Interacting with the Human Eye: Gaze Vector Shape Based Recognition and the Design of an Improved Episcleral Venomanometer

by

Trevor L. Craig

A Thesis

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Master of Science

Major: Mechanical Engineering & Applied Mechanics

Under the Supervision of Professor Carl Nelson

Lincoln, Nebraska

August, 2017

Interacting with the Human Eye: Gaze Vector Shape Based Recognition and the Design of an Improved Episcleral Venomanometer

Trevor Lynn Craig, M.S.

University of Nebraska, 2017

Advisor: Carl Nelson

The sense of sight is one of the main outlets to how we interact with the world. Using eye tracking methods, this sensory input channel may also be used as an output channel to provide commands for robots to follow. These gaze-commanded robots could then be used to assist severely mobility-limited individuals in the home or similar environments. This thesis explores the use of visually drawn shapes as the input for robot commands. These commands were recorded using low-cost gaze tracking hardware (Gazepoint GP3 Eye Tracker). The data were then processed using a custom algorithm in MATLAB to detect commands to be passed to two different mobile robots. The ability to use stochastic analysis for path prediction is also explored. Using the techniques and procedures given in this paper, people with limited mobility will be able to input shape commands to have robots react as personal assistants. This research is extensible to gaze-based human-machine interfaces in general for a variety of applications.

In order to better under understand the eye, improvements and retro-fitting of an episcleral venomanometer were conducted. A portable video enabled venomanometer was created to observe vein occlusion and its correlating pressure. This was then

improved upon through design iteration. The current issue with measuring is receiving accurate and precise readings of eye parameters due to variations in user technique. Designing improved medical devices for collection of information on ocular health and function will provide better understanding of related medical conditions, including but not limited to, glaucomatous damage.

Contents

CHAPTER 1: Introduction 8
CHAPTER 2: APPROACH 14
i. BACKGROUND: 14
ii. STATE MACHINE CONCEPT: 15
CHAPTER 3: METHODS 17
i. PROCEDURE: 17
ii. SHAPERECOGN (HIGH-LEVEL ALGORITHM): 18
iii. CENTER FINDING TECHNIQUES: 18
iv. FILTERING: 19
v. BEST FIT RADIUS ALGORITHM 21
vi. CORNER FINDING ALGORITHM 22
vii. HOUGH TRANSFORM METHODS 23
a. PREPROCESSING: 24
b. SHAPE DETECTION ALGORITHM: 24
viii. FINAL RESULT: 27
CHAPTER 4: RESULTS: 28
i. BEST FIT RADIUS ALGORITHM: 29
ii. CORNER FINDING ALGORITHM: 31
iii. HOUGH METHODS: 32
a. AREA OF HOUGH TRANSFORM METRIC: 32
b. MEAN OF HOUGH TRANSFORM METRIC: 33
c. SLOPE OF HOUGH TRANSFORM METRIC: 34
d. MEDIAN OF HOUGH TRANSFORM METRIC: 36
iv. RESULTS SUMMARY: 37
v. HUMAN TESTING: 37
vi. TESTING WITH NAO ROBOT: 41
vii. TESTING WITH LOCATION DEPENDENT COMMANDS: 41
CHAPTER FIVE: STOCHASTIC ANALYSIS 44
i. FORECASTING: 51

ii. FREQUENCY, DAMPING RATIO, POWER (VARIANCE) CONTRIBUTION: 54
iii. DISCUSSION: 55
CHAPTER SIX: EPISCLERAL VENOMANOMETER 58
i. INTRODUCTION: 58
ii. PROBLEM DESCRIPTION: 60
iii. SOLUTION: 62
a. ITERATION 1: 62
b. ITERATION 2: 66
c. ITERATION 3: 71
d. ITERATION 4: 73
e. ITERATION 5: 77
CHAPTER SEVEN: CONCLUSIONS 81
i. CONCLUSION OF GAZE VECTOR SHAPE BASED RECOGNITION 81
ii. CONCLUSION OF DESIGN OF AN IMPROVED EPISCLERAL VENOMANOMETER: 82
ACKNOWLEDGEMENT: 84
Works Cited 85
APPENDIX 1: 92
Stocachstic Equations: 92
Circle Y: 96
Square X: 99
Square Y: 103
Triangle X: 106
Triangle Y: 111
Summary: 114
HOUGH TRANSFORMS: 115
Circle: 115
Square: 119
Triangle: 122
Summarized Results: 124
Appendix 2: 126

