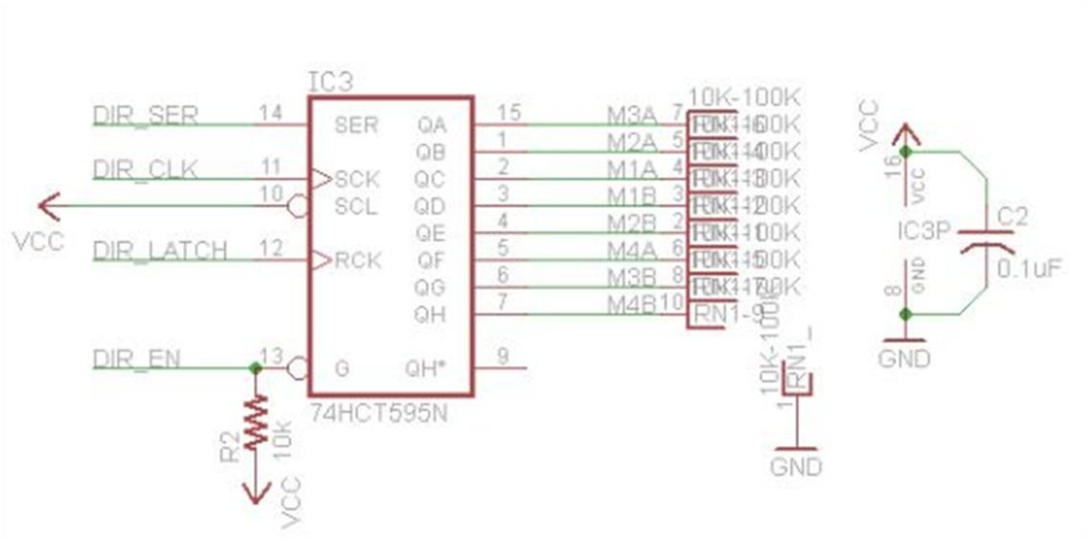


Figure 9: 74HC595 pin configuration in relation to figure 4

4.1.3 OWI Robotic Arm Edge 535

Initially we set out to create our own robotic arm for this senior design project. However, after timely consultations with the faculty at CARE University, as well as at SEECS, NUST, we were advised to use a kit- based robotic arm instead. This advice proved its worth when we found that not only was buying kit based arm cheaper, but allowed up-to five degree freedom of motion and was again made of composite materials instead of steel, which reduced its overall weight. After thorough research, OWI 535 robotic arm was chosen as the most suited candidate.

The OWI 535 is a cheap robotic arm with a lifting capacity of 100g and 5 degrees of freedom of motion. The Arm comprises of simple DC motors that are connected to worm drive gearboxes. The low torque of the “worm” rod is translated into low speed and high torque of the worm wheel inside this gearbox technique, hence the arm moves slowly, with relatively good precision but the action time suffers as shown in

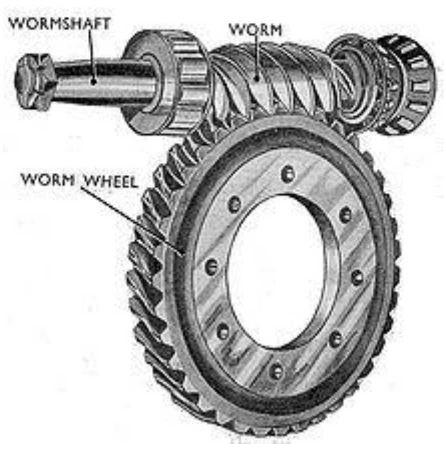


figure 10.

Hence, the arm is not meant as a real time prosthetic, just as a substitute in the critical conditions mentioned in the initial paragraphs of the report. We also see that because of this revelation, we needed the motor driver board because dc motors are inherently not meant to be accurate and precise in their movement unlike Servos that feature internal feedback mechanisms. The DC motor is an open loop system. To negate the effects of this open loop system, OWI 535 will be modified by connecting potentiometers to each arm joint and set to a specific voltage.

Figure 10: Worm drive gearbox

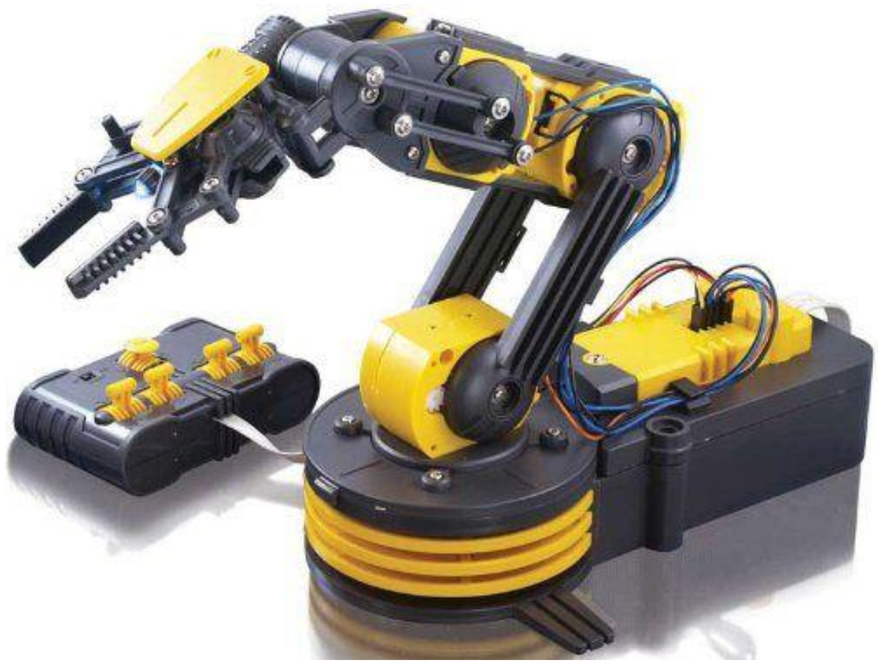


Figure 11: The OWI 535 robotic arm

The voltage forms a reference which is to be achieved by the motion of the arm. For example, if the pot is set to 3 volts, the arm will move until it achieves 3 volts. This serves as an internal feedback mechanism that effectively converts the worm drive DC motors to "Servos". The actual signal voltage subtracted from the reference signal set on the pot, forms the error signal, which the Arduino (the controller) aims to set to zero.

The Arduino sketch sample has also been written, it requires the use of the adafruit's motor shield library which is specifically written for worm drive gear boxes. It features a motor speed translated onto a scale of 0-255. The median value used mostly is 100, but this can be modified in the sketch itself. A snippet of the sketch is shown here, to give an idea of how the motor is read and interpreted, and the arm moved:

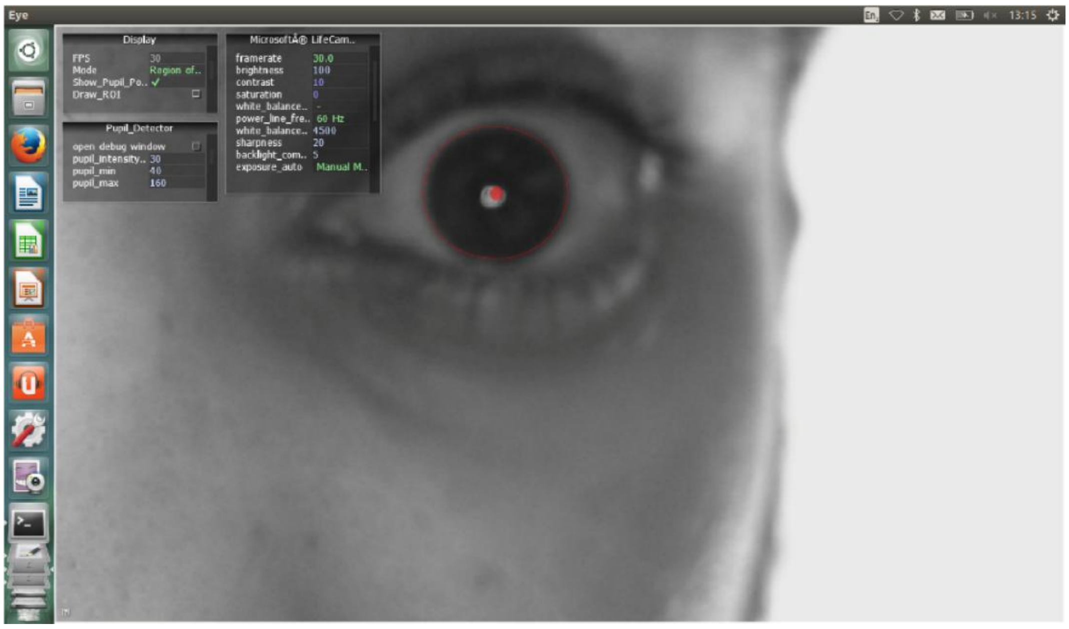
```
motor1.setSpeed(100);    // set the speed to 100/255
motor1.run(FORWARD);
motor2.setSpeed(100);    // set the speed to 100/255
motor2.run(FORWARD);
motor3.setSpeed(100);    // set the speed to 100/255
motor3.run(FORWARD);
motor4.setSpeed(100);    // set the speed to 100/255
motor4.run(FORWARD);
delay (1000);
motor1.setSpeed(0);      // turn off motors
motor2.setSpeed(0);
motor3.setSpeed(0);
motor4.setSpeed(0);

// Now read the sensors to see how they changed

val0 = analogRead(0);   // read the input pin 0
val1 = analogRead(1);   // read the input pin 1
val2 = analogRead(2);   // read the input pin 2
val3 = analogRead(3);   // read the input pin 3

Serial.print("1 = ");   // report the new readings
Serial.print(val0);
Serial.print("  2 = ");
Serial.print(val1);
Serial.print("  3 = ");
Serial.print(val2);
Serial.print("  4 = ");
Serial.println(val3);
delay(100);
```

The code snippet above assumes that the reader knows about the Serial library functions. The code snippet moves all the motors forward, and in effect, causes the entire arm to lunge forward. Function definitions are not shown because their names are self-explanatory for reasons of reading continuity, the final code is included in Appendix B.

4.2 Software

We had been working in tandem with the Open Pupil community [16] on this open source project and have made more than ten commits to their Git Repository. The framework provided on their Git repo was compatible with Webcams only. We have been successfully able to run the Pupil Framework after customizing it to work with the Microsoft HD-Life 6000 Cam that we aim to use with our project. The code also requires the addition of buttons so that the user can interact with robotic modules. Looking at a button for 2 seconds or more triggers the action associated with that button, that is, a signal is sent via the USB port (Serial over USB) to the Arduino where the custom sketch uploaded on it interprets it and drives the module.

We have so far worked on two core modules namely, Main.py and Eye.py, two computer to control system action modules namely that mouse.py and gaze_arduino.py. This is in addition to 15 other modules we have programmed for accurate pupil detection, for low-level algorithm implementation, for the front-end and for the front-end to back-end coordination. We have also successfully debugged some glitches that were present earlier in the software framework. These glitches, as well as their solutions were sent to both the OpenCV and Pupil communities and is in itself, a commendable feat by our group. The code, as it is right now is capable of pupil detection under infra-red light [15]. The Framework was initially very unresponsive. After hours of debugging, coding, optimization and calibrations, it is now fully functional. It now continuously tracks all eye movements with little or no issues. This also involved complete isolation of the world.py code and its severing from the rest of the application framework since we do not require the tracking feedback from the world camera.



Display	
FPS	30
Mode	Region of..
Show_Pupil_Po..	✓
Draw_ROI	☐

Pupil_Detector	
open debug window	☐
pupil_intensity..	30
pupil_min	40
pupil_max	160

MicrosoftÂ® LifeCam..	
framerate	30.0
brightness	100
contrast	10
saturation	0
white_balance..	-
power_line_fre..	60 Hz
white_balance..	4500
sharpness	20
backlight_com..	5
exposure_auto	Manual M..

Figure 12: Eye Tracking interface

Our steps consisted of first making sure the framework's algorithm was able to detect the pupil and give the required feedback. This involved adjusting the absolute zoom and focus of the eye camera. Once that was done, we changed the mode to Camera Tracking, to make the tracker more visible to the user and calibrate the user's eyesight mapped to what he's looking at. This calibration technique is known as "Natural Features". This procedure involves the user to look at various positions on the screen as marked as by himself. After this the Pupil Frameworks processes these inputs and calibrates itself to better track eye movements by mapping an eye gaze position to world feature. Once all of this was done, pointer control was established with the user's eye, along with a stare interface in the world cam window, enabling one to send outputs to the serial port by staring at specific points on the interface [14], [15].

