# Eye Tracking

Final Report - 4OI6

April 15, 2016

Khalid Asad (1147299)
Danish Gill (1149355)
Bhautik Gandhi (0964720)
Vishal Kharker (1140837)

## Introduction

Amyotrophic Lateral Sclerosis (ALS) is a rapidly progressive, invariably fatal neurological disease that attacks the nerve cells responsible for controlling voluntary muscles such as the ones used for arms, legs and face. Degeneration of motor neurons affect the ability of a person to move their arms, legs and body. ALS most commonly affects individuals between the age of age 20-70 years old. There are 5,000 new cases of ALS diagnosed every year in USA alone. ALS affects individuals worldwide regardless of racial or ethnic boundaries.Unfortunately there aren't many technological developments that help individuals with ALS [3]

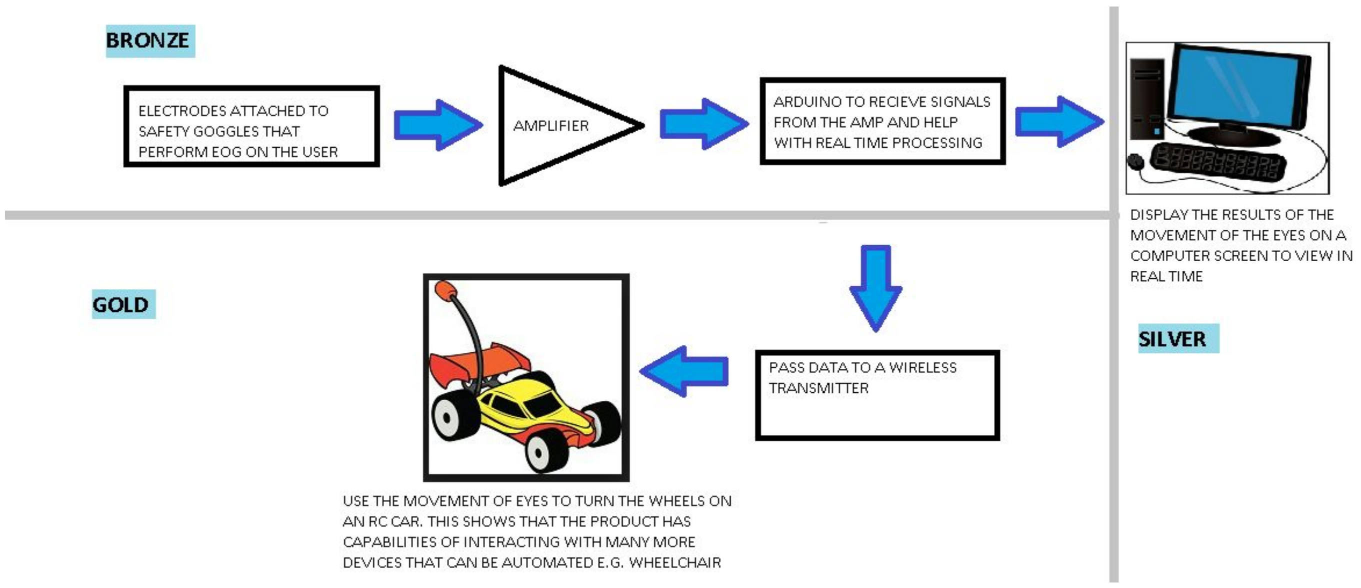
The rapid development and adoption of new and expanding technologies that measure impulses from different parts of the human body has opened doors to a plethora of exciting applications that can make human life more comfortable or easier. The market for input systems of various tasks and intelligent systems is rapidly growing, driving development for innovation and is making initiatives trouble-free.

Newer ways of interacting with technology are being developed everyday and we would like to make a system that can convert our eye movement into input for technological devices. The industry currently uses a mouse and keyboard as the standard means of inputting instructions into a device. Few companies have developed gesture controlled systems, but we would like to go a step future and make a system that can give inputs through your eyes.

We plan to make glasses that the user can wear like any other wearable and measure eye movements using EOG. Although we do not hope to completely replace a mouse or keyboard with this approach, we would like simplify some basic control movements. We feel this could have many applications in gaming and with people with disabilities.

## Project Goals and Planned Approach

The main goal of this project is to have a device that uses Electrooculography (EOG) as a method to receive differential voltage as input for the system [2]. These signals are fed into a microcontroller that helps generate a noise-filtered signal to distinguish the voltage difference. Upon getting a filtered output from the system the signal, can be converted to a digital signal. This can be processed using complex classification techniques into specific eye movements. When this is done, the movements can can be displayed on a User Interface such as a game. Future applications of this project extend a wide range to controlling most devices that are automated. Below figure demonstrates how different parts of the system interact with each other.


BRONZE
ELECTRODES ATTACHED TO SAFETY GOGGLES THAT PERFORM EOG ON THE USER
AMPLIFIER
ARDUINO TO RECIEVE SIGNALS FROM THE AMP AND HELP WITH REAL TIME PROCESSING
DISPLAY THE RESULTS OF THE MOVEMENT OF THE EYES ON A COMPUTER SCREEN TO VIEW IN REAL TIME
SILVER
PASS DATA TO A WIRELESS TRANSMITTER
GOLD
USE THE MOVEMENT OF EYES TO TURN THE WHEELS ON AN RC CAR. THIS SHOWS THAT THE PRODUCT HAS CAPABILITIES OF INTERACTING WITH MANY MORE DEVICES THAT CAN BE AUTOMATED E.G. WHEELCHAIR

## TASKS, SCHEDULING AND IMPLEMENTATION

### October

- Finalize the design of the product
- Place in orders for required hardware
- Finalize coding strategy and approach
- Begin low-mid level coding to verify different approaches and feasibility

### November

- Continuation of software development
- Upon delivery of hardware, begin initial system integration
- Demonstrate for milestone A

### December

- Rework on suggestion from milestone A demonstration
- Continuation of software development towards Silver level
- Improve the efficiency of hardware to perform better for milestone B

### January

- Continuation of software development
- Testing hardware-software integration with new code
- Demonstration for milestone B

### February

- Rework on suggestion from milestone A demonstration
- Continuation of software development towards Silver level
- Demonstration for milestone C