Iteration 1: 126
Iteration 2: 129
Iteration 3: 131
Iteration 4: 133
Iteration 5: 135
APPENDIX 3: 139
ShapeRecognGuiTest.m: 140
Settings.ini: 147
Inifile.m: 152
GazePointApi: 171
FindtheCenter.m: 174
DataFilter.m: 175
ShapeRecognFnc.m: 176
ShapeScanner.m: 184
CornerDetection.m: 185
ShapeDetect.m: 187
HoughAssist.m: 194
DISPLAYXY.m: 195
ProgramLauncer.m: 196
SaveToFiles.m: 197
RandomName.m: 198
RandomTestOrder.m: 199

List of Figures:

Figure 1: Nintendo Hands Free Controller 9
Figure 2. Shape guide with hypothetical command overlay 15
Figure 3. General data flow for shape detection 18
Figure 4. Smoothing Example 20
Figure 5. Saccade Example 21
Figure 6. Smooth Shape Example 21
Figure 7. Drawn a shape with bounds for best-fit radius algorithm 22
Figure 8. Corner Locations for the Circle 23
Figure 9: Enclosed Shapes 25
Figure 10. Unique Graphs for Respective Shapes from Fig. 8 27
Figure 11. Example of Slope and Peak Location for Square 35
Figure 12. User Interface 38
Figure 13. The NAO robot is responding to gaze-based commands 41
Figure 14. Robot at Position A 42
Figure 15. Robot with 3 LEDs 43
Figure 16: ARMA (2,1) Model Circle X Coordinate 48
Figure 17: ARMA (2,1) Model Circle X Coordinate Future Predictions 49
Figure 18. Episcleral Venomanometer ..... **Error! Bookmark not defined.**
Figure 19. Espiceral Venous Pressure Pictures 59
Figure 20. Basic Anatomy of the Human Eye 61
Figure 21: Raspberry Pi Zero 63
Figure 22: Adafruit PiTFT 2.8" Touchscreen 64
Figure 23. Iteration 1 65
Figure 24: Lipped and Groove Back Plate 66
Figure 25: Electronics Bay 67
Figure 26: Iteration 2 68
Figure 27. Potentiometer Arm 69
Figure 28: Loc-Line Tubing 70
Figure 29. Carson MicroBrite Pocket Microsocpe 71
Figure 30. Iteration 3 72
Figure 31. JamStand and Gooseneck 74
Figure 32. Iteration 4 75
Figure 33. Iteration 5 79

CHAPTER 1: Introduction

Assistive robots have been given significant research attention in recent decades, especially in the home care domain for facilitating and enhancing the daily living, of the disabled and elderly. Extensive effort has been put forward to enhance robot capability in performing various tasks like cooking [1], [2], doing laundry [3], [4], object retrieval [5], [6], performing bed baths [7], assisting walking [8], [9], etc. However, with continuously increased functionality and complexity of the robotic system [10], [11], [12], managing these robots becomes inevitably more complex [13], which is burdensome or even infeasible with traditional control interfaces consisting of buttons, switches, knobs, touch screens, motion control, and joysticks. Moreover, new challenges arise in designing for human-robot interactions (HRI) due to the fact that a large portion of the target user population is disabled or elderly. The question of how the human user can effectively and efficiently interact with these robotic systems has drawn much attention in robotics research.