A second feature, to imitate and execute screen based 'click' notion was programmed and added. This was based on the premise that once the pupil coordinates were streamed, closing the eye altogether serves as an interrupt. If this interrupt is more than 5 milliseconds in duration, it would serve as a click. A similar second delay acts as the second click and if performed simultaneously, emulates the "double-click". This would not only enable computer usage, but is integral to the movement of robotic arm via buttons as mentioned previously.

The pupil detection algorithm used in the framework works on the principle of infra-red illumination of the eye and then edge detection followed by ellipse fitting to find a suitable pupil candidate. The figure 13 below helps explain the entire process:

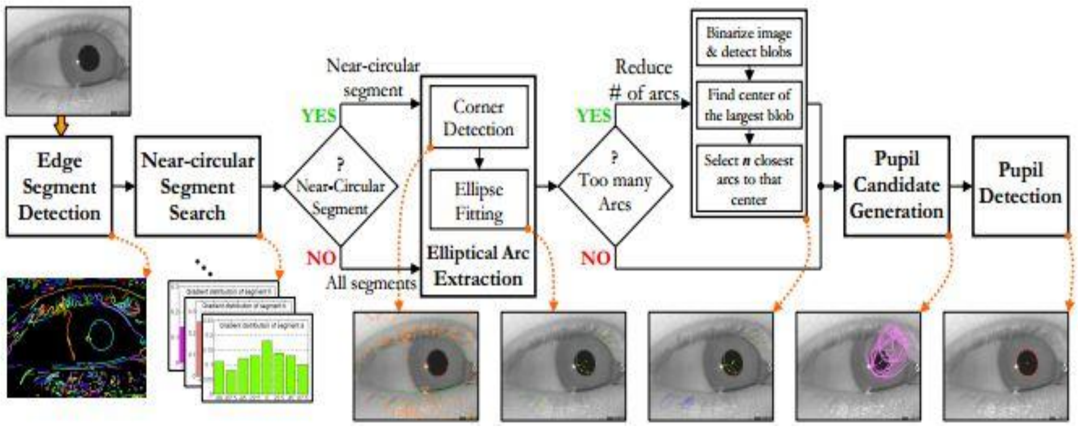

Edge Segment Detection
Near-circular Segment Search
Near-circular segment
YES
? Near-Circular Segment
NO
All segments
Corner Detection
Ellipse Fitting
Elliptical Arc Extraction
Reduce # of arcs
YES
? Too many Arcs
NO
Binarize image & detect blobs
Find center of the largest blob
Select n closest arcs to that center
Pupil Candidate Generation
Pupil Detection

Figure 13: Pupil detection process [Ref: 20]

The edge detection (ED) algorithm works by identifying points in an image called anchors, and joins these anchors using a smart routing procedure, meaning it literally draws edges in an image, these edges are connected along a continuous pixel chain. Then near circular segment search is carried out on the ED frame, which consists of analysing the gradient of all the edges in the ED frame. Circular objects have a relatively constant colour gradient compared to other shapes, hence, the edge resulting in the least varying gradient is chosen to be processed by the arc extraction method. If a circular edge is detected in the previous step, it is processed by the arc extractor, if not, then the entire image is subjected to arc extraction. The process basically tries to fit ellipses around the edges obtained from the edge segmentation procedure, when an ellipse or ellipses (continuous pixel line) is/are found, then the algorithm tends to reduce that number down to a single pupil candidate by drawing arcs around the most complete ellipse obtained to counter occlusions and draw accurate boundaries, finally, the pupil is shown as a completely detected ellipse, for complete details on the algorithm along with the mathematical studies, please refer to [20].

5. BOTTLENECKS AND ISSUES

SMD LEDs (figure 14) are not easily available in Pakistan so we had to order them online and it'll take some weeks to get here. So for the time being, we have attached a perf-board containing low power LEDs. This has put a considerable strain on our progress and has caused great inaccuracy in our results.

Figure 14: SMD LEDs

Another 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.

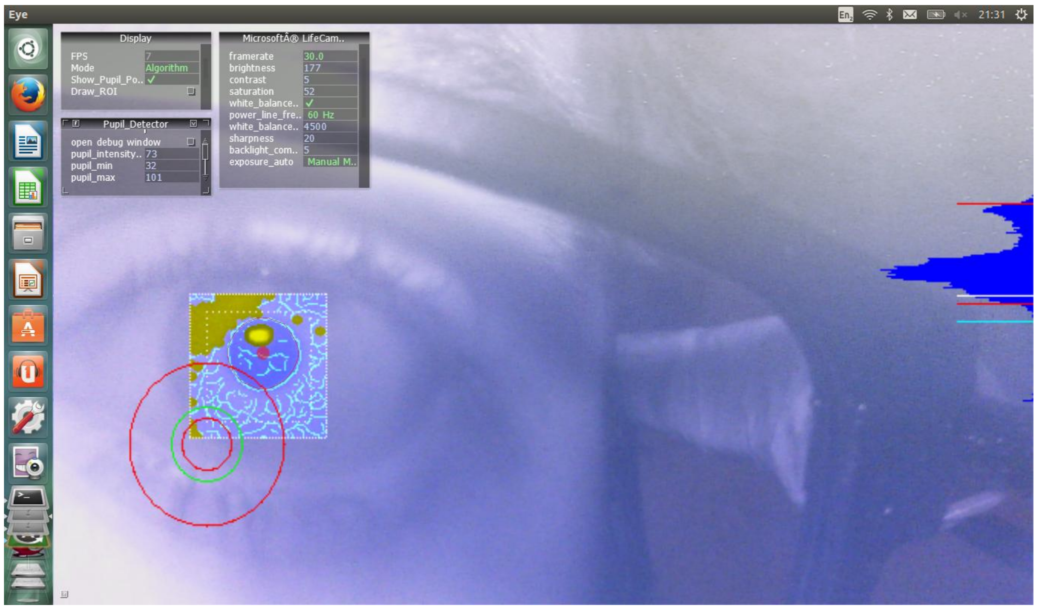
IR-Based Eye Tracking also involves removal of major part of the visible light spectrum. In order to do that we initially used the dark magnetic disk that's found inside floppy disks but it blocked considerable portion of IR Light as well. We then went with the Film Negatives that were used in Analogue Cameras. It was able to remove visible light and allow sufficient IR Light to pass through.

A 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.

Originally we were not able to perform calibration on our eye-tracking cameras, so the accuracy of our results were very limited. We overcame this hurdle by contacting the pupil community and different developers across the world.

6. PROJECT ANALYSIS & EVALUATION

We start by presenting results of the working contour detecting algorithm that works by the principle of IR illumination, whereby infra-red light illuminates the pupil, and causes the pupil to stand out as a dark circle. The pupil is detected via the algorithm explained previously (see figure 13). Figure 16 below shows the algorithm performing in our framework:



Display

FPS	7
Mode	Algorithm
Show_Pupil_Po..	✓
Draw_ROI	☐

Pupil_Detector

open debug window	☐
pupil_intensity..	73
pupil_min	32
pupil_max	101

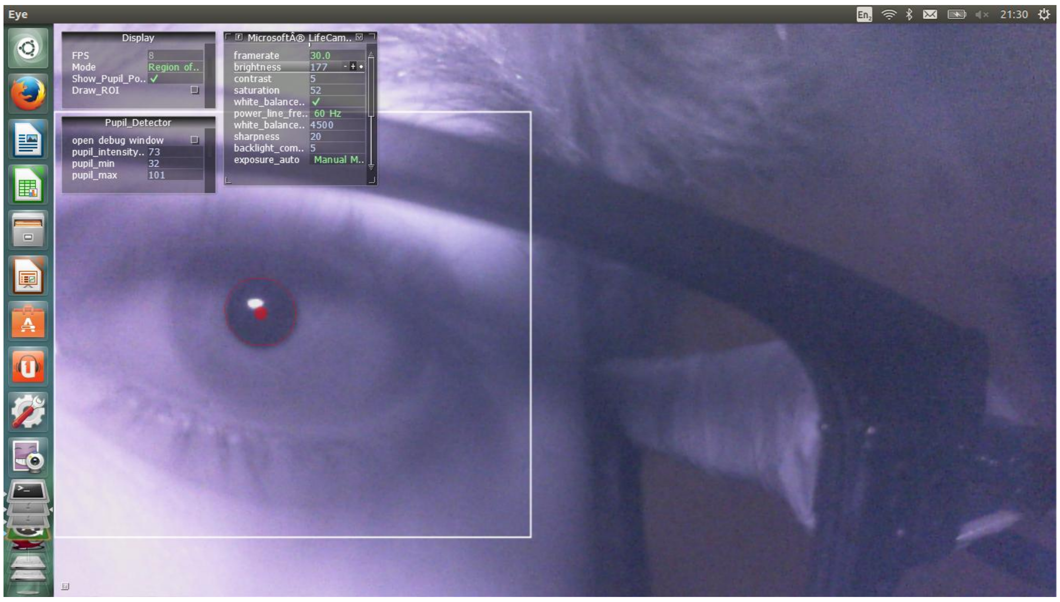
MicrosoftÂ® LifeCam..

framerate	30.0
brightness	177
contrast	5
saturation	52
white_balance..	✓
power_line_fre..	60 Hz
white_balance..	4500
sharpness	20
backlight_com..	5
exposure_auto	Manual M..

Figure 15: the algorithm performing pupil detection

Notice that the colour gradient on the right hand side of the screen has colour spikes around the drawn ROI, but along the pupil, the gradient is flat (0). Also note that the pupil min/max thresholds depicted on the bottom left hand of the screen are parameters that determine how well the pupil will be detected and against the limit thresholds. The ED algorithm demands these parameters to be set before running the entire program to perform various actions as explained in the project abstract. It should be noted however, we minimize processing time for the image and for pupil detection by drawing a region of interest around the eye so that the algorithm works only on that portion of the image, the ROI is shown on the next page.

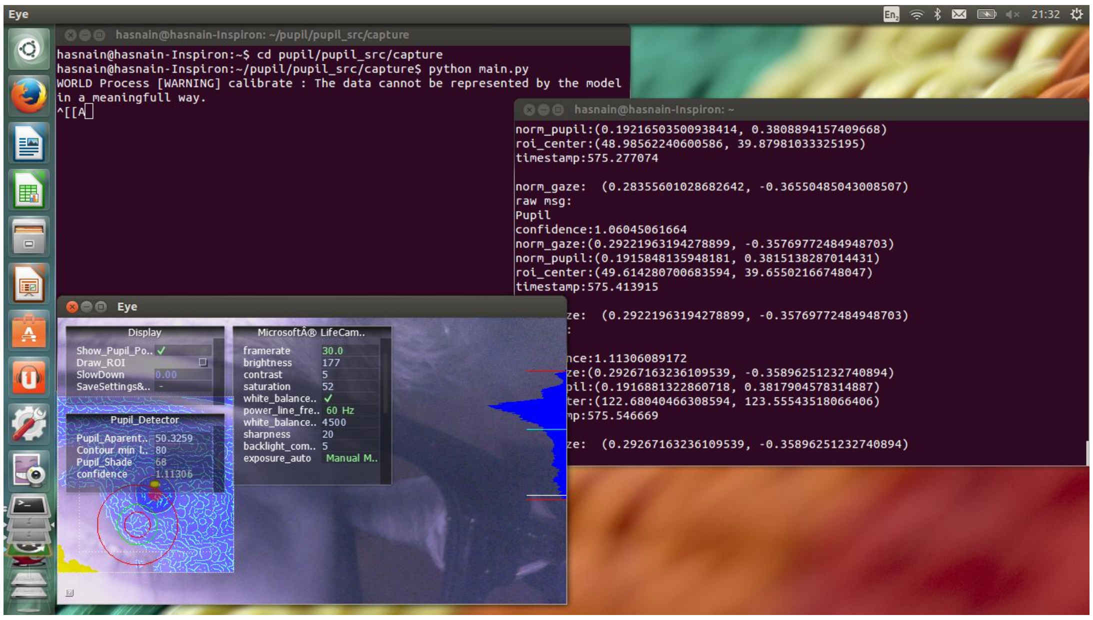
The white borders define the region of interest for the algorithm to perform on, this minimises the number of pupil candidates generated by the ellipse extractor since other portions of the image may serve to only act as noise for the image.



```
FPS                 8
Mode                Region of..
Show_Pupil_Po..     ✓
Draw_ROI

open debug window
pupil_intensity..   73
pupil_min           32
pupil_max           101

framerate           30.0
brightness          177
contrast            5
saturation          52
white_balance..     ✓
power_line_fre..    60 Hz
white_balance..     4500
sharpness           20
backlight_com..     5
exposure_auto       Manual M..
```

Figure 16: The Region of Interest

Next up, we analyse the results of the pupil detection by running a tcp server via zmq on port 5000. The framework transmits the pupil positions to this server after eye to world cam mapping has been performed by the calibration routine, as can be seen in the terminal window on the next page.