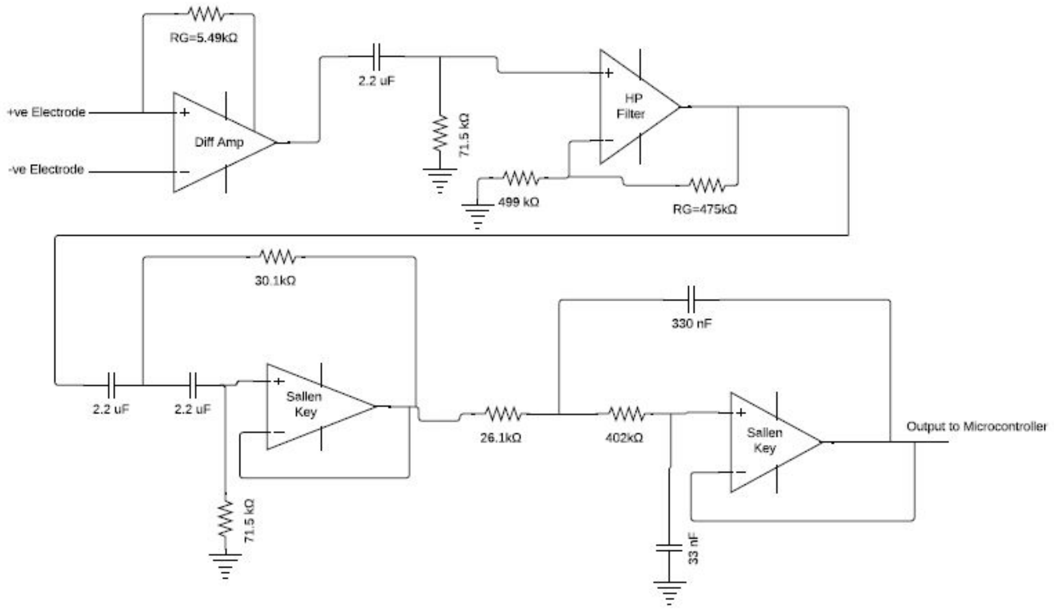
### March - April

- Leaving this time frame for making any changes to the overall product
- Trying better methods after receiving feedback from previous demonstration

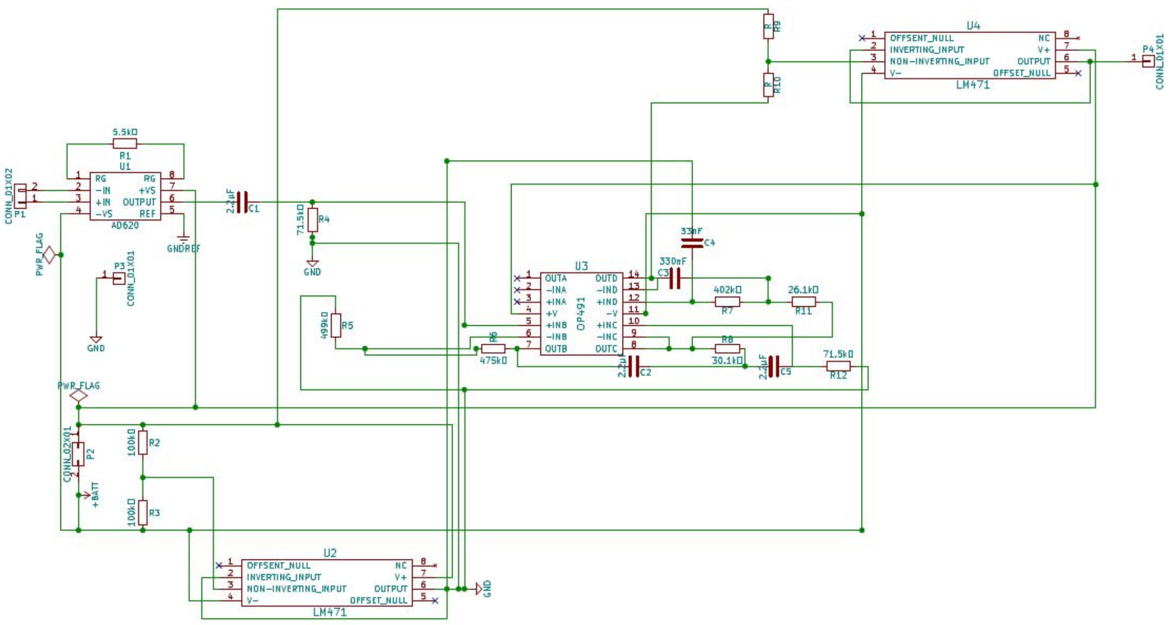
- Integrate the entire system according to Gold level

**Actual approach (design, modeling/simulation, implementation)**

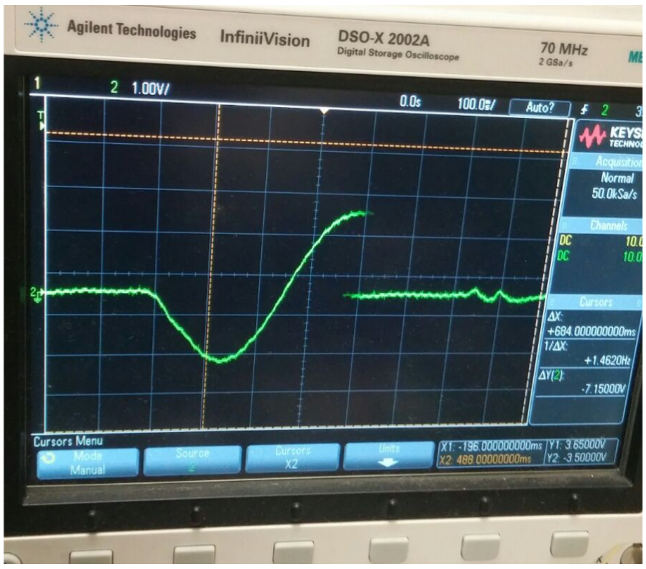
We started our design by figuring out the flow of information and the different parts that were needed. We started by creating a set of filters to get a clean signal between 0 to 30 Hz, where the literature told us to expect the EOG signals. We designed the filters as such in the diagram below. The differential amplifier is used to amplify the weak signal with a gain of 1000.

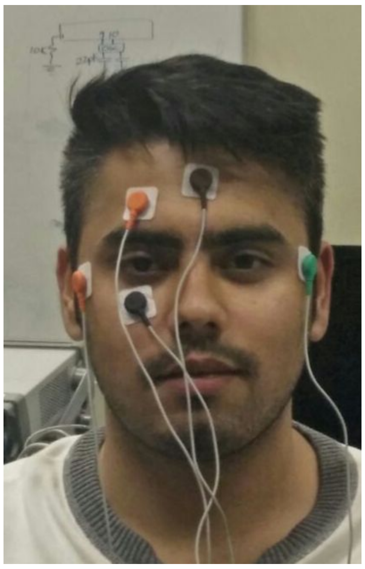


The final circuit was implemented in KiCAD as a PCB design, shown below.