Even common devices such as keyboards are being optimized for enhanced durability, faster response times, different feels of the click, and ergonomics. The keyboard market for mechanical keyboards alone is $602.1 million with growth to $642.2 million by the end of 2016 [14]. The market for computer input devices and expansion is huge. A subset of this larger market is people who do not have the ability to use the newest keyboard or touchscreen; these individuals are the group whose physical or neurological problems do not allow this common input method. The video game company Nintendo had devised a partial solution to this problem by creating the NES

Hands Free controller, Figure 1. Directional input is achieved by adjusting the controller with your chin and button presses are replaced by sucking in or blowing out air. These devices allowed minor fitting adjustments with straps and required partial movement of the player; although not ideal it allowed basic communication to the NES console.

Figure 1: Nintendo Hands Free Controller

To facilitate human-robot interaction, researchers have been investigating various new communication signals to extend the communication channels. These new communication signals are mainly adapted from natural interpersonal communication signals and biosignals, including speech [15], [16], facial expressions [17], [18], body gestures [19], [20], electromyography (muscle) signals (EMG) [21], [22], and electroencephalogram brain signals (EEG) [23], [24]. The goal of this investigation is to find the applicable communication signal that can be handled by the user with little effort to learn and utilize. Substantial research has been conducted on these signals mentioned

above to validate their feasibility and investigate their functionality. However, there is another promising natural signal, eye gaze, which has not been given enough attention in HRI. According to a new market research report [25] the eye tracking business is quickly expanding as well, estimated to reach $1,028.1 million by 2020. Although the market is large, the applications are not yet fully realized.

Gaze represents where a person is looking, which is estimated from eye movements. In light of monitoring technologies, gaze tracking can be categorized into three types: contact lens [26], video-based optical methods [27], [28], [29], [30], and electrooculogram (EOG) [31]. Nowadays, the most widely used technology is video-based eye tracking, a noninvasive optical method; in contrast, the other two need direct contact with the eyeballs or the skin around the eyes. Gaze tracking technology has been investigated for a long time and was widely used as a tool for human behavior studies to support research in neurobiology [32], [33], psychology [34], [35], computer science [36], [37], and human factors [38], [39]. However, there is limited work reported which used gaze as an interaction modality.

Gaze as an interaction modality between a human and a robot is natural and effortless, which makes it particularly promising for the disabled and elderly. Gaze is a natural communication modality among humans; for example, one person often uses his/her gaze to guide another person to an object of interest for joint attention. Managing gaze to look at particular locations is almost effortless, which does not require learning.

Even though the gaze modality is promising HRI, it is quite difficult to effectively utilize it to generate various control commands. Without appropriate interpretation, gaze has only been used as a pointing device to select a target from a candidate pool. One popular gaze interpretation method is to use the gaze direction to trigger a step driving command along one particular direction. In [31], [40], [41], wheelchairs were steered by gaze to move forward, backward, left or right, and the same method has also been used to steer a mobile robot in [42]. Similarly, gaze was used to drive a quadcopter [43] or rotate a robotic laparoscope system [44], [45] upward, downward, left or right. In these studies, a user had to continuously generate step driving commands using gaze until the robot incrementally reached the destination. However, gradually steering the robots step by step using gaze could be tedious for the users.

In previous work, users directly specify the destination that a robot needed to approach using gaze, which avoided triggering the incremental steering commands. In [46], [47], [48], gaze was used to define the concentration area of a robotic laparoscope system, and in [49], gaze was used to define the destination of a mobile robot. Controlling a robot by defining the goal using gaze is much easier as the users only need to generate the control command once, rather than repetitively triggering the incremental steering commands. Thus, the mental and physical burden can be reduced.

In these gaze-based robotic interaction examples, gaze has only been used to navigate robots to approach a desired location or orientation by controlling their motion, such as driving forward, backward, left and right. However, methods for generating rich and varied control commands, to interact with a robot, still constitutes an open problem.