Eye
21:32
hasnain@hasnain-Inspiron: ~/pupil/pupil_src/capture
hasnain@hasnain-Inspiron: ~
Eye
Display
Pupil_Detector
MicrosoftÂ® LifeCam..

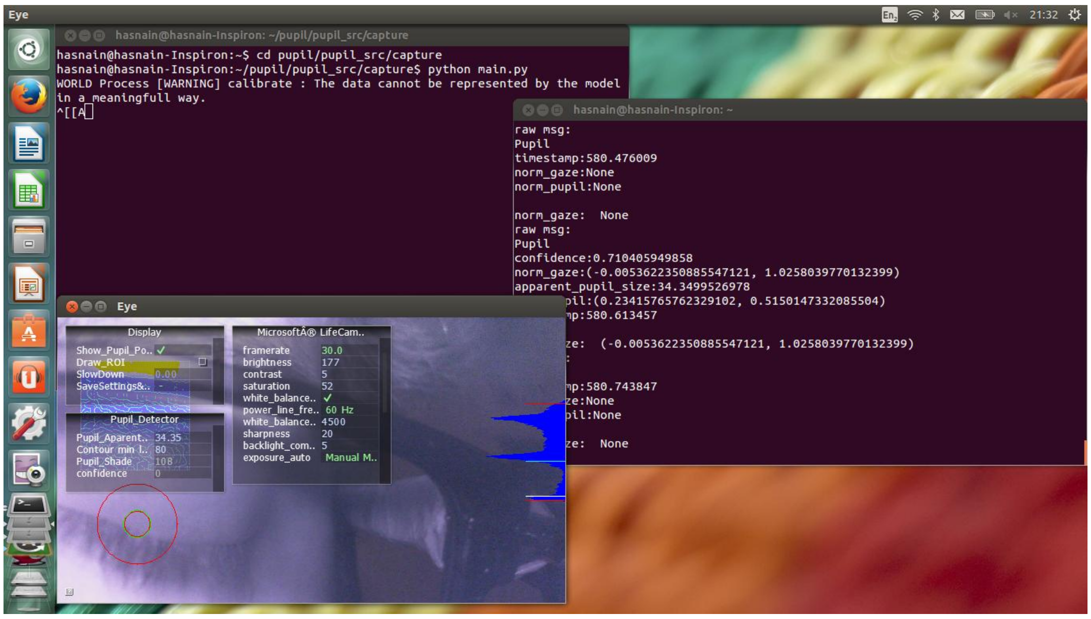
```
hasnain@hasnain-Inspiron:~$ cd pupil/pupil_src/capture
hasnain@hasnain-Inspiron:~/pupil/pupil_src/capture$ python main.py
WORLD Process [WARNING] calibrate : The data cannot be represented by the model
in a meaningfull way.
^[[A
```

```
norm_pupil:(0.19216503500938414, 0.3808894157409668)
roi_center:(48.98562240600586, 39.87981033325195)
timestamp:575.277074

norm_gaze:  (0.28355601028682642, -0.36550485043008507)
raw msg:
Pupil
confidence:1.06045061664
norm_gaze:(0.29221963194278899, -0.35769772484948703)
norm_pupil:(0.1915848135948181, 0.3815138287014431)
roi_center:(49.614280700683594, 39.65502166748047)
timestamp:575.413915

ze:  (0.29221963194278899, -0.35769772484948703)
:

nce:1.11306089172
ze:(0.29267163236109539, -0.35896251232740894)
pil:(0.1916881322860718, 0.3817904578314887)
ter:(122.68040466308594, 123.55543518066406)
mp:575.546669

ze:  (0.29267163236109539, -0.35896251232740894)
```

```
Show_Pupil_Po..     ✓
Draw_ROI
SlowDown            0.00
SaveSettings&..     -

Pupil_Aparent..     50.3259
Contour min L..     80
Pupil_Shade         68
confidence          1.11306

framerate           30.0
brightness          177
contrast            5
saturation          52
white_balance..     ✓
power_line_fre..    60 Hz
white_balance..     4500
sharpness           20
backlight_com..     5
exposure_auto       Manual M..
```

Figure 17: Gaze and pupil positions transmitted in real time

Also, 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 data transmitted during a blink can be seen below:


Eye
hasnain@hasnain-Inspiron: ~/pupil/pupil_src/capture
hasnain@hasnain-Inspiron: ~
21:32

```
hasnain@hasnain-Inspiron:~$ cd pupil/pupil_src/capture
hasnain@hasnain-Inspiron:~/pupil/pupil_src/capture$ python main.py
WORLD Process [WARNING] calibrate : The data cannot be represented by the model
in a meaningfull way.
^[[A
```

```
raw msg:
Pupil
timestamp:580.476009
norm_gaze:None
norm_pupil:None

norm_gaze:  None
raw msg:
Pupil
confidence:0.710405949858
norm_gaze:(-0.0053622350885547121, 1.0258039770132399)
apparent_pupil_size:34.3499526978
   pil:(0.23415765762329102, 0.5150147332085504)
   mp:580.613457

   ze:  (-0.0053622350885547121, 1.0258039770132399)
   :

   mp:580.743847
   ze:None
   pil:None

   ze:  None
```

Display	
Show_Pupil_Po..	✓
Draw_ROI	☐
SlowDown	0.00
SaveSettings&..	-

Pupil_Detector	
Pupil_Aparent..	34.35
Contour min l..	80
Pupil_Shade	108
confidence	0

MicrosoftÂ® LifeCam..	
framerate	30.0
brightness	177
contrast	5
saturation	52
white_balance..	✓
power_line_fre..	60 Hz
white_balance..	4500
sharpness	20
backlight_com..	5
exposure_auto	Manual M..

Figure 18: No gaze and eye positions transmitted during a blink

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.

Note that the smoothing factor is what reduces the jitteriness of the mouse, because the eye is a faster input device compared to a mouse (hand), gaze changes rapidly hence mouse control needs to be precise and non-jittery, here a test value of 0.5 is used.

One major result that most of us were interested in was whether or not the eyes are a feasible way of human computer interaction. After all glimpses and other random eye movements add a lot off erroneous and jerky movements in the cursor control if compared to the relatively smoother movements of the mouse. We see here first of all that the project targets mostly ALS patients or patients with other physical disabilities that render their bodies from the neck down paralyzed. To correct the eye jitter and error, we add something known as "dwell time", which simply refers to keep staring at an interface button for a predetermined amount of time, and the button activates. It is somewhat different from the point and click mechanism of the mouse. The usual values of dwell time are specified as 4ms for very accurate gaze controllers, down to about 50ms for inaccurate ones, for a dwell time of 4ms, we tested out different actions, A through D, and noted down the time required to click an interface button, to give us an idea of how well the interface may perform in the real world. 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 20.

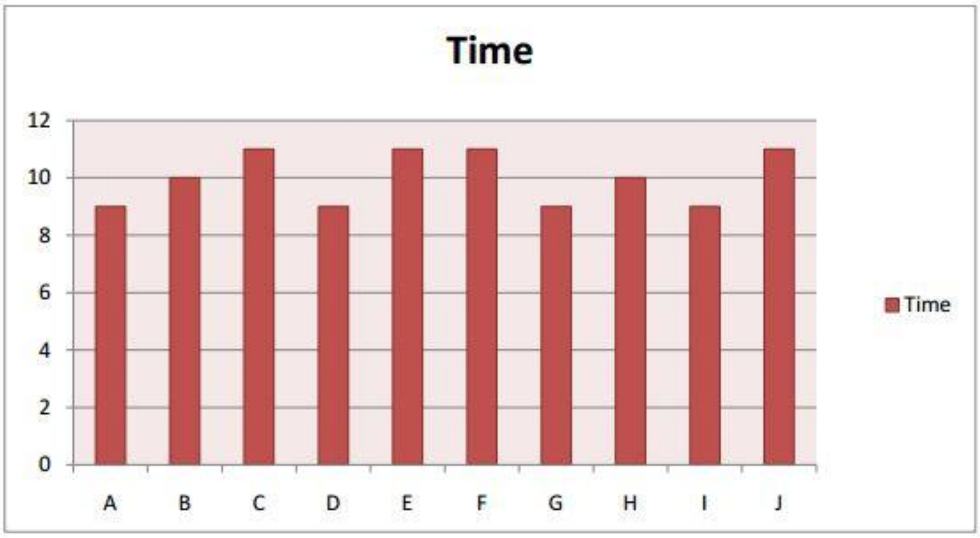


Figure 19: Timings of actions A through J for mouse control

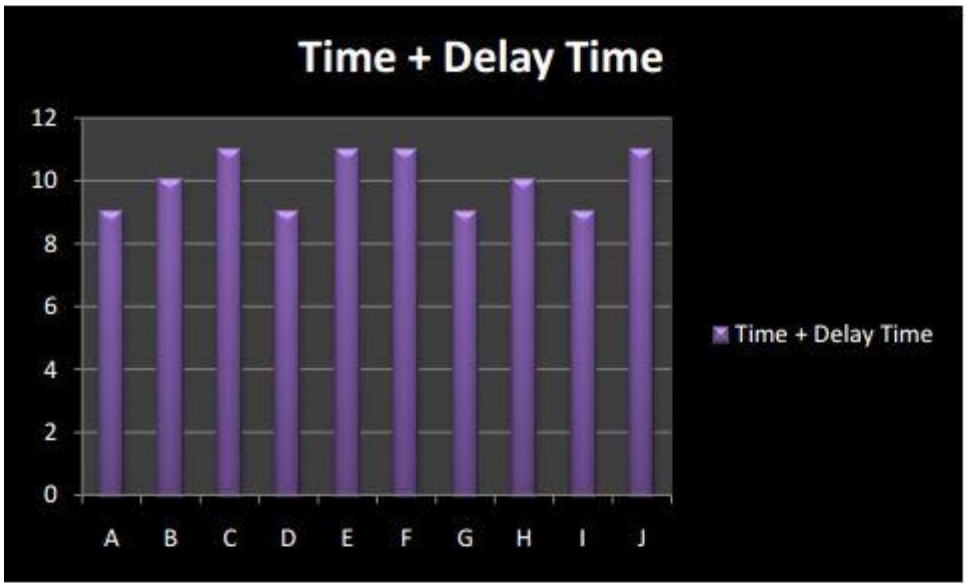


Figure 20: Timings of actions A through J for eye control

Do note that the results are rounded off to the nearest millisecond. We see that once proficiency has been achieved, the user can perform the same tasks in approximately the same amount of time as can be done with mouse. Although typing is one major area where the eye pointer suffers, hence if using as a mode of communication, the user can simply use a graphics tool to draw letters rather than typing them on with a virtual keyboard which is a tedious task even for a mouse user. We also see that since the project interface is highly customized towards control of a robotic arm, we negate the issues faced with the loss of the use of a keyboard and provide a complete interface that enables feasible operation of the arm, and in theory, any other digital/Arduino compatible machinery. The OWI 535 was connected to the Arduino/Eye control setup as explained above and simple test movement have been performed, the image below shows the arm connected to the Arduino microcontroller and the motor driver:

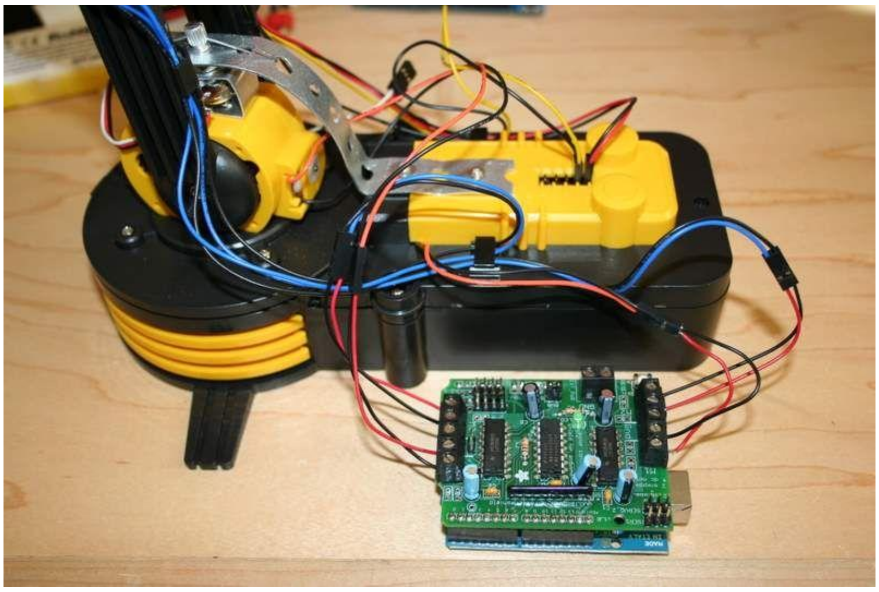


Figure 21: The Arduino and the motor driver connected to the OWI 535

A suitable form of control for the robotic arm is achieved by mounting the world camera on the jaws of the gripper itself, thus enabling the user to stare at the world cam window enabling them to look through the first person perspective of the robot, moving it by the mouse control schemes shown above. For this specialized control, the screen can be divided into different regions, when a user looks into a different region, the arm moves in its corresponding direction. There are several modes of control, it depends on the specific patients with which mode of control they are the most comfortable.