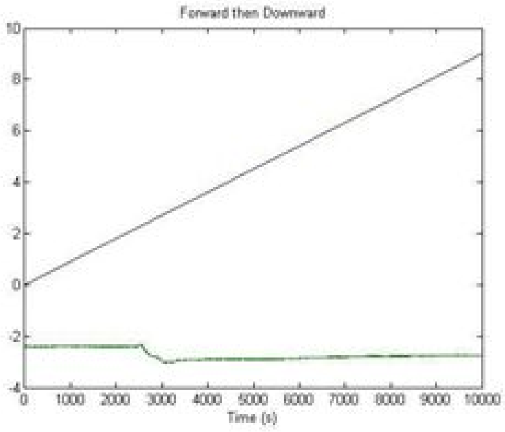


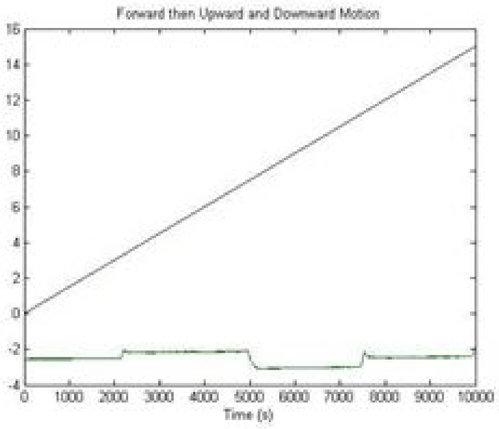
**Circuit Output**: This is the output we got from the filters and amplifiers in the osciloscope with the setup shown on the right.

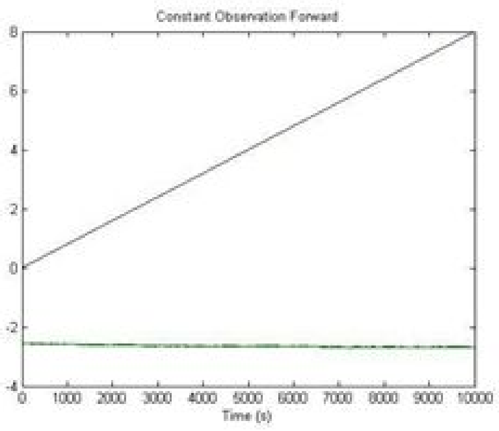


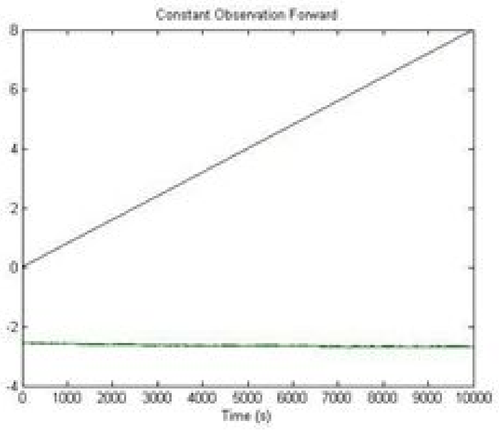


Output from Labview processed data shown below, just for presentation and proof of concept from our experiments, but it wasn't actually used for the rest of the project.