One attempt to generate various commands using eye gaze signals is to create/code a blink-based language with which the user indicates commands by blinking the eyes in patterns based on duration, number, and/or frequency. This has been previously used to control wheelchairs [50] but requires up to 4 blinks for simple commands. This is impractical for complex commands, as it involves memorizing multiple blink-based patterns, requires the ability to stop blinking after long commands, and includes the assumption that the user does not blink to re-center their eyes or otherwise introduce "false positive" blink events. A problem with using timing for the duration of blinks is that humans are not extremely accurate at controlling this, and the blink duration will vary from person to person as well as be influenced by factors such as fatigue [51].

Another downfall of blink-based algorithms is the presence of involuntary blink events. Involuntary blinking, along with quick eye movements (called saccades), increases with the complexity of a given situation [52]. For example, in heavy traffic a person may blink 22.1 times per minute with 20.2 saccades per minute; this is compared to low complexity values of 19.6 and 11.7 respectively [52]. Depending on the situation, these involuntary eye movements and blinks could be difficult to distinguish from intentional commands. Regardless of these shortcomings, tracking blink events accurately has been accomplished [53] but should be used with moderation as the human variation in blink duration [51] and the inaccuracies introduced by involuntary blinking [52] could provide unsatisfactory results. With the addition of saccade effects [52], there seem to be too many issues to rely on blinking alone.

As a result of the numerous deficiencies, a new approach is suggested in this thesis. This new approach suggests that drawing shapes with the eye gaze will allow for more complex command inputs, greater remembrance of commands, and fewer mistakes interpreting the input compared to the blink-based approach. This approach can use blink-based commands to instruct the program when to start and stop recording shape-based commands. Through this combination of blink and gaze data channels, fewer errors are anticipated and a more natural HRI may be achieved. This thesis aims to explain the process of how these techniques work and how they are designed with the severely disabled user in mind to facilitate activities of daily living. A primary motivation is to enable robotic assistance for individuals with limited mobility and motor control, with the assumption that these individuals retain full control of eye movements (consistent with certain types of injury to, and degenerative diseases of, the nervous system). The gaze-based command language will create a diverse command set for the robot to carry out.

CHAPTER 2: APPROACH

i. BACKGROUND:

This work was carried out under the assumption that users have the ability to move their eyes; this is especially important in consideration of the target population of physically- and mobility-challenged users. We also assume the use of eye tracking software using a conventional computer monitor or screen (the Gazepoint GP3 Eye Tracker system is used in this work). The Gazepoint system collects data at 60 Hz, is accurate to within one degree of visual angle, and is compatible with a maximum screen size of 24 inches [54]. The software transforms the x and y pixel coordinates on the screen into coordinate gaze vectors which are used to construct the shapes for command detection, as described below. Coordinate vector tracking has been successfully accomplished in this way in previous research projects [55] for other applications.

Since a screen or monitor must be used with the Gazepoint system, it can also be used to feed information back to the user about the possible commands available (or provide context specific prompts). Overlaid guidance for drawing shapes can be provided as seen in Figure 2. This aspect is important as it reduces a user's workload of memorizing the commands and thus improves the user experience for people with impairment who may be confused by the technology. This will also improve the command detection accuracy as the user can trace the shape with their eyes. Using a screen allows for many different modes of feedback to the user, and indication of intent from the robot.