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.

Eye tracking has also been proposed for the PC, with companies like Tobii showing off the technology at the 2012 version of the Consumer Electronics Show. The technology lets users gaze at a desktop computer, tablet or laptop and use eye movements to play a game or interact with applications on Microsoft's Windows 8 operating system. The device sends a pattern of infrared light to the user's eyes and tracks its reflections. Unlike pointing a laser at your eye, it doesn't hurt. This technology provides similar precision as touch. 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

Understanding user’s behaviour and expectations for web search can be very valuable for site developers and web workers. While many SEO techniques rely on the actual actions of the user, for example mouse clicks or query streams, eye tracking can give us more detailed observations about how users actually interact with the information in front of them [21].

The fact that the top search results get the most attention from users is self-evident. But a study by Google backed up claims that strategies for scanning search results are different for different task types. These two task types are defined as transactional and informational. It means that you must understand which of these terms describes your website before deciding on SEO and SEM activity. Eye-tracking result for a transaction-oriented query has been shown in figure 22.

What these images present is rather bad news for informative websites. While transaction oriented sites can afford to be further down the search listings, information-oriented sites cannot. We can see that 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.

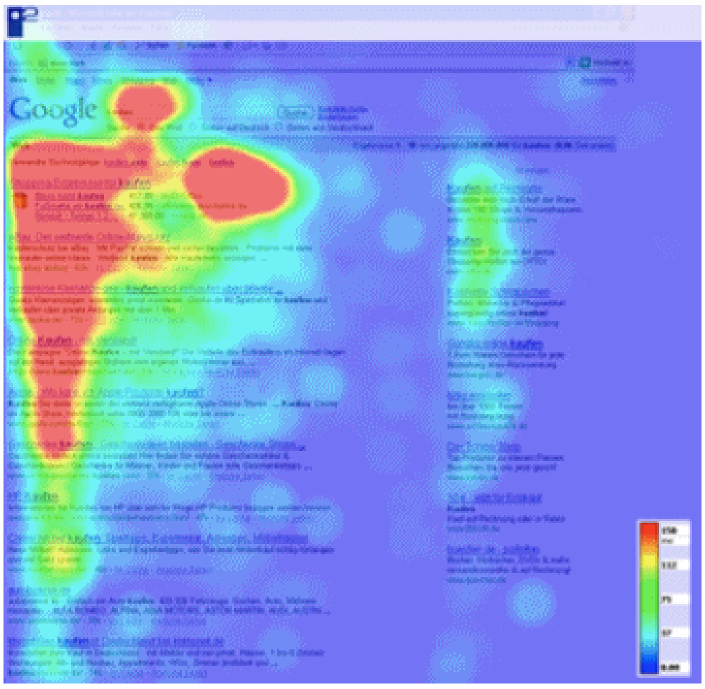


Figure 22: Eye Tracking result for transaction type query

And for an information-oriented query [21]:

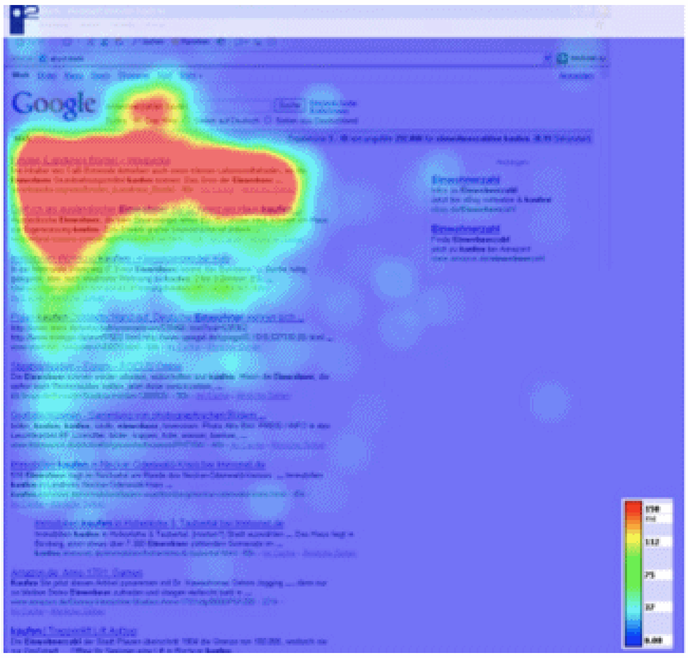


Figure 23: Eye gaze results for informational query

7.2.2 Market Research and Advertising Testing

The perhaps the biggest field in terms of money is the use of eye trackers for market research. When designing posters for an advertisement campaign the marketing research department likes to test the materials. They present the posters to potential clients whose eyes are tracked to get answers to questions like “Did the person look at the product?”, “How much time did the gaze spend on the company’s logo?” and so on. With a portable eye tracker it is also possible to send people to a supermarket to find out which products the people notice and how much influence the form or colour or positioning of the product has on being noticed. An example is shown in figure 24 below:

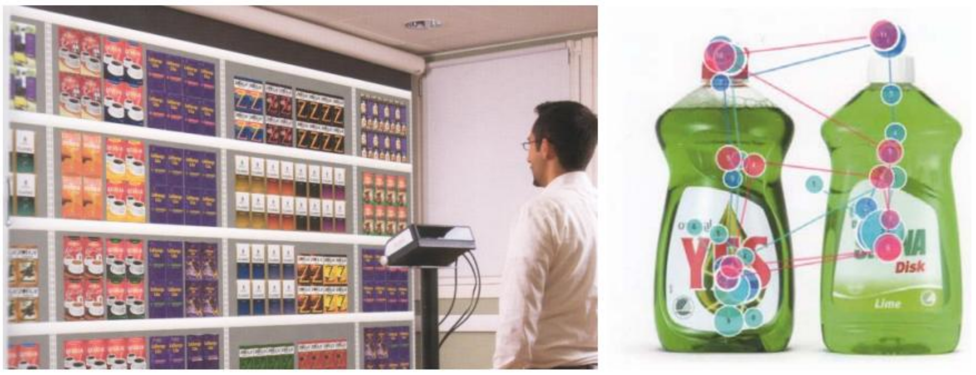


Figure 24: Customer gazing upon a product, points of visual attention marked by eye tracker

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 Usability Research

Another field of commercial interest is usability testing. The first use of eye trackers was done for the American air force to find out the best positions for the controls in an aircraft cockpit. When offering a new device to somebody whose eyes are tracked, it is easy to see where the gaze moves in the expectation to find the control for solving the given task.

7.2.4 Gaze interaction and car assistant systems

Future applications for eye-tracking technologies are gaze interaction for regular users. A TV set could switch on by itself when somebody is looking at it. The car industry does research in this field too, with the aim of developing assistant systems for cars. For example, an eye tracker in the car could warn the driver when she or he falls asleep while driving the car. This field of application for eye tracking is not yet commercially available and needs further research.

7.3 Potential Future Areas Requiring Further Research

At the moment eye-tracker systems are available but they are not prepared for working as an input device except for accessibility systems which do not bring benefit to the regular users. Most eye trackers existing today serve the purpose of recording and analyzing gaze data and the demand in this market is low compared to the demand for input devices like mouse devices or webcams. Consequently, the prices for eye trackers are still high, again compared to mouse devices or webcams.

The future of gaze-aware systems will depend on an application. The graphical user interface was the application that pushed the mouse device from a special input device for CAD (computer aided design) engineers to an input device for the masses.

Although it is possible to direct a graphical user interface solely with the keyboard and without a mouse, in many cases even more efficiently, most people are not able to operate such a system if the mouse is missing or not working. This is the reason for the big success of the mouse device [4], [12]. There is no comparable application for gaze-aware systems on the horizon that everybody wants to have and which does not work well without eye tracker as it was the case for graphical user interface and mouse.

An application that could have the potential to create a mass market for eye-tracking technologies is a computer game. Computer games are a growing market and special input devices for game stations fill the shops. An eye tracker for computer games could come along as a head-mounted device, a headset with earphones and microphone, which are commonly used for gaming already, and with two extra cameras. One camera is mounted near the microphone and focuses on one eye and the other camera next to one earphone with the same view as the eye. The camera at the microphone tracks the eye and because it has a rigid connection with the head, head movements do not influence it [3]. The camera at the earphone sees the display and can calculate the head position from detecting the corners of the display or from matching the known display content to the camera picture. The hardware for such an eye tracker consists of a headset, two webcams and perhaps an infrared LED and all together is available for less than 100 Euro or Dollars. The main costs are the software development and the effort to make it a product. These costs are small per piece if produced in high quantities. The reason why an eye tracker could be successful in the market as input device for computer games lies in the speed. Eye-tracking interaction is fast if the targets are not too small and if an extra input modality is used. For a typical shooting game the targets are big enough in size and the extra input modality is the fire button. While a saving of 300 milliseconds for a pointing operation does not make much difference for a spreadsheet application or a word processor, it makes a big difference for an action game. The excitement and finally the level of adrenalin in the body are directly related to the speed of the game. Research on eye tracker input in first person shooter games [19] could not (yet) show an increase in performance compared to classical mouse input. This result is in contradiction to the findings for the "hardware button" of Ware and Mikaelian [20] where gaze positioning together with key input were significantly faster than a classical mouse. As the computer game industry always searches for new ideas it is only a question of time until cheap eye trackers for gaming will be in the shops. The availability of cheap eye trackers will lead to the development of further applications for such an eye tracker.

Attention sensors for mobile devices are a further possibility to introduce eye tracking to the mass market. The costs for an attention sensor are negligible and the manufacturers of mobile devices always look for new features to have an advantage in the highly competitive market. Such an attention sensor in a mobile video viewer can provide the functionality of pausing the video when not looking at it. It can also provide a power-saving function by switching off the display when nobody is looking at it. The careful use of energy is very important for mobile devices as the capacity of the batteries is limited [1]. A laptop typically switches off the display after a certain time without key or mouse input. This concept does not work when watching a video as there is no input from the mouse or keyboard. The problem is also well known from mobile MP3 players which try to solve the problem with a HOLD switch. While it seems to be nearly impossible to detect whether somebody listens to audio content an eye tracker can detect whether somebody is watching video content. A good chance for eye tracking in the smaller high-end market is the trend to large displays.

Interaction with large displays or multiple monitor setups by a mouse has problems. One problem is that people cannot find the mouse pointer on the large display area; another problem is how to adjust the control-gain ratio for the mouse. A high gain causes problems for the precision of the mouse movement as explained by Fitts' law while a low gain will lead to mouse movements which exceed the range of the hand. Concepts like focus activation by gaze [5] or MAGIC pointing [2] and preferably MAGIC touch can help. As large displays are still expensive, the cost for the eye-tracking device does not contribute to the total costs too much. The MAGIC touch principle also saves many hand movements and for this reason helps people who suffer from RSI (repetitive strain injuries). As medical treatment is expensive an eye tracker and a touch-sensitive mouse can be the cheaper alternative. All these visions of gaze-aware systems could become reality within the next years and some probably will.

Prophecies for longer periods are speculations. Nevertheless, it is clear that the evolution of human-computer interfaces will lead to systems that are more 'human' and not to systems where the humans have to act like computers. As the eye gaze is very important for the human-human interaction, it will definitely be very important for the development of future human-computer interfaces.