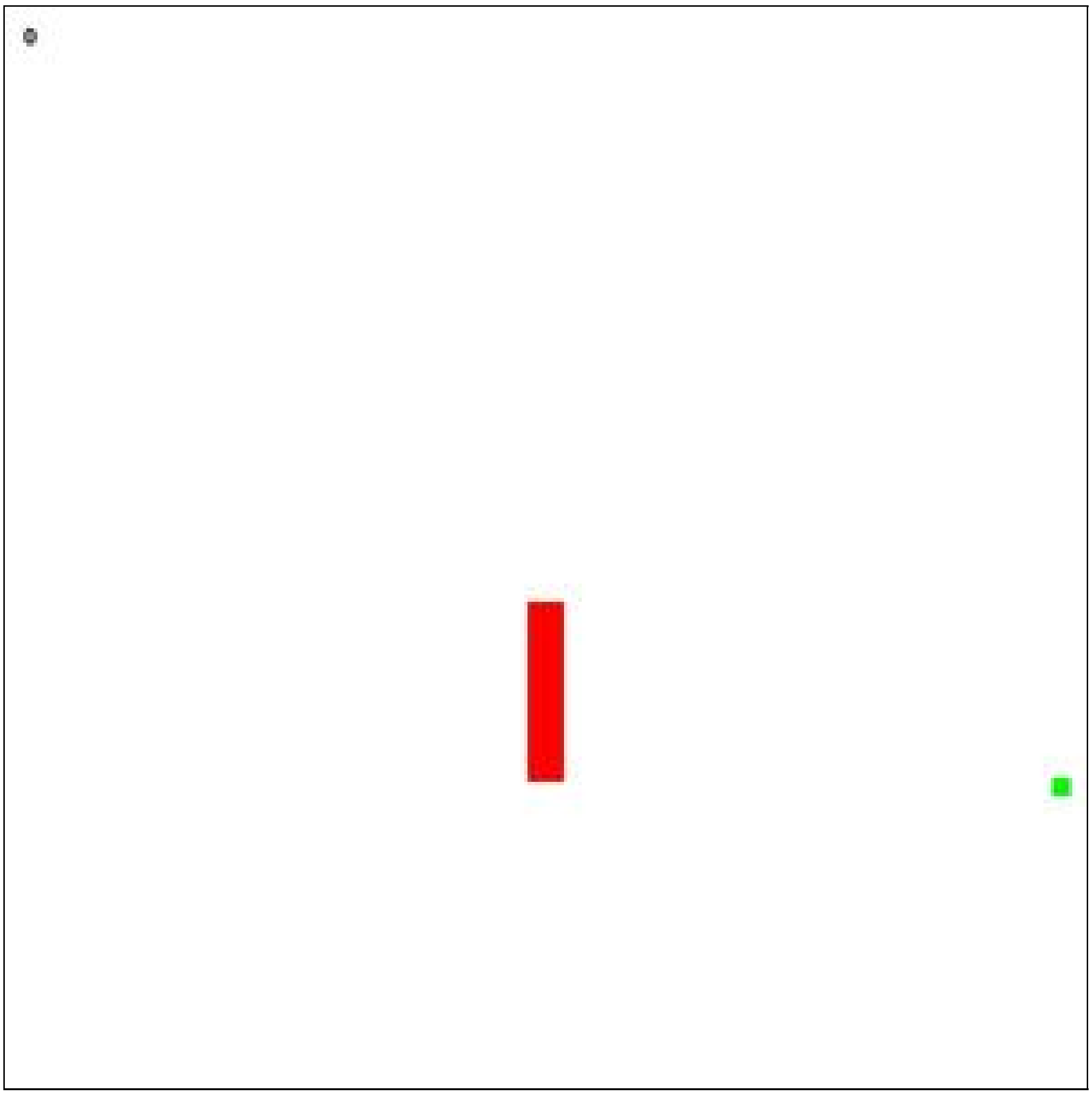








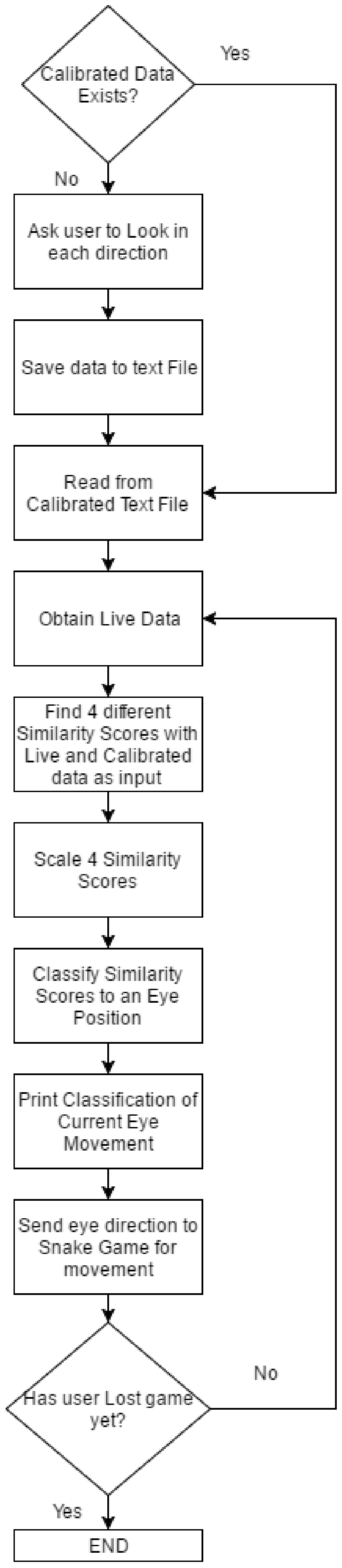
## SNAKE GAME



The objective of the Snake game is to obtain the green pellets and as one does so, the length of the snake increases. This snake game does not allow for the snake to move beyond the borders and start on the opposing side. Hitting the borders means death. The game is open source and uses the PyGame library [1]. We modified the direction events to only happen when and eye movement has been classified using our classification program. Before the snake game starts, the user has the option of using calibrated data, or calibrating new data. The latter option will give a better success rate in the movements of the snake.

An implementation of the Tetris game was also under process, however we ran out of time to finish coding that from scratch. We could not find a decent open source Tetris game.

**Flowchart Diagram and Logical Approach:**


Calibrated Data Exists?
Yes
No
Ask user to Look in each direction
Save data to text File
Read from Calibrated Text File
Obtain Live Data
Find 4 different Similarity Scores with Live and Calibrated data as input
Scale 4 Similarity Scores
Classify Similarity Scores to an Eye Position
Print Classification of Current Eye Movement
Send eye direction to Snake Game for movement
Has user Lost game yet?
No
Yes
END

## <u>Critical problems solved</u>

Much was learned through the development of this project as we encountered and resolved several major problems. The first we encountered was during Milestone B after our initial lab tests when we attempted to get the signal read into the computer. What seemed simple turned out to be more difficult. After doing some tests to rule out our hardware prototype, it became apparent that the issue was with the Arduino interface, where we discovered multiple issues: one was the sampling BAUD rate which had to be adjusted as well as the board ground pin had to be connected properly. Once that was overcome we had clear signal reading into the computer.

Next, we split the project into two streams so that we can work more efficiently; one was to update the hardware to make it portable, compact so that it can work outside of the lab and look presentable, and the other was the algorithm development for categorization and the application for demonstration.

On the hardware side, we ran into the issue of having to provide rail-to-rail power (+4.5v to -4.5V) to the all the amplifiers with a single source. After some research and checking many designs, we settled on a simple design of virtual ground using an opamp available to us as it was decided that we didn't require anything complicated for our application and demonstration. Using a 9 volt battery, using a voltage divider and an opamp, a virtual ground of 4.5V was created for grounding, which gave us +4.5V and -4.5V output to use for the amplifiers.

Once that was solved, we designed a PCB for the final implementation. However we were having problems with our PCB's not working properly which made them difficult to work with. The issues arised from the soldering being difficult for some of our components, but it was a difficult problem to solve without spending more money which we did not have at the time. A positive outcome was that we planned backup by creating our hardware on two bread boards in case things go wrong, which turned out to be very fruitful as the PCB's stopped working the night before and couldn't be fixed in time.

Another issue we had was that while the horizontal signal was great, our vertical movement signal was very susceptible to blinks and also horizontal movements. But also that the signals depended greatly on the how well the electrodes were placed on the user and how clean the contact was on the face. We used this information when designing the goggles for the user; we made it so that the user can ensure proper placement and apply the right pressure to keep good contact, and also used alcohol swabs to clean the surface before placing the electrodes. We modified safety goggles as we found it to be a good solution and it was also aesthetically pleasing.

One of the hardest problems we faced was with classification algorithm for our signals. This appeared to be theoretically easy but turned out to be far more difficult than anticipated. We experimented with various techniques for found in literature, such as Z transformation/Z-thresholding, moving average smoothing and Support Vector Machines. However, in the end we decided to use computationally less intensive methods such as taxicab metric, euclidean distance, chebyshev distance, pearson coefficient (see Appendix A). This method worked sufficiently well for us, but we believe other methods would produce better results.

On the application side, it was fairly straight forward. We decided that the best way to display practical application of our product was to get the audience as involved as possible. One way we thought this would possible was to integrate a simple open source snake game into our code. Since we were only able to identify directional movements, this game was the best application of the project.

## **Conclusion (final result achieved)**

ALS is a severe disease however it is crucial for patients to be able to communicate freely. ALS patients require 80% accuracy for the interface to be usable. The applications of this project vary from communicating via on-screen UI to be able to send wireless signals and control wheel chair movements. Future improvements can be made by further reducing noise with PCBs and filters, and implementing better calibration through advanced algorithms such as machine learning. Our results show the efficacy of the EOG method for computer interface. We were able to achieve reasonable accuracy of around 60 to 70% depending the calibration process and how well the electrodes were placed. The users were able to control the snake in the game and enjoy the process, which we consider a success for our project.

## **Future plan (any potential market value or specific usage?)**

The usage of  EOG technology demonstrated in this project has various future applications. One of the major uses of this technology would be to help patients with ALS have access to a method of communication with their loved ones. The existing technologies are comparatively more expensive to the one featured in this project. In this project there was a lot of weight put on the usage of hardware as opposed to expensive cameras and software. This allows for a wider accessibility of this technology to individuals who are not as financially stable.