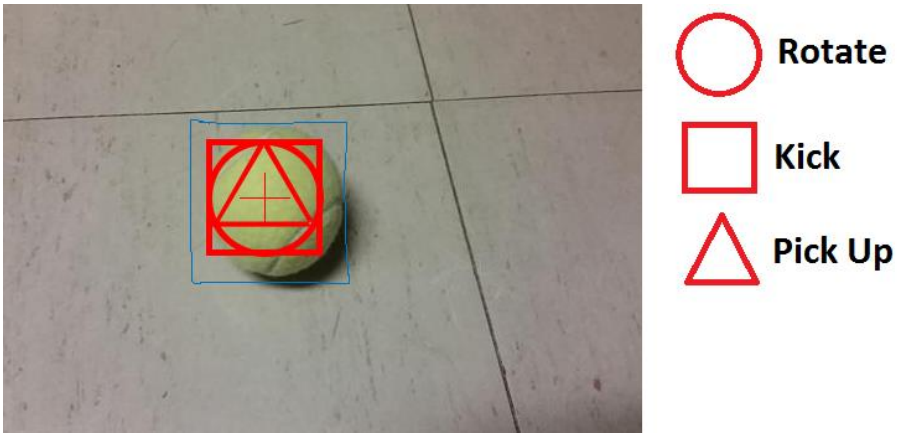


Figure 2. Shape guide with hypothetical command overlay

The scope of commands in this work is based on commercially available robots suitable for HRI. We adopt the small humanoid platform called NAO (Aldebaran Robotics) [56] as one of two experimental platforms, with the other being a custom-built wheeled robot. Algorithm results (gaze-based commands) are transmitted to the respective robot for execution by its integrated control software.

ii. STATE MACHINE CONCEPT:

A general concept that can be utilized to help simplify commands in the gaze-based language is the idea of state dependent commands. State-dependent programming allows for a limited set of options to be available to the user in a particular situation. A common example of a state machine is the context-specific menus that appear when performing a right mouse click in most software. This programming structure makes sense for assistant robots as not every command will be applicable or needed for every situation. The system can predict the user's likely set of intentions based on knowledge of the robot's surroundings, location, etc. and only enable relevant commands. This context-sensitive menu can then be displayed to the user on the screen when the user instructs the

program to start recording shape-based commands. This type of contextual interaction is achievable with the NAO robot using its documented object recognition capabilities [57].

The state-machine framework can be further illustrated by an example pertinent to our goal of assistive HRI. If NAO knew it was in the kitchen and recognized an empty glass on the table, this could trigger menu options such as ‘fill the glass’ or ‘put the glass in the cupboard’. These options would be displayed to the user’s screen, and after command input, a final blink approval could be administered for command execution.

Rather than elaborate on the development of state-based menu options for HRI in this paper, we focus on the task of recognizing shapes from user gaze (and distinguishing between different shapes). These shapes are the building blocks of a state-based HRI framework, as a few shape primitives can be used to “drill down” into the state- or context-specific menus displayed to the user. We specifically consider circles, triangles, and squares as candidate shapes.

CHAPTER 3: METHODS

This section presents the concepts of how the shape detection algorithm works and the reasoning behind its different elements; more explanation for specific results is discussed later in the thesis.

i. PROCEDURE:

The main program takes in gaze vectors in x and y coordinates and then runs through multiple testing procedures to determine which shape is traced (if a shape exists in the data). The overall flow process can be followed in Figure 3. For the convenience of the user and to account for variations among user characteristics and preferences, all settings used for determining the shape can be personalized for best performance. Experiments were run with these same processes as described in more detail below, to determine if using eye gaze to draw a shape was comparable to drawing shapes with a mouse. A final score was calculated, known as the Shape Points Score Estimation (SPSE), based on combined results of six different metrics which will be described in more detail (best-fit radius, corner detection, the area of Hough transform, the mean of Hough transform, the slope of Hough transform, the median of Hough transform).

$$\mathrm{SPSE} = \sum_{i=1}^{n} w_i \left(1 - \frac{M_{i,\mathrm{actual}} - M_{i,\mathrm{ideal}}}{M_{i,\mathrm{ideal}}}\right) \quad (1)$$

For n=6 shape-matching metrics (normalized M_i), each is multiplied by a weight w_j which is found through calibration (see Table 1 below). Higher metrics indicate a stronger correlation to matching the respective ideal shape.