8. CONCLUSION

As a final review, the project aims to deliver a low cost eye tracker that can produce digital outputs interface-able to external TTL machinery that is mainly directed for use by the physically disabled. The project highlights its software uses via the control of a mouse, and its hardware uses via the control of an OWI 535 robotic arm edge. The designed system is low cost and efficient, utilizing only an Arduino board, a Motor driver, and SMD IR LEDs for eye illumination, software modules coded in C++ and Python Programming language, running on a laptop to run a worm drive based robotic arm. Problems that arise such as jittery eye movements and direction of attention and pupil stability are negated by the use of large interface buttons with dwell time even on a blink gesture, these strategies help eliminate sporadic movements of the cursor and enable efficient control of the mouse and the external hardware (in this case, the robotic arm). The project also highlights the outputs of the contour detection algorithm used to detect the eye within specified min/max pupil size thresholds and defined regions of interest with a colour spectrum showing up on the side. The debug window provides insight into the detection of the pupil with a histogram to hunt down bugs and jitter in the eye detection or eye to world mapping processes. 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

Arduino – A development board armed with EEPROM Flash Memory bundled with onboard Microprocessor that allows code to be burned to it again and again and is capable of Analogue and Digital Communications

Arduino Due – A special edition of Arduino that uses an ARM Cortex M3 CPU as Processor instead of the usual ATMEGA to provide better processing speed

DTMF Decoder – A dual tone multi-frequency decoder, it decode the frequencies on a dial pad of phone and translates them to binary outputs of the key number pressed

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

Ellipse Fitting – A scheme where an ellipse is detected by tracing the colour gradient from the image edges. A smooth colour gradient identifies an ellipse.

Git – A famous Software Version Control system, used by developers to keep track of application code and manage their versioning

GLFW3 – An Open Source, multi-platform library for creating windows with OpenGL contexts and managing input and events

L293D – A special Motor Driver IC used to control Motors under a specific power limit; used in small Motor Driver Circuits

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

Motor Driver – A circuit that amplifies current whilst also reducing timing jitter used for precision motor control

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

OWI Robotic Arm – A DIY Robotic Arm, manufactured by OWI Robotics; used by enthusiasts to utilize features of Robotic Arms without building them from scratch

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

Pupil Framework – An eye-tracking software framework written in Python and C++ that can pinpoint eyesight on real-world objects

Servo Motor – A type of motor that works on Pulse Width Modulated (PWM) signals to rotate to pre-specified positions.

SMD – Stands for "Surface Mounted Device", used as SMD LEDs in the document; very small LEDs that are directly soldered on the PCB as an IC

TTL – Transistor Logic (TTL) is a class of digital circuits built from bipolar junction transistors (BJT) and resistors. It is called transistor logic because amplification and gating are both done by transistors

Visible light Filter – A filter lens or IR written that blocks visible light

APPENDIX B – MAIN CODE and SCRIPTS

The main python code for the eye detection application is given below:

```
'''
(*)~----------------------------------------------------------------
---------------------
Eye Detection and Tracking Application
NC Hasnain Raza | NC Shaheer Cheema | NC Ali Sheheryar | NC
Sheharyar Naseer
-----------------------------------------------------------------
------------------~(*)
'''

import sys, os,platform
from time import sleep
from ctypes import c_bool, c_int
if platform.system() == 'Darwin':
    from billiard import Process, Pipe,
Event,Queue,forking_enable,freeze_support
    from billiard.sharedctypes import RawValue, Value, Array
else:
    from multiprocessing import Process, Pipe, Event, Queue
    forking_enable = lambda x: x #dummy fn
    from multiprocessing import freeze_support
    from multiprocessing.sharedctypes import RawValue, Value,
Array

if getattr(sys, 'frozen', False):
    if platform.system() == 'Darwin':
        user_dir = os.path.expanduser('~/Desktop/pupil_settings')
        rec_dir =
os.path.expanduser('~/Desktop/pupil_recordings')
        version_file =
os.path.join(sys._MEIPASS,'_version_string_')
    else:
        # Specifiy user dirs.
        user_dir =
os.path.join(sys._MEIPASS.rsplit(os.path.sep,1)[0],"settings")
        rec_dir =
os.path.join(sys._MEIPASS.rsplit(os.path.sep,1)[0],"recordings")
        version_file =
os.path.join(sys._MEIPASS,'_version_string_')


else:
    # We are running in a normal Python environment.
    # Make all pupil shared_modules available to this Python
session.
    pupil_base_dir =
os.path.abspath(__file__).rsplit('pupil_src', 1)[0]
    sys.path.append(os.path.join(pupil_base_dir, 'pupil_src',
'shared_modules'))
      # Specifiy user dirs.
    rec_dir = os.path.join(pupil_base_dir,'recordings')
    user_dir = os.path.join(pupil_base_dir,'settings')


# create folder for user settings, tmp data and a recordings
folder
if not os.path.isdir(user_dir):
    os.mkdir(user_dir)
if not os.path.isdir(rec_dir):
    os.mkdir(rec_dir)


import logging
# Set up root logger for the main process before doing imports of
logged modules.
logger = logging.getLogger()
logger.setLevel(logging.DEBUG)
# create file handler which logs even debug messages
fh =
logging.FileHandler(os.path.join(user_dir,'world.log'),mode='w')
fh.setLevel(logging.DEBUG)
# create console handler with a higher log level
ch = logging.StreamHandler()
ch.setLevel(logging.WARNING)
# create formatter and add it to the handlers
formatter = logging.Formatter('World Process: %(asctime)s -
%(name)s - %(levelname)s - %(message)s')
fh.setFormatter(formatter)
formatter = logging.Formatter('WORLD Process [%(levelname)s]
%(name)s : %(message)s')
ch.setFormatter(formatter)
# add the handlers to the logger
logger.addHandler(fh)
logger.addHandler(ch)
# mute OpenGL logger
logging.getLogger("OpenGL").propagate = False
logging.getLogger("OpenGL").addHandler(logging.NullHandler())


#if you pass any additional argument when calling this script.
The profiler will be used.
if len(sys.argv) >=2:
    from eye import eye_profiled as eye
    from world import world_profiled as world
else:
    from eye import eye
    from world import world

from methods import Temp

#get the current software version
if getattr(sys, 'frozen', False):
    with open(version_file) as f:
        version = f.read()
else:
    from git_version import get_tag_commit
    version = get_tag_commit()


def main():
    # To assign camera by name: put string(s) in list
    #eye_src = ["Microdia Sonix 1.3 MP Laptop Integrated Webcam"]
    #eye_src = ["Microsoft", "6000","Integrated Camera"]
    #world_src = ["Logitech Camera","(046d:081d)","C510","B525",
"C525","C615","C920","C930e"]

    # to assign cameras directly, using integers as demonstrated
below
    eye_src = 2
    world_src = 1

    # to use a pre-recorded video.
    # Use a string to specify the path to your video file as
demonstrated below
    # eye_src =
"/Users/mkassner/Pupil/datasets/eye2_fieldtest/eye 10.avi"
    # world_src =
"/Users/mkassner/Desktop/2014_01_21/000/world.avi"

    # Camera video size in pixels (width,height)
    eye_size = (640,360)
    world_size = (1280,720)


    # on MacOS we will not use os.fork, elsewhere this does
nothing.
    forking_enable(0)

    # Create and initialize IPC
    g_pool = Temp()
    g_pool.pupil_queue = Queue()
    g_pool.eye_rx, g_pool.eye_tx = Pipe(False)
    g_pool.quit = RawValue(c_bool,0)
    # make some constants avaiable
    g_pool.user_dir = user_dir
    g_pool.rec_dir = rec_dir
    g_pool.version = version
    g_pool.app = 'capture'
    # set up subprocesses
    p_eye = Process(target=eye, args=(g_pool,eye_src,eye_size))

    # Spawn subprocess:
    p_eye.start()
    if platform.system() == 'Linux':
        # We need to give the camera driver some time before
requesting another camera.
        sleep(0.5)

    world(g_pool,world_src,world_size)

    # Exit / clean-up
    p_eye.join()

if __name__ == '__main__':
    freeze_support()
    main()
```