EOG can also be used for applications such as moving a wheelchair. There are several projects that are trying to research on collecting information using brain signals. As EOG signals are easier to record and calibrate, this technology can be readily available to commercialize in a short period of time.

There are several rehab programs that focus on strategically training ALS patients during the early phases of the disease to maximize on patient's independence in mobility and daily activities. Providing such organizations with EOG technology can help improve process of rehab and a broader spectrum of independence can be guaranteed to ALS patients.

## <u>REFERENCES</u>

**[1]** Samuel, Backman. "Snake in 35 Lines." PYGAME. July 25, 2008. http://pygame.org/project-Snake in 35 lines-818-.html.

**[2]** Morimoto, Carlos H., and Marcio R.m. Mimica. "Eye Gaze Tracking Techniques for Interactive Applications." Computer Vision and Image Understanding 98.1 (2005): 4-24. Web.

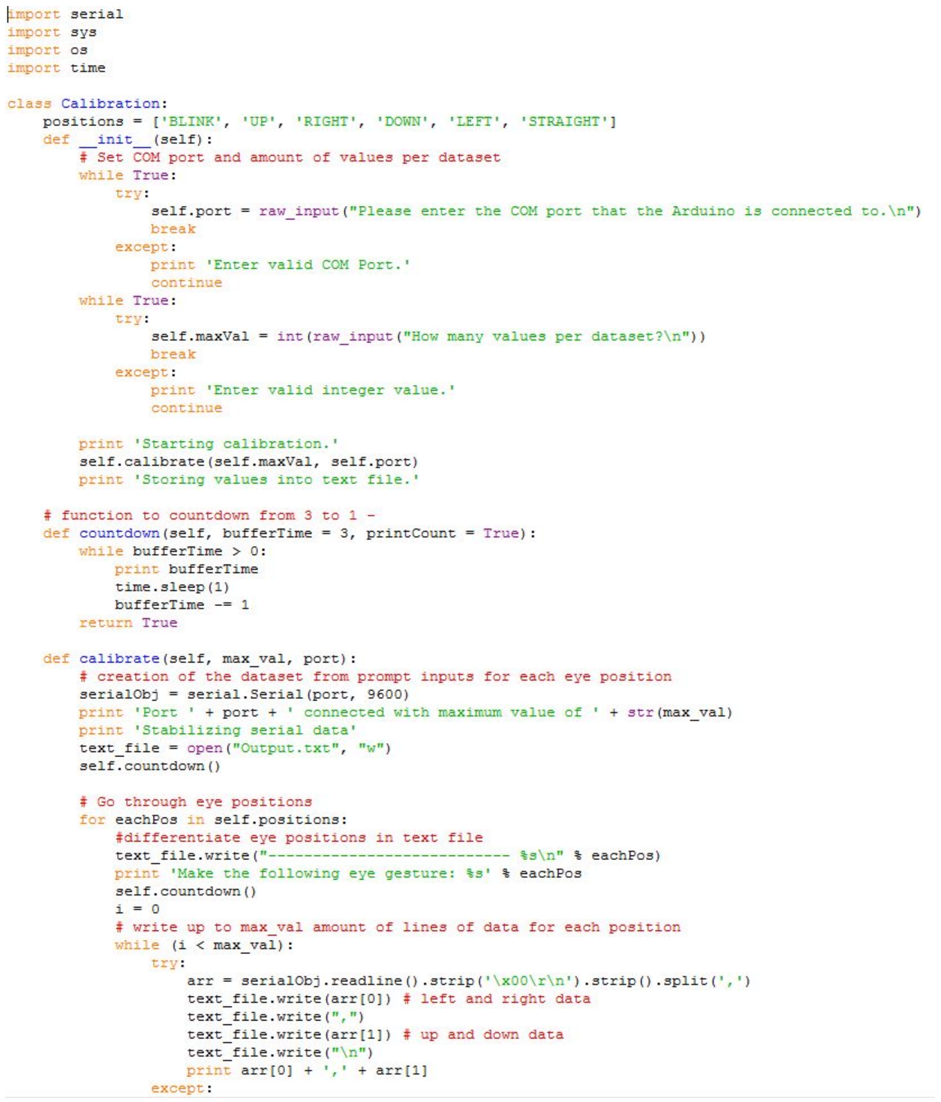
**[3]** Mcnaughton, David, Janice Light, and Linda Groszyk. ""Don't Give Up": Employment Experiences of Individuals with Amyotrophic Lateral Sclerosis Who Use Augmentative and Alternative Communication." Augmentative and Alternative Communication TAAC Augmentative & Alternative Comm. 17.3 (2001): 179-95. Web.

**[4]** Sharma, Anjana, and Pawanesh Abrol. "Eye Gaze Techniques for Human Computer Interaction: A Research Survey." International Journal of Computer Applications IJCA 71.9 (2013): 18-25.

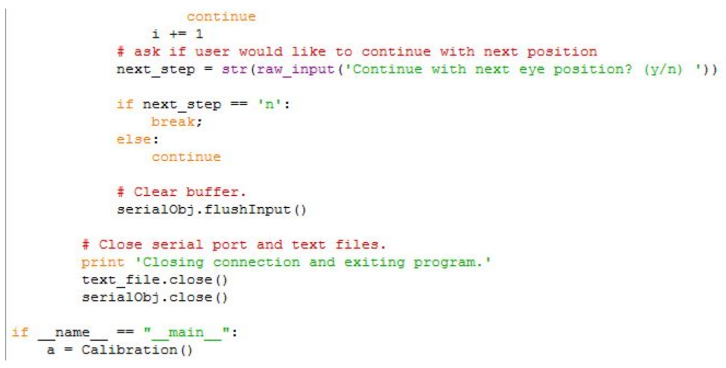
**[5]** Zhu, Zhiwei, and Qiang Ji. "Novel Eye Gaze Tracking Techniques Under Natural Head Movement." IEEE Transactions on Biomedical Engineering IEEE Trans. Biomed. Eng. 54.12 (2007): 2246-260.