Another code set of the actual Eye detection Algorithm is given below:

```
# make shared modules available across pupil_src
if __name__ == '__main__':
    from sys import path as syspath
    from os import path as ospath
    loc = ospath.abspath(__file__).rsplit('pupil_src', 1)
    syspath.append(ospath.join(loc[0], 'pupil_src',
'shared_modules'))
    del syspath, ospath

import cv2
from time import sleep
import numpy as np
from methods import *
import atb
from ctypes import c_int,c_bool,c_float
import logging
logger = logging.getLogger(__name__)
from c_methods import eye_filter
import random
from glfw import *
from gl_utils import adjust_gl_view, draw_gl_texture,
clear_gl_screen, draw_gl_point_norm,
draw_gl_polyline,basic_gl_setup


class Pupil_Detector(object):
    """base class for pupil detector"""
    def __init__(self):
        super(Pupil_Detector, self).__init__()
        var1 = c_int(0)

    def detect(self,frame,u_roi,p_roi,visualize=False):
        img = frame.img
        # hint: create a view into the img with the bounds of the
coarse pupil estimation
        pupil_img =
img[u_roi.lY:u_roi.uY,u_roi.lX:u_roi.uX][p_roi.lY:p_roi.uY,p_roi.
lX:p_roi.uX]

        if visualize:
            # draw into image whatever you like and it will be
displayed
            # otherwise you shall not modify img data inplace!
            pass

        candidate_pupil_ellipse = {'center': (None,None),
                        'axes': (None, None),
                        'angle': None,
                        'area': None,
                        'ratio': None,
                        'major': None,
                        'minor': None,
                        'goodness': 0} #some estimation on how
sure you are about the detected ellipse and its fit. Smaller is
better
```

```
        # If you use region of interest p_roi and u_roi make sure
to return pupil coordinates relative to the full image
        candidate_pupil_ellipse['center'] =
u_roi.add_vector(p_roi.add_vector(candidate_pupil_ellipse['center
']))
        candidate_pupil_ellipse['timestamp'] = frame.timestamp
        result = candidate_pupil_ellipse #we found something
        if result:
            return candidate_pupil_ellipse # all this will be
sent to the world process, you can add whateever you need to
this.

        else:
            self.goodness.value = 100
            no_result = {}
            no_result['timestamp'] = frame.timestamp
            no_result['norm_pupil'] = None
            return no_result


    def create_atb_bar(self,pos):
        self.bar = atb.Bar(name = "Pupil_Detector", label="Pupil
Detector Controls",
            help="pupil detection params", color=(50, 50, 50),
alpha=100,
            text='light', position=pos,refresh=.3, size=(200,
200))
        self.bar.add_var("VAR1",self.var1,
step=1.,readonly=False)




class MSER_Detector(Pupil_Detector):
    """docstring for MSER_Detector"""
    def __init__(self):
        super(MSER_Detector, self).__init__()

    def detect(self,frame,u_roi,visualize=False):
        #get the user_roi
        img = frame.img
        # r_img = img[u_roi.lY:u_roi.uY,u_roi.lX:u_roi.uX]
        debug= True
        PARAMS = {'_delta':10, '_min_area': 2000, '_max_area':
10000, '_max_variation': .25, '_min_diversity': .2,
'_max_evolution': 200, '_area_threshold': 1.01, '_min_margin':
.003, '_edge_blur_size': 7}
        pupil_intensity= 150
        pupil_ratio= 2
        mser = cv2.MSER(**PARAMS)
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        regions = mser.detect(gray, None)
        hulls = []
        # Select most circular hull
        for region in regions:
            h = cv2.convexHull(region.reshape(-1, 1,
2)).reshape((-1, 2))
```

```
            cv2.drawContours(frame.img,[h],-1,(255,0,0))
            hc = h - np.mean(h, 0)
            _, s, _ = np.linalg.svd(hc)
            r = s[0] / s[1]
            if r > pupil_ratio:
                logger.debug('Skipping ratio %f > %f' % (r,
pupil_ratio))
                continue
            mval = np.median(gray.flat[np.dot(region,
np.array([1, img.shape[1]]))])
            if mval > pupil_intensity:
                logger.debug('Skipping intensity %f > %f' %
(mval,pupil_intensity))
                continue
            logger.debug('Kept: Area[%f] Intensity[%f] Ratio[%f]'
% (region.shape[0], mval, r))
            hulls.append((r, region, h))
        if hulls:
            hulls.sort()
            gaze = np.round(np.mean(hulls[0][2].reshape((-1, 2)),
0)).astype(np.int).tolist()
            logger.debug('Gaze[%d,%d]' % (gaze[0], gaze[1]))
            norm_pupil = normalize((gaze[0], gaze[1]),
(img.shape[1], img.shape[0]),flip_y=True )
            return
{'norm_pupil':norm_pupil,'timestamp':frame.timestamp,'center':(ga
ze[0], gaze[1])}
        else:
            return
{'norm_pupil':None,'timestamp':frame.timestamp}



    def create_atb_bar(self,pos):
        self.bar = atb.Bar(name = "MSER_Detector", label="MSER
PUPIL Detector Controls",
            help="pupil detection params", color=(50, 50, 50),
alpha=100,
            text='light', position=pos,refresh=.3, size=(200,
200))
        # self.bar.add_var("VAR1",self.var1,
step=1.,readonly=False)


class Canny_Detector(Pupil_Detector):
    """a Pupil detector based on Canny_Edges"""
    def __init__(self):
        super(Canny_Detector, self).__init__()

        # coase pupil filter params
        self.coarse_filter_min = 100
        self.coarse_filter_max = 400

        # canny edge detection params
        self.blur = c_int(1)
        self.canny_thresh = c_int(200)
        self.canny_ratio= c_int(2)
        self.canny_aperture = c_int(7)
        # edge intensity filter params
        self.intensity_range = c_int(17)
        self.bin_thresh = c_int(0)

        # contour prefilter params
        self.min_contour_size = 80

        #ellipse filter params
        self.target_ratio=1.0
        self.target_size=c_float(100.)
        self.goodness = c_float(1.)
        self.size_tolerance=10.


        #debug window
        self._window = None
        self.window_should_open = False
        self.window_should_close = False

        #debug settings
        self.should_sleep = False

    def detect(self,frame,u_roi,visualize=False):

        if self.window_should_open:
            self.open_window()
        if self.window_should_close:
            self.close_window()



        #get the user_roi
        img = frame.img
        r_img = img[u_roi.lY:u_roi.uY,u_roi.lX:u_roi.uX]
        gray_img = grayscale(r_img)


        # coarse pupil detection
        integral = cv2.integral(gray_img)
        integral =  np.array(integral,dtype=c_float)
        x,y,w,response =
eye_filter(integral,self.coarse_filter_min,self.coarse_filter_max
)
        p_roi = Roi(gray_img.shape)
        if w>0:
            p_roi.set((y,x,y+w,x+w))
        else:
            p_roi.set((0,0,-1,-1))
        coarse_pupil_center = x+w/2.,y+w/2.
        coarse_pupil_width = w/2.
        padding = coarse_pupil_width/4.
        pupil_img = gray_img[p_roi.lY:p_roi.uY,p_roi.lX:p_roi.uX]



        # binary thresholding of pupil dark areas
        hist = cv2.calcHist([pupil_img],[0],None,[256],[0,256])
#(images, channels, mask, histSize, ranges[, hist[, accumulate]])
        bins = np.arange(hist.shape[0])
        spikes = bins[hist[:,0]>40] # every intensity seen in
more than 40 pixels
        if spikes.shape[0] >0:
            lowest_spike = spikes.min()
            highest_spike = spikes.max()
        else:
            lowest_spike = 200
            highest_spike = 255

        offset = self.intensity_range.value
        spectral_offset = 5
        if visualize:
            # display the histogram
            sx,sy = 100,1
            colors =
((0,0,255),(255,0,0),(255,255,0),(255,255,255))
            h,w,chan = img.shape
            hist *= 1./hist.max()  # normalize for display

            for i,h in zip(bins,hist[:,0]):
                c = colors[1]
                cv2.line(img,(w,int(i*sy)),(w-
int(h*sx),int(i*sy)),c)
            cv2.line(img,(w,int(lowest_spike*sy)),(int(w-
.5*sx),int(lowest_spike*sy)),colors[0])

cv2.line(img,(w,int((lowest_spike+offset)*sy)),(int(w-
.5*sx),int((lowest_spike+offset)*sy)),colors[2])
            cv2.line(img,(w,int((highest_spike)*sy)),(int(w-
.5*sx),int((highest_spike)*sy)),colors[0])
            cv2.line(img,(w,int((highest_spike- spectral_offset
)*sy)),(int(w-.5*sx),int((highest_spike -
spectral_offset)*sy)),colors[3])

        # create dark and spectral glint masks
        self.bin_thresh.value = lowest_spike
        binary_img =
bin_thresholding(pupil_img,image_upper=lowest_spike + offset)
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,
(7,7))
        cv2.dilate(binary_img, kernel,binary_img, iterations=2)
        spec_mask = bin_thresholding(pupil_img,
image_upper=highest_spike - spectral_offset)
        cv2.erode(spec_mask, kernel,spec_mask, iterations=1)

        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,
(9,9))

        #open operation to remove eye lashes
        pupil_img = cv2.morphologyEx(pupil_img, cv2.MORPH_OPEN,
kernel)

        if self.blur.value >1:
            pupil_img = cv2.medianBlur(pupil_img,self.blur.value)

        edges = cv2.Canny(pupil_img,
                            self.canny_thresh.value,

self.canny_thresh.value*self.canny_ratio.value,
                              apertureSize=
self.canny_aperture.value)


        # edges = cv2.adaptiveThreshold(pupil_img,255,
cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY_INV,
self.canny_aperture.value, 7)

        # remove edges in areas not dark enough and where the
glint is (spectral refelction from IR leds)
        edges = cv2.min(edges, spec_mask)
        edges = cv2.min(edges,binary_img)

        if visualize:
            overlay =
img[u_roi.lY:u_roi.uY,u_roi.lX:u_roi.uX][p_roi.lY:p_roi.uY,p_roi.
lX:p_roi.uX]
            chn_img = grayscale(overlay)
            overlay[:,:,2] = cv2.max(chn_img,edges) #b channel
            overlay[:,:,0] = cv2.max(chn_img,binary_img) #g
channel
            overlay[:,:,1] = cv2.min(chn_img,spec_mask) #b
channel

            pupil_img =
frame.img[u_roi.lY:u_roi.uY,u_roi.lX:u_roi.uX][p_roi.lY:p_roi.uY,
p_roi.lX:p_roi.uX]
            # draw a frame around the automatic pupil ROI in
overlay...
            pupil_img[::2,0] = 255,255,255
            pupil_img[::2,-1]= 255,255,255
            pupil_img[0,::2] = 255,255,255
            pupil_img[-1,::2]= 255,255,255

            pupil_img[::2,padding] = 255,255,255
            pupil_img[::2,-padding]= 255,255,255
            pupil_img[padding,::2] = 255,255,255
            pupil_img[-padding,::2]= 255,255,255


frame.img[u_roi.lY:u_roi.uY,u_roi.lX:u_roi.uX][p_roi.lY:p_roi.uY,
p_roi.lX:p_roi.uX] = pupil_img


        # from edges to contours
        contours, hierarchy = cv2.findContours(edges,
                                        mode=cv2.RETR_LIST,

method=cv2.CHAIN_APPROX_NONE,offset=(0,0)) #TC89_KCOS
        # contours is a list containing array([[[108,
290]],[[111, 290]]], dtype=int32) shape=(number of
points,1,dimension(2) )


        ### first we want to filter out the bad stuff
        # to short
        good_contours = [c for c in contours if
c.shape[0]>self.min_contour_size]
```

```
        # now we learn things about each contour though looking
at the curvature. For this we need to simplyfy the contour
        arprox_contours =
[cv2.approxPolyDP(c,epsilon=1.5,closed=False) for c in
good_contours]
        # cv2.drawContours(pupil_img,good_contours,-
1,(255,255,0))
        # cv2.drawContours(pupil_img,arprox_contours,-
1,(0,0,255))

        if self._window:
            debug_img = np.zeros(img.shape,img.dtype)

        x_shift = coarse_pupil_width*2 #just vor display
        color = zip(range(0,250,30),range(0,255,30)[::-
1],range(230,250))
        split_contours = []
        for c in arprox_contours:
            curvature = GetAnglesPolyline(c)
            # print curvature
            # we split whenever there is a real kink
(abs(curvature)<right angle) or a change in the genreal direction
            kink_idx = find_kink_and_dir_change(curvature,100)
            # kinks,k_index = convexity_defect(c,curvature)
            # print "kink_idx", kink_idx
            segs = split_at_corner_index(c,kink_idx)
            # print len(segs)
            # segs.sort(key=lambda e:-len(e))
            for s in segs:
                split_contours.append(s)
                if self._window:
                    c = color.pop(0)
                    color.append(c)
                    # if s.shape[0] >=5:
                    #
cv2.polylines(debug_img,[s],isClosed=False,color=c)
                    s = s.copy()
                    s[:,:,1] +=  coarse_pupil_width*2

cv2.polylines(debug_img,[s],isClosed=False,color=c)
                    s[:,:,0] += x_shift
                    x_shift += 5

cv2.polylines(debug_img,[s],isClosed=False,color=c)
        # return {'timestamp':frame.timestamp,'norm_pupil':None}

        #these segments may now be smaller, we need to get rid of
those not long enough for ellipse fitting
        good_contours = [c for c in split_contours if
c.shape[0]>=5]
        #
cv2.polylines(img,good_contours,isClosed=False,color=(255,255,0))

        shape = edges.shape
        ellipses = ((cv2.fitEllipse(c),c) for c in good_contours)
        ellipses = ((e,c) for e,c in ellipses if (padding <
e[0][1] < shape[0]-padding and padding< e[0][0] < shape[1]-
padding)) # center is close to roi center
```

```
        ellipses = ((e,c) for e,c in ellipses if
binary_img[e[0][1],e[0][0]]) # center is on a dark pixel
        ellipses = [(e,c) for e,c in ellipses if
is_round(e,self.target_ratio)] # roundness test
        result = []
        for e,c in ellipses:
            size_dif = size_deviation(e,self.target_size.value)
            pupil_ellipse = {}
            pupil_ellipse['contour'] = c
            a,b = e[1][0]/2.,e[1][1]/2. # majar minor radii of
candidate ellipse
            pupil_ellipse['circumference'] = np.pi*abs(3*(a+b)-
np.sqrt(10*a*b+3*(a**2+b**2)))
            # pupil_ellipse['convex_hull'] =
cv2.convexHull(pupil_ellipse['contour'])
            pupil_ellipse['contour_area'] =
cv2.contourArea(cv2.convexHull(c))
            pupil_ellipse['ellipse_area'] = np.pi*a*b
            # print abs(pupil_ellipse['contour_area']-
pupil_ellipse['ellipse_area'])
            if abs(pupil_ellipse['contour_area']-
pupil_ellipse['ellipse_area']) <10:
                pupil_ellipse['goodness'] =
abs(pupil_ellipse['contour_area']-
pupil_ellipse['ellipse_area'])/10 #perfect match we'll take this
one
            else:
                pupil_ellipse['goodness'] = size_dif
            if visualize:
                    pass
                    #
cv2.drawContours(pupil_img,[cv2.convexHull(c)],-
1,(size_dif,size_dif,255))
                    # cv2.drawContours(pupil_img,[c],-
1,(size_dif,size_dif,255))
            pupil_ellipse['pupil_center'] = e[0] # compensate for
roi offsets
            pupil_ellipse['center'] =
u_roi.add_vector(p_roi.add_vector(e[0])) # compensate for roi
offsets
            pupil_ellipse['angle'] = e[-1]
            pupil_ellipse['axes'] = e[1]
            pupil_ellipse['major'] = max(e[1])
            pupil_ellipse['minor'] = min(e[1])
            pupil_ellipse['ratio'] =
pupil_ellipse['minor']/pupil_ellipse['major']
            pupil_ellipse['norm_pupil'] =
normalize(pupil_ellipse['center'], (img.shape[1],
img.shape[0]),flip_y=True )
            pupil_ellipse['timestamp'] = frame.timestamp
            result.append(pupil_ellipse)


        #### adding support
        if result:
            result.sort(key=lambda e: e['goodness'])
            # for now we assume that this contour is part of the
pupil
            the_one = result[0]
            # (center, size, angle) =
cv2.fitEllipse(the_one['contour'])
            # print "itself"
            distances =
dist_pts_ellipse(cv2.fitEllipse(the_one['contour']),the_one['cont
our'])
            # print np.average(distances)
            # print np.sum(distances)/float(distances.shape[0])
            # print "other"
            # if self._window:
                # cv2.polylines(debug_img,[result[-
1]['contour']],isClosed=False,color=(255,255,255),thickness=3)
            with_another = np.concatenate((result[-
1]['contour'],the_one['contour']))
            distances =
dist_pts_ellipse(cv2.fitEllipse(with_another),with_another)
            # if 1.5 >
np.sum(distances)/float(distances.shape[0]):
            #     if self._window:
            #         cv2.polylines(debug_img,[result[-
1]['contour']],isClosed=False,color=(255,255,255),thickness=3)

            perimeter_ratio =
cv2.arcLength(the_one["contour"],closed=False)/the_one['circumfer
ence']
            if perimeter_ratio > .9:
                size_thresh = 0
                eccentricity_thresh = 0
            elif perimeter_ratio > .5:
                size_thresh = the_one['major']/(5.)
                eccentricity_thresh = the_one['major']/2.
                self.should_sleep = True
            else:
                size_thresh = the_one['major']/(3.)
                eccentricity_thresh = the_one['major']/2.
                self.should_sleep = True
            if self._window:
                center =
np.uint16(np.around(the_one['pupil_center']))

cv2.circle(debug_img,tuple(center),int(eccentricity_thresh),(0,25
5,0),1)

            if self._window:

cv2.polylines(debug_img,[the_one["contour"]],isClosed=False,color
=(255,0,0),thickness=2)
                s = the_one["contour"].copy()
                s[:,:,0] +=coarse_pupil_width*2

cv2.polylines(debug_img,[s],isClosed=False,color=(255,0,0),thickn
ess=2)
            # but are there other segments that could be used for
support?
            new_support = [the_one['contour'],]
            if len(result)>1:
                the_one = result[0]
                target_axes = the_one['axes'][0]
                # target_mean_curv =
np.mean(curvature(the_one['contour'])
                for e in result:

                    # with_another =
np.concatenate((e['contour'],the_one['contour']))
                    # with_another = np.concatenate([r['contour']
for r in result])
                    with_another = e['contour']
                    distances =
dist_pts_ellipse(cv2.fitEllipse(with_another),with_another)
                    # print np.std(distances)
                    thick =  int(np.std(distances))
                    if 1.5 > np.average(distances) or 1:
                        if self._window:
                            # print thick
                            thick = min(20,thick)

cv2.polylines(debug_img,[e['contour']],isClosed=False,color=(255,
255,255),thickness=thick)

                    if self._window:

cv2.polylines(debug_img,[e["contour"]],isClosed=False,color=(0,10
0,100))
                    center_dist =
cv2.arcLength(np.array([the_one["pupil_center"],e['pupil_center']
],dtype=np.int32),closed=False)
                    size_dif = abs(the_one['major']-e['major'])

                    # #lets make sure the countour is not behind
the_one/'s coutour
                    # center_point =
np.uint16(np.around(the_one['pupil_center']))
                    # other_center_point =
np.uint16(np.around(e['pupil_center']))

                    # mid_point =
the_one["contour"][the_one["contour"].shape[0]/2][0]
                    # other_mid_point =
e["contour"][e["contour"].shape[0]/2][0]

                    # #reflect around mid_point
                    # p = center_point - mid_point
                    # p = np.array((-p[1],-p[0]))
                    # mir_center_point = p + mid_point
                    # dist_mid =
cv2.arcLength(np.array([mid_point,other_mid_point]),closed=False)
                    # dist_center =
cv2.arcLength(np.array([center_point,other_mid_point]),closed=Fal
se)
                    # if self._window:
                    #
cv2.circle(debug_img,tuple(center_point),3,(0,255,0),2)
                    #
cv2.circle(debug_img,tuple(other_center_point),2,(0,0,255),1)
                    #     #
cv2.circle(debug_img,tuple(mir_center_point),3,(0,255,0),2)
```

```
                    #     #
cv2.circle(debug_img,tuple(mid_point),2,(0,255,0),1)
                    #     #
cv2.circle(debug_img,tuple(other_mid_point),2,(0,0,255),1)
                    #
cv2.polylines(debug_img,[np.array([center_point,other_mid_point])
,np.array([mid_point,other_mid_point])],isClosed=False,color=(0,2
55,0))


                    if center_dist < eccentricity_thresh:
                    # print dist_mid-dist_center
                    # if dist_mid > dist_center-20:

                        if  size_dif < size_thresh:


                            new_support.append(e["contour"])
                            if self._window:

cv2.polylines(debug_img,[s],isClosed=False,color=(255,0,0),thickn
ess=1)
                                s = e["contour"].copy()
                                s[:,:,0] +=coarse_pupil_width*2

cv2.polylines(debug_img,[s],isClosed=False,color=(255,255,0),thic
kness=1)

                        else:
                            if self._window:
                                s = e["contour"].copy()
                                s[:,:,0] +=coarse_pupil_width*2

cv2.polylines(debug_img,[s],isClosed=False,color=(0,0,255),thickn
ess=1)
                    else:
                        if self._window:

cv2.polylines(debug_img,[s],isClosed=False,color=(0,255,255),thic
kness=1)

                    # new_support = np.concatenate(new_support)

            self.goodness.value = the_one['goodness']

            ###here we should AND original mask, selected
contours with 2px thinkness (and 2px fitted ellipse -is the last
one a good idea??)
            support_mask = np.zeros(edges.shape,edges.dtype)

cv2.polylines(support_mask,new_support,isClosed=False,color=(255,
255,255),thickness=2)
            # #draw into the suport mast with thickness 2
            new_edges = cv2.min(edges, support_mask)
            new_contours = cv2.findNonZero(new_edges)
            if self._window:

debug_img[0:support_mask.shape[0],0:support_mask.shape[1],2] =
new_edges
```

```
            ###### do the ellipse fit and filter think again
            ellipses = ((cv2.fitEllipse(c),c) for c in
[new_contours])
            ellipses = ((e,c) for e,c in ellipses if (padding <
e[0][1] < shape[0]-padding and padding< e[0][0] < shape[1]-
padding)) # center is close to roi center
            ellipses = ((e,c) for e,c in ellipses if
binary_img[e[0][1],e[0][0]]) # center is on a dark pixel
            ellipses =
[(size_deviation(e,self.target_size.value),e,c) for e,c in
ellipses if is_round(e,self.target_ratio)] # roundness test
            for size_dif,e,c in ellipses:
                pupil_ellipse = {}
                pupil_ellipse['contour'] = c
                a,b = e[1][0]/2.,e[1][1]/2. # majar minor radii
of candidate ellipse
                # pupil_ellipse['circumference'] =
np.pi*abs(3*(a+b)-np.sqrt(10*a*b+3*(a**2+b**2)))
                # pupil_ellipse['convex_hull'] =
cv2.convexHull(pupil_ellipse['contour'])
                pupil_ellipse['contour_area'] =
cv2.contourArea(cv2.convexHull(c))
                pupil_ellipse['ellipse_area'] = np.pi*a*b
                # print abs(pupil_ellipse['contour_area']-
pupil_ellipse['ellipse_area'])
                if abs(pupil_ellipse['contour_area']-
pupil_ellipse['ellipse_area']) <10:
                    pupil_ellipse['goodness'] = 0 #perfect match
we'll take this one
                else:
                    pupil_ellipse['goodness'] = size_dif
                if visualize:
                        pass
                        #
cv2.drawContours(pupil_img,[cv2.convexHull(c)],-
1,(size_dif,size_dif,255))
                        # cv2.drawContours(pupil_img,[c],-
1,(size_dif,size_dif,255))
                pupil_ellipse['center'] =
u_roi.add_vector(p_roi.add_vector(e[0])) # compensate for roi
offsets
                pupil_ellipse['angle'] = e[-1]
                pupil_ellipse['axes'] = e[1]
                pupil_ellipse['major'] = max(e[1])
                pupil_ellipse['minor'] = min(e[1])
                pupil_ellipse['ratio'] =
pupil_ellipse['minor']/pupil_ellipse['major']
                pupil_ellipse['norm_pupil'] =
normalize(pupil_ellipse['center'], (img.shape[1],
img.shape[0]),flip_y=True )
                pupil_ellipse['timestamp'] = frame.timestamp
                result = [pupil_ellipse,]
            # the_new_one = result[0]

            #done - if the new ellipse is good, we just overwrote
the old result
```

```
        if self._window:
            self.gl_display_in_window(debug_img)
            if self.should_sleep:
                # sleep(3)
                self.should_sleep = False
        if result:
            # update the target size
            if result[0]['goodness'] >=3: # perfect match!
                self.target_size.value = result[0]['major']
            else:
                self.target_size.value  = self.target_size.value
+  .2 * (result[0]['major']-self.target_size.value)
                result.sort(key=lambda e: abs(e['major']-
self.target_size.value))
            if visualize:
                pass
            return result[0]

        else:
            self.goodness.value = 100
            no_result = {}
            no_result['timestamp'] = frame.timestamp
            no_result['norm_pupil'] = None
            return no_result



    def create_atb_bar(self,pos):
        self._bar = atb.Bar(name = "Canny_Pupil_Detector",
label="Pupil_Detector",
            help="pupil detection parameters", color=(50, 50,
50), alpha=100,
            text='light', position=pos,refresh=.3, size=(200,
100))
        self._bar.add_button("open debug window",
self.toggle_window)

self._bar.add_var("pupil_intensity_range",self.intensity_range)
        self._bar.add_var("Pupil_Aparent_Size",self.target_size)
        self._bar.add_var("Pupil_Shade",self.bin_thresh,
readonly=True)
        self._bar.add_var("Pupil_Certainty",self.goodness,
readonly=True)
        self._bar.add_var("Image_Blur",self.blur,
step=2,min=1,max=9)
        self._bar.add_var("Canny_aparture",self.canny_aperture,
step=2,min=3,max=7)
        self._bar.add_var("canny_threshold",self.canny_thresh,
step=1,min=0)
        self._bar.add_var("Canny_ratio",self.canny_ratio,
step=1,min=1)

    def toggle_window(self):
        if self._window:
            self.window_should_close = True
        else:
            self.window_should_open = True

    def open_window(self):
        if not self._window:
            if 0: #we are not fullscreening
                monitor =
self.monitor_handles[self.monitor_idx.value]
                mode = glfwGetVideoMode(monitor)
                height,width= mode[0],mode[1]
            else:
                monitor = None
                height,width= 640,360

            active_window = glfwGetCurrentContext()
            self._window = glfwCreateWindow(height, width,
"Plugin Window", monitor=monitor, share=None)
            if not 0:
                glfwSetWindowPos(self._window,200,0)

            self.on_resize(self._window,height,width)

            #Register callbacks

glfwSetWindowSizeCallback(self._window,self.on_resize)
            # glfwSetKeyCallback(self._window,self.on_key)

glfwSetWindowCloseCallback(self._window,self.on_close)

            # gl_state settings
            glfwMakeContextCurrent(self._window)
            basic_gl_setup()
            glfwMakeContextCurrent(active_window)

            self.window_should_open = False

    # window calbacks
    def on_resize(self,window,w, h):
        active_window = glfwGetCurrentContext()
        glfwMakeContextCurrent(window)
        adjust_gl_view(w,h)
        glfwMakeContextCurrent(active_window)

    def on_close(self,window):
        self.window_should_close = True

    def close_window(self):
        if self._window:
            glfwDestroyWindow(self._window)
            self._window = None
            self.window_should_close = False


    def gl_display_in_window(self,img):
        active_window = glfwGetCurrentContext()
        glfwMakeContextCurrent(self._window)
        clear_gl_screen()
        # gl stuff that will show on your plugin window goes here
        draw_gl_texture(img,interpolation=False)
        glfwSwapBuffers(self._window)
        glfwMakeContextCurrent(active_window)
```

```
class Blob_Detector(Pupil_Detector):
    """a Pupil detector based on Canny_Edges"""
    def __init__(self):
        super(Blob_Detector, self).__init__()
        self.intensity_range = c_int(18)
        self.canny_thresh = c_int(200)
        self.canny_ratio= c_int(2)
        self.canny_aperture = c_int(5)


    def detect(self,frame,u_roi,visualize=False):
        #get the user_roi
        img = frame.img
        r_img = img[u_roi.lY:u_roi.uY,u_roi.lX:u_roi.uX]
        gray_img = grayscale(r_img)
        # coarse pupil detection
        integral = cv2.integral(gray_img)
        integral =  np.array(integral,dtype=c_float)
        x,y,w,response = eye_filter(integral,100,400)
        p_roi = Roi(gray_img.shape)
        if w>0:
            p_roi.set((y,x,y+w,x+w))
        else:
            p_roi.set((0,0,-1,-1))
        coarse_pupil_center = x+w/2.,y+w/2.
        coarse_pupil_width = w/2.
        padding = coarse_pupil_width/4.
        pupil_img = gray_img[p_roi.lY:p_roi.uY,p_roi.lX:p_roi.uX]
        # binary thresholding of pupil dark areas
        hist = cv2.calcHist([pupil_img],[0],None,[256],[0,256])
#(images, channels, mask, histSize, ranges[, hist[, accumulate]])
        bins = np.arange(hist.shape[0])
        spikes = bins[hist[:,0]>40] # every intensity seen in
more than 40 pixels
        if spikes.shape[0] >0:
            lowest_spike = spikes.min()
            highest_spike = spikes.max()
        else:
            lowest_spike = 200
            highest_spike = 255


        offset = self.intensity_range.value
        spectral_offset = 5
        if visualize:
            # display the histogram
            sx,sy = 100,1
            colors =
((0,0,255),(255,0,0),(255,255,0),(255,255,255))
            h,w,chan = img.shape
            hist *= 1./hist.max()  # normalize for display

            for i,h in zip(bins,hist[:,0]):
                c = colors[1]
                cv2.line(img,(w,int(i*sy)),(w-
int(h*sx),int(i*sy)),c)
            cv2.line(img,(w,int(lowest_spike*sy)),(int(w-
.5*sx),int(lowest_spike*sy)),colors[0])
```

```
cv2.line(img,(w,int((lowest_spike+offset)*sy)),(int(w-
.5*sx),int((lowest_spike+offset)*sy)),colors[2])
            cv2.line(img,(w,int((highest_spike)*sy)),(int(w-
.5*sx),int((highest_spike)*sy)),colors[0])
            cv2.line(img,(w,int((highest_spike- spectral_offset
)*sy)),(int(w-.5*sx),int((highest_spike -
spectral_offset)*sy)),colors[3])


        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,
(9,9))

        #open operation to remove eye lashes
        pupil_img = cv2.morphologyEx(pupil_img, cv2.MORPH_OPEN,
kernel)


        # PARAMS = {}
        # blob_detector = cv2.SimpleBlobDetector(**PARAMS)
        # kps =  blob_detector.detect(pupil_img)

        # blur = cv2.GaussianBlur(pupil_img,(5,5),0)
        blur = pupil_img
        # ret3,th3 =
cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
        ret3,th3 =
cv2.threshold(blur,lowest_spike+offset,255,cv2.THRESH_BINARY)
        # ret3,th3 =
cv2.threshold(th3,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
        th3 = cv2.Laplacian(th3,cv2.CV_64F)

        edges = cv2.Canny(pupil_img,
                            self.canny_thresh.value,

self.canny_thresh.value*self.canny_ratio.value,
                            apertureSize=
self.canny_aperture.value)

        r_img[p_roi.lY:p_roi.uY,p_roi.lX:p_roi.uX,1] = th3
        r_img[p_roi.lY:p_roi.uY,p_roi.lX:p_roi.uX,2] = edges
        # for kp in kps:
        #     print kp.pt
        #
cv2.circle(r_img[p_roi.lY:p_roi.uY,p_roi.lX:p_roi.uX],tuple(map(i
nt,kp.pt,)),10,(255,255,255))

        no_result = {}
        no_result['timestamp'] = frame.timestamp
        no_result['norm_pupil'] = None
        return no_result



    def create_atb_bar(self,pos):
        self._bar = atb.Bar(name = "Canny_Pupil_Detector",
label="Pupil_Detector",
            help="pupil detection parameters", color=(50, 50,
50), alpha=100,
            text='light', position=pos,refresh=.3, size=(200,
100))
        #
self._bar.add_var("pupil_intensity_range",self.intensity_range)
```