## Appendix A

### Calibration.py - *Handles calibration per position and saves to a text file*



```
import serial
import sys
import os
import time

class Calibration:
    positions = ['BLINK', 'UP', 'RIGHT', 'DOWN', 'LEFT', 'STRAIGHT']
    def __init__(self):
        # Set COM port and amount of values per dataset
        while True:
            try:
                self.port = raw_input("Please enter the COM port that the Arduino is connected to.\n")
                break
            except:
                print 'Enter valid COM Port.'
                continue
        while True:
            try:
                self.maxVal = int(raw_input("How many values per dataset?\n"))
                break
            except:
                print 'Enter valid integer value.'
                continue

        print 'Starting calibration.'
        self.calibrate(self.maxVal, self.port)
        print 'Storing values into text file.'

    # function to countdown from 3 to 1 -
    def countdown(self, bufferTime = 3, printCount = True):
        while bufferTime > 0:
            print bufferTime
            time.sleep(1)
            bufferTime -= 1
        return True

    def calibrate(self, max_val, port):
        # creation of the dataset from prompt inputs for each eye position
        serialObj = serial.Serial(port, 9600)
        print 'Port ' + port + ' connected with maximum value of ' + str(max_val)
        print 'Stabilizing serial data'
        text_file = open("Output.txt", "w")
        self.countdown()

        # Go through eye positions
        for eachPos in self.positions:
            #differentiate eye positions in text file
            text_file.write("--------------------------- %s\n" % eachPos)
            print 'Make the following eye gesture: %s' % eachPos
            self.countdown()
            i = 0
            # write up to max_val amount of lines of data for each position
            while (i < max_val):
                try:
                    arr = serialObj.readline().strip('\x00\r\n').strip().split(',')
                    text_file.write(arr[0]) # left and right data
                    text_file.write(",")
                    text_file.write(arr[1]) # up and down data
                    text_file.write("\n")
                    print arr[0] + ',' + arr[1]
                except:
```



```
                    continue
                i += 1
            # ask if user would like to continue with next position
            next_step = str(raw_input('Continue with next eye position? (y/n) '))

            if next_step == 'n':
                break;
            else:
                continue

            # Clear buffer.
            serialObj.flushInput()

        # Close serial port and text files.
        print 'Closing connection and exiting program.'
        text_file.close()
        serialObj.close()

if __name__ == "__main__":
    a = Calibration()
```

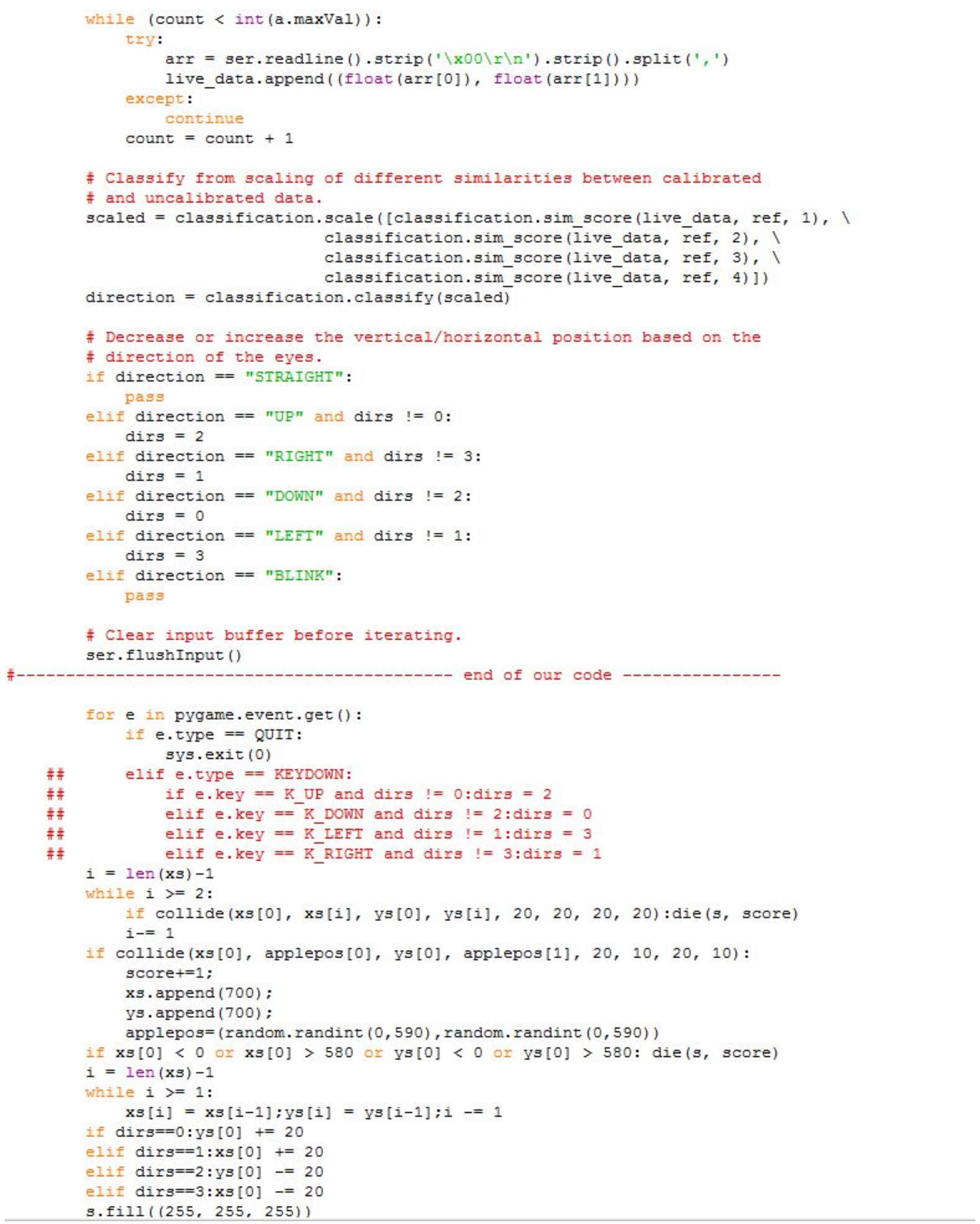
**Classification.py** - *Uses Calibration output to classify eye movements to mouse movements*

```python
#!/usr/bin/env python
import win32api, win32con
from time import sleep
import serial
import calibration
import os
from math import sqrt

# Define a click using win32api.
def click(x,y):
    win32api.SetCursorPos((x,y))
    win32api.mouse_event(win32con.MOUSEEVENTF_LEFTDOWN,x,y,0,0)
    win32api.mouse_event(win32con.MOUSEEVENTF_LEFTUP,x,y,0,0)

# Scale a bunch of similarity scores.
def scale(big_list):
    scaled = []
    for x in range(len(big_list[0])):
        scaled.append(0)
        for y in range(len(big_list)):
            scaled[x] += big_list[y][x]
        scaled[x] /= len(big_list)
    return scaled

# Use similarity scores to find which score is closest in similarity.
def classify(scaled_values):
    pos = {1 : 'BLINK',
           2 : 'UP',
           4 : 'RIGHT',
           6 : 'DOWN',
           8 : 'LEFT',
           10 : 'STRAIGHT'}
    posVals = pos.values()
    classified = {}
    # Assign eye positions to each scaled value.
    for a in range(len(scaled_values)):
        classified[scaled_values[a]] = posVals[a]
    scaled_values.sort()
    # Return eye position that has the highest similarity score.
    return classified[scaled_values[-1]]

# Take live data from Arduino and calibrated data, and use that
# to calculate similarity scores based on the type of calculation
# that the user wants to use.
def sim_score(live_data, calib_data, similarity):
    simScore = []
    filX = 0
    filY = 0
    # Iterate through sets of data.
    for calibSet in range(len(calib_data)):
        refLength=len(calib_data[calibSet])
        simScore.append(0)
        sum1=0
        sum2=0
        sum1_squared=0
        sum2_squared=0
        totalSum=0


        # Apply similarity metrics.
        for pair in range(refLength):
            if similarity==1: #Taxicab metric.
                simScore[calibSet]+=1/float(1+(abs(live_data[pair][0]-calib_data[calibSet][pair][0])+\
                                      abs(live_data[pair][1]-calib_data[calibSet][pair][1])))
            elif similarity==2: #Euclidean distance.
                simScore[calibSet]+=1/float(1+sqrt(pow(live_data[pair][0]-calib_data[calibSet][pair][0],2)\
                                      +pow(live_data[pair][1]-calib_data[calibSet][pair][1],2)))
            elif similarity==3: #Chebyshev distance.
                filX+=abs(live_data[pair][0]-calib_data[calibSet][pair][0])
                filY+=abs(live_data[pair][1]-calib_data[calibSet][pair][1])
            elif similarity==4: #Pearson coefficient.
                sum1+=live_data[pair][1]
                sum2+=calib_data[calibSet][pair][1]
                sum1_squared+=pow(live_data[pair][1],2)
                sum2_squared+=pow(calib_data[calibSet][pair][1],2)
                totalSum+=live_data[pair][1]*calib_data[calibSet][pair][1]
        if similarity==3:
            simScore[calibSet]=1/float(1+max(filX,filY))
            filX=0
            filY=0
        if similarity==4:
            top=totalSum-float(sum1*sum2/refLength)
            bottom=sqrt((sum1_squared-float(pow(sum1,2)/refLength))*\
                         (sum2_squared-float(pow(sum2,2)/refLength)))
            if bottom==0 or top==0:
                simScore[calibSet]=0
            else:
                simScore[calibSet]=float(top)/float(bottom)
    return simScore
```