The script for controlling the mouse is given below:

```
import zmq
import Xlib.display
from pymouse import PyMouse

# Constants/Variables
NULL_REQUIR = 10
current     = 0,0
null_count  = 0

# Mouse & Serial Setup
m = PyMouse()

# Screen Resolution & Thresholds
x_dim, y_dim = m.screen_size()
smooth_x, smooth_y= 0.5, 0.5

# Network Setup
context = zmq.Context()
socket = context.socket(zmq.SUB)
socket.connect("tcp://127.0.0.1:5000")
socket.setsockopt(zmq.SUBSCRIBE, '')

surface_name = "screen"
while True:
    msg = socket.recv()
    items = msg.split("\n")
    msg_type = items.pop(0)
    items = dict([i.split(':') for i in items[:-1] ])
    if msg_type == 'Pupil':
        try:
            my_gaze = items['norm_gaze']

            if my_gaze != "None":
                raw_x,raw_y = map(float,my_gaze[1:-1].split(','))

                # smoothing out the gaze so the mouse has
smoother movement
                smooth_x += 0.5 * (raw_x-smooth_x)
                smooth_y += 0.5 * (raw_y-smooth_y)

                x = smooth_x
                y = smooth_y

                y = 1-y # inverting y so it shows up correctly on
screen
                x *= x_dim
                y *= y_dim

                # PyMouse or MacOS bugfix - can not go to extreme
corners because of hot corners?
                x = min(x_dim-10, max(10,x))
                y = min(y_dim-10, max(10,y))

                # Move mouse to x,y
                if null_count > NULL_REQUIR:
                    m.click(current[0], current[1])
                    null_count = 0
                else:
                    current = x,y
                    m.move(current[0], current[1])
                    null_count = 0

            else:
                print "No Data     (null_count:", null_count, ")"
                null_count += 1

        except KeyError:
            pass
    else:
        # process non gaze position events from plugins here
        pass
```

The script for moving the robotic arm with the users eye is given below:

```
import sys
import zmq
import serial
import Xlib.display
from pymouse import PyMouse

# Constants/Variables
NULL_OUTPUT = "0"
UP_OUTPUT   = "1"
DWN_OUTPUT  = "2"
CLS_OUTPUT  = "3"
OPN_OUTPUT  = "4"
LFT_OUTPUT  = "5"
RGT_OUTPUT  = "6"
current     = "0"
last_sent   = "0"

NULL_REQUIR = 10
null_count  = 0

BAUDRATE = 9600
#ARDUINO_PORT = '/dev/tty.usbmodem1421'
ARDUINO_PORT = '/dev/ttyACM0'

def send_null():
    global last_sent
    if last_sent != NULL_OUTPUT:
        ser.write(NULL_OUTPUT)
        last_sent = NULL_OUTPUT

def graceful_shutdown():
    send_null()
    print "manual"
    sys.exit(0)

def send_serial(mode):
    global last_sent
    if last_sent != mode:
        ser.write(mode)
        last_sent = mode

def update_current(mode):
    global current
    global null_count
    null_count = 0
    if current != mode:
        current = mode

def check_to_send_data(mode):
    if (current == mode) and (null_count > NULL_REQUIR):
        send_serial(mode)
    else:
        update_current(mode)

# Mouse & Serial Setup
m = PyMouse()
ser = serial.Serial(ARDUINO_PORT, BAUDRATE)

# Screen Resolution & Thresholds
x_dim, y_dim = m.screen_size()
smooth_x, smooth_y= 0.5, 0.5
x_thresh_2 = x_dim / 3 * 2
x_thresh_1 = x_dim / 3
y_thresh   = y_dim / 2

# Network Setup
context = zmq.Context()
socket = context.socket(zmq.SUB)
socket.connect("tcp://127.0.0.1:5000")
socket.setsockopt(zmq.SUBSCRIBE, '')

surface_name = "screen"
while True:
    msg = socket.recv()
    items = msg.split("\n")
    msg_type = items.pop(0)
    items = dict([i.split(':') for i in items[:-1] ])
    if msg_type == 'Pupil':
        try:
            my_gaze = items['norm_gaze']

            if my_gaze != "None":
                raw_x,raw_y = map(float,my_gaze[1:-1].split(','))

                # smoothing out the gaze so the mouse has
smoother movement
                smooth_x += 0.5 * (raw_x-smooth_x)
                smooth_y += 0.5 * (raw_y-smooth_y)

                x = smooth_x
                y = smooth_y

                y = 1-y # inverting y so it shows up correctly on
screen
                x *= x_dim
                y *= y_dim

                # PyMouse or MacOS bugfix - can not go to extreme
corners because of hot corners?
                x = min(x_dim-10, max(10,x))
                y = min(y_dim-10, max(10,y))

                # Move mouse to x,y
                m.move(x,y)

                # Check which corner, and send appropriate
command to Arduino
                if y < y_thresh:
# Top
                    if x < x_thresh_1:
# Left
                        print "Top-Left      (x:",x,", y:", y,")"
                        check_to_send_data(UP_OUTPUT)
                    elif x < x_thresh_2:
# Middle
                        print "Top-Center    (x:",x,", y:", y,")"
                        check_to_send_data(RGT_OUTPUT)
                    else:
# Right
                        print "Top-Right     (x:",x,", y:", y,")"
                        check_to_send_data(DWN_OUTPUT)

                else:
# Bottom
                    if x < x_thresh_1:
# Left
                        print "Bottom-Left   (x:",x,", y:", y,")"
                        check_to_send_data(CLS_OUTPUT)
                    elif x < x_thresh_2:
# Middle
                        print "Bottom-Center (x:",x,", y:", y,")"
                        check_to_send_data(LFT_OUTPUT)
                    else:
# Right
                        print "Bottom-Right  (x:",x,", y:", y,")"
                        check_to_send_data(OPN_OUTPUT)

            else:
                print "No Data     (null_count:", null_count, ")"
                null_count += 1
                send_null()
        except KeyboardInterrupt:
            graceful_shutdown()
        except KeyError:
            pass
    else:
        # process non gaze position events from plugins here
        pass
```

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. Connect the Arduino to the subject’s laptop.

6. Connect the the IR LEDs power supply to the user’s laptop.

7. Connect the motor driver’s power port to the user’s laptop.

8. 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.

9. Launch the terminal.

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

11. Launch the main app with “python main.py”.

12. Wear the eye-glasses and adjust the pupil thresholds, brightness, contrast, white balance and the region of interest to make sure the subject’s pupil is detected in the eye window.

APPENDIX D – USER GUIDE

After running through the setup guide, follow the steps below to ensure proper usage of the project:

1. Select the world window and change the calibration method to “Nature features” under “calibration settings”.

2. Position the world camera (on the robotic arm) to look at a static image, having distinct features e.g., the ground.

3. Select the world cam window and press “c” on the keyboard to start the calibration routine

4. The user should not move their head for step 5, and keep as stationary as possible.

5. In the world cam window, select a point at the corner of the screen (click), and stare at the green dot created until it disappears. Repeat for all 4 corners of the screen, and the center of the screen. The more points used, the better the tracking.

6. Once finished, press “c” again to end the calibration routine. Slight head movements are allowed now.

7. The user should check the tracking by looking around on the screen to make sure the pink maker follows the user’s gaze.

8. If satisfactory results are achieved, then launch the server to stream gaze data by pressing “s” when the world window is active. Also start the terminal.

9. Change directory to wherever the scripts given in appendix vii are stored.

10. Run the mouse movement script by opening it through the sudo command, example “sudo python mouse-script-name-here”

11. The pointer follows the user’s gaze, to perform a click, blink for 1.5 seconds so that the counter goes up in the terminal to 10. To ensure best performance, the subject environment should be setup in a way that single clicks substitute for double clicks.

12. To type, the user can use the onscreen keyboard.

13. To use the robotic arm, all other scripts should be closed first (close the running terminal window)

14. Launch the robotic arm control script given in appendix vii in the same manner as in step 10.

15. Horizontally, sequential portions of 1/3 of the screen and vertically 1/2 of the screen are divided in to 6 different regions, each maps to a different arm movement.

16. From left to right on top row, the buttons are i. Move up, ii. Open gripper iii. Move down.

17. From left to right on bottom row, the buttons are i. Move left, ii. Close gripper, iii. Move right.

18. The user can stare in a region, blink for 1.5 seconds, and the robotic arm will start the movement as per the button activated.

19. To stop the arm at a location, blink in another region.

20. The world camera can be used as optical feedback for the user to see where the arm is actually going.

21. To end, close the running script by pressing “ctrl+c” in the script terminal or closing the script window or stopping the server in the world cam window by pressing “s” again on the keyboard or on the onscreen keyboard.

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