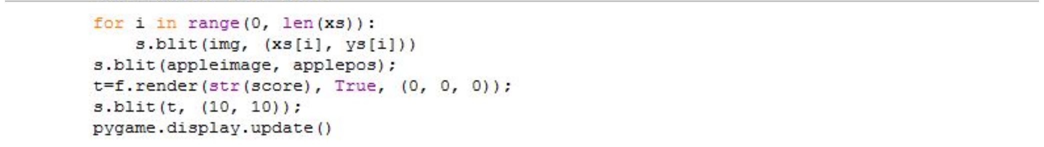
```
if __name__ == "__main__":
    # Create calibration object.
    a = calibration.Calibration()
    ser = serial.Serial(a.port, 9600) # Open serial port.

    # Open calibration file and save lines to ref.
    count = 0
    ref = []
    with open('Output.txt', 'rb') as ins:
        for lines in ins:
            if lines.startswith("-"):
                print lines.split()[1]
            else:
                ref.append((float(lines.split(',')[0]), float(lines.split(',')[1])))
            count=count+1
    ins.close()

    # Save live data to array.
    while True:
        live_data = []
        count = 0

        while (count < int(a.maxVal)):
            try:
                arr = ser.readline().strip('\x00\r\n').strip().split(',')
                live_data.append((float(arr[0]), float(arr[1])))
            except:
                continue
            count = count + 1
        # Classify from similarity scores based on live and calibrated data.
        scaled = scale([sim_score(live_data, ref, 1), \
                        sim_score(live_data, ref, 2), \
                        sim_score(live_data, ref, 3), \
                        sim_score(live_data, ref, 4)])
        direction = classify(scaled)
        print direction

        # Move cursor in direction.
        if direction == "STRAIGHT":
            pass
        elif direction == "UP":
            win32api.SetCursorPos((1366/2,0))
        elif direction == "RIGHT":
            win32api.SetCursorPos((1366,768/2))
        elif direction == "DOWN":
            win32api.SetCursorPos((1366/2,768))
        elif direction == "LEFT":
            win32api.SetCursorPos((0,768/2))
        elif direction == "BLINK":
            click(1366/2,768/2)

        # Clear buffer.
        ser.flushInput()
```

**Snake.py** - *Uses Classification to convert eye movements to in game movements (open source game using PyGame with only modifications to Left, Right, Up, Down events)*

```
import pygame, random, sys
from pygame.locals import *
from time import sleep
import classification
import calibration
import serial

# collision function
def collide(x1, x2, y1, y2, w1, w2, h1, h2):
    if x1+w1>x2 and x1<x2+w2 and y1+h1>y2 and y1<y2+h2:return True
    else:return False

# user death function
def die(screen, score):
    f=pygame.font.SysFont('Arial', 30);
    t=f.render('Your score was: '+str(score), True, (0, 0, 0));
    screen.blit(t, (10, 270));
    pygame.display.update();
    pygame.time.wait(2000);
    sys.exit(0)

# main
if __name__ == "__main__":
    xs = [290, 290, 290, 290, 290];ys = [290, 270, 250, 230, 210];
    dirs = 0;
    score = 0;applepos = (random.randint(0, 590), random.randint(0, 590));
    pygame.init();
    s=pygame.display.set_mode((600, 600));
    pygame.display.set_caption('Snake');
    appleimage = pygame.Surface((10, 10));
    appleimage.fill((0, 255, 0));
    img = pygame.Surface((20, 20));
    img.fill((255, 0, 0));
    f = pygame.font.SysFont('Arial', 20);
    clock = pygame.time.Clock()

#-------------------------------------------- start of our code ---------------
    # Create a Calibration object from calibration.py.
    a = calibration.Calibration()
    # Open up serial port.
    ser = serial.Serial('COM11', 9600)

    # read from calibrated text file and save into ref array
    count = 0
    ref = []
    with open('Output.txt', 'rb') as ins:
        for lines in ins:
            if lines.startswith("-"):
                print lines.split()[1]
            else:
                ref.append((float(lines.split(',')[0]), float(lines.split(',')[1])))
            count=count+1
    ins.close()

    while True:
        # use clock ticks of 10
        clock.tick(10)
        live_data = []
        count = 0

        # save live serial data from arduino into a new array
```



```
        while (count < int(a.maxVal)):
            try:
                arr = ser.readline().strip('\x00\r\n').strip().split(',')
                live_data.append((float(arr[0]), float(arr[1])))
            except:
                continue
            count = count + 1

        # Classify from scaling of different similarities between calibrated
        # and uncalibrated data.
        scaled = classification.scale([classification.sim_score(live_data, ref, 1), \
                                classification.sim_score(live_data, ref, 2), \
                                classification.sim_score(live_data, ref, 3), \
                                classification.sim_score(live_data, ref, 4)])
        direction = classification.classify(scaled)

        # Decrease or increase the vertical/horizontal position based on the
        # direction of the eyes.
        if direction == "STRAIGHT":
            pass
        elif direction == "UP" and dirs != 0:
            dirs = 2
        elif direction == "RIGHT" and dirs != 3:
            dirs = 1
        elif direction == "DOWN" and dirs != 2:
            dirs = 0
        elif direction == "LEFT" and dirs != 1:
            dirs = 3
        elif direction == "BLINK":
            pass

        # Clear input buffer before iterating.
        ser.flushInput()
#------------------------------------------- end of our code ----------------

        for e in pygame.event.get():
            if e.type == QUIT:
                sys.exit(0)
    ##      elif e.type == KEYDOWN:
    ##          if e.key == K_UP and dirs != 0:dirs = 2
    ##          elif e.key == K_DOWN and dirs != 2:dirs = 0
    ##          elif e.key == K_LEFT and dirs != 1:dirs = 3
    ##          elif e.key == K_RIGHT and dirs != 3:dirs = 1
        i = len(xs)-1
        while i >= 2:
            if collide(xs[0], xs[i], ys[0], ys[i], 20, 20, 20, 20):die(s, score)
            i-= 1
        if collide(xs[0], applepos[0], ys[0], applepos[1], 20, 10, 20, 10):
            score+=1;
            xs.append(700);
            ys.append(700);
            applepos=(random.randint(0,590),random.randint(0,590))
        if xs[0] < 0 or xs[0] > 580 or ys[0] < 0 or ys[0] > 580: die(s, score)
        i = len(xs)-1
        while i >= 1:
            xs[i] = xs[i-1];ys[i] = ys[i-1];i -= 1
        if dirs==0:ys[0] += 20
        elif dirs==1:xs[0] += 20
        elif dirs==2:ys[0] -= 20
        elif dirs==3:xs[0] -= 20
        s.fill((255, 255, 255))
```



```
        for i in range(0, len(xs)):
            s.blit(img, (xs[i], ys[i]))
        s.blit(appleimage, applepos);
        t=f.render(str(score), True, (0, 0, 0));
        s.blit(t, (10, 10));
        pygame.display.update()
```

**Extra Libraries Installed:**

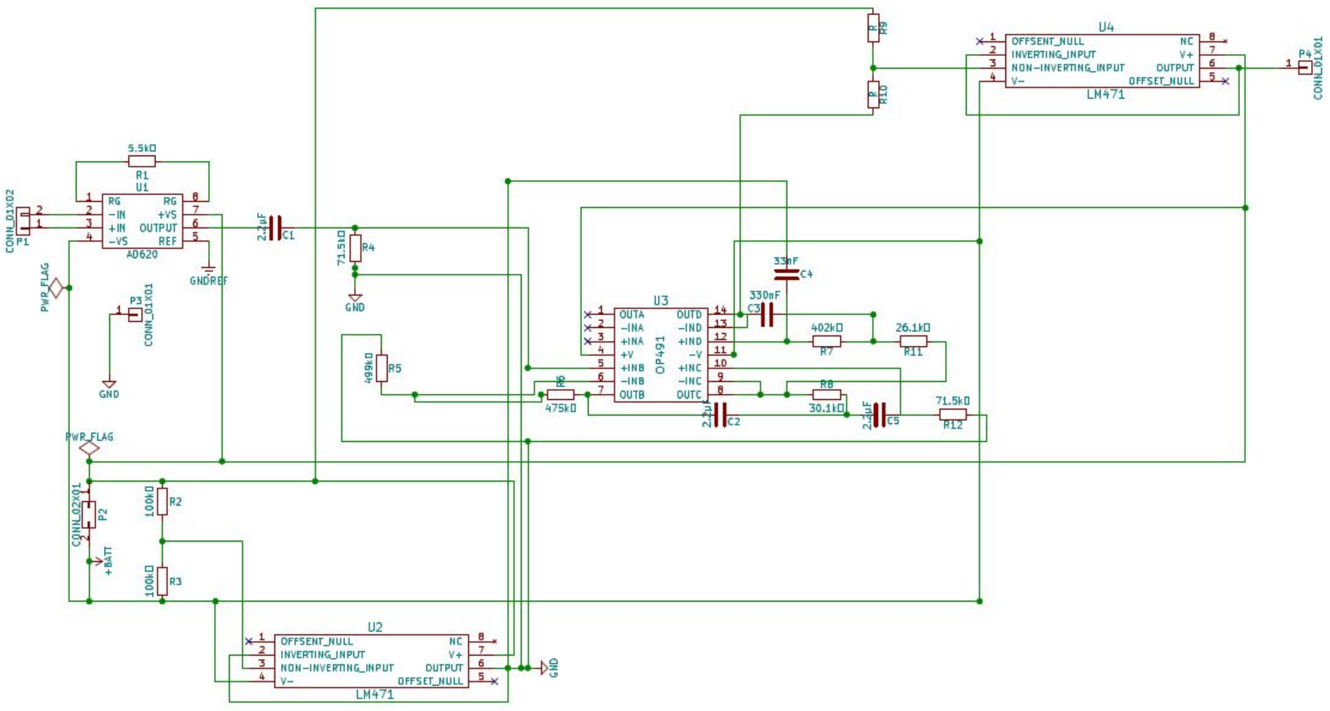
*Win32api*

*Serial*

*PyGame*

**APPENDIX B**

Circuit Schematic for PCB Design



U1
AD620
R1
5.5kΩ
C1
2.2µF
R4
71.5kΩ
GND
GNDREF
P1
CONN_01X02
P2
CONN_02X01
P3
CONN_01X01
P4
CONN_01X01
PWR_FLAG
+BATT
R2
100kΩ
R3
100kΩ
U2
LM471
U3
OP491
U4
LM471
R5
499kΩ
R6
475kΩ
R7
402kΩ
R8
30.1kΩ
R9
R10
R11
26.1kΩ
R12
71.5kΩ
C2
2.2µF
C3
330nF
C4
33nF
C5
2.2µF
OFFSENT_NULL
INVERTING_INPUT
NON-INVERTING_INPUT
V-
NC
V+
OUTPUT
OFFSET_NULL