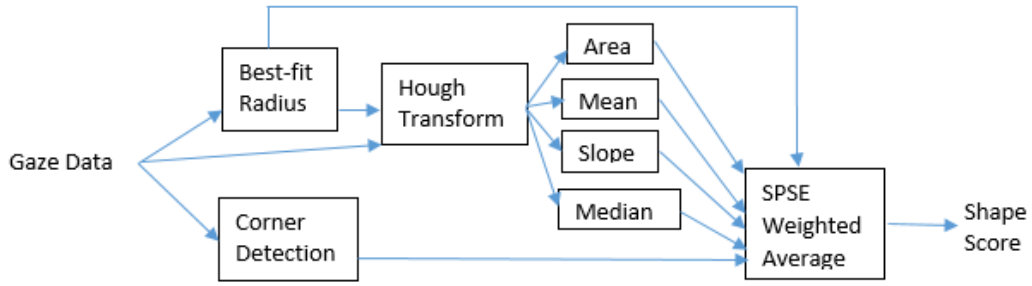


Figure 3. General data flow for shape detection

ii. SHAPERECOGN (HIGH-LEVEL ALGORITHM):

The main core of the program (the high-level algorithm) is a script called ShapeRecogn. This program takes in the settings.ini file and calls other functions, combining their results to output the command that can be interpreted by the robot. The user inputs the centroid of their position of interest by allowing the gaze to dwell on it and blinking once (detected from the eye gaze data, analogous to a mouse click) and the software then tracks the subsequent x and y gaze data, passing it to the various shape detection subroutines described below. The user blinks again to end the collection of shape data. The same effect is achieved with a mouse using a right click to start drawing and a right click to end drawing. Both sets of data (gaze-input and mouse-input) included shapes across a spectrum of "neatness" to reflect the inexact nature of the input data that would typically be input to the algorithm.

iii. CENTER FINDING TECHNIQUES:

The center of the object that is drawn is important for many aspects of the control algorithm. When shapes are drawn with the eyes, the center location may need further refinement, due to the high variance of the gaze point. Two options are available to the

user for center finding. One is optimized for speed and suffers in robustness whereas the other is optimized for shapes that are shifted off of their center location. The first option takes the mean value for both the eye gaze coordinate vectors in x and y and then calculates this as the center. The drawback of this method is that it is heavily affected by the noise of the unfiltered vectors. The other option is taking the vectors and creating a closed shape that is then converted to a binary image, for which properties information can be generated using the MATLAB function regionprops(). From there the centroid can be found from the statistics. This method is more computationally intensive but creates more accurate responses and is the chosen method by default.

iv. FILTERING:

The original gaze vectors are very noisy and need to be filtered for better shape recognition. Multiple steps are done to ensure accurate filtering. Some unintentional side effects of the filtering will be rounder corners, a slight decrease in shape size, and less gaze coordinate vector data overall to process. The first step is taking the standard deviation of both the x and y portions of the gaze vectors. This is then used as a criterion to reject outliers that are more than 1.5 times past the standard deviation from the center. This works as all usable data will be within the same general area, and the data beyond this threshold represent saccades or the start or end of drawing a shape. This also helps account for blinking as the eyes can jump far from the intended target during a blink.

The next step is a moving average filter that helps smooth the lines for easier shape detection, as shown in Figure 4. The filter size is dependent on a percentage of the total gaze data points from the start of the shape to the end of the shape. The default is set

to 10%. The results of the filtering can be seen in Figure 4, Figure 5, and Figure 6. Notice that Figure 6 did not need much filtering and as a result little filtering was done.

The importance of filtering data from gaze, as compared to other inputs such as a mouse, is due to the existence of saccades. Large saccades can cause the center of a shape to shift in the direction of the saccade leading to inaccurate detections, as in Figure 5. To combat this issue, an extra step to re-compute the center can be done after filtering is completed. This increases processing time but can also increase detection accuracy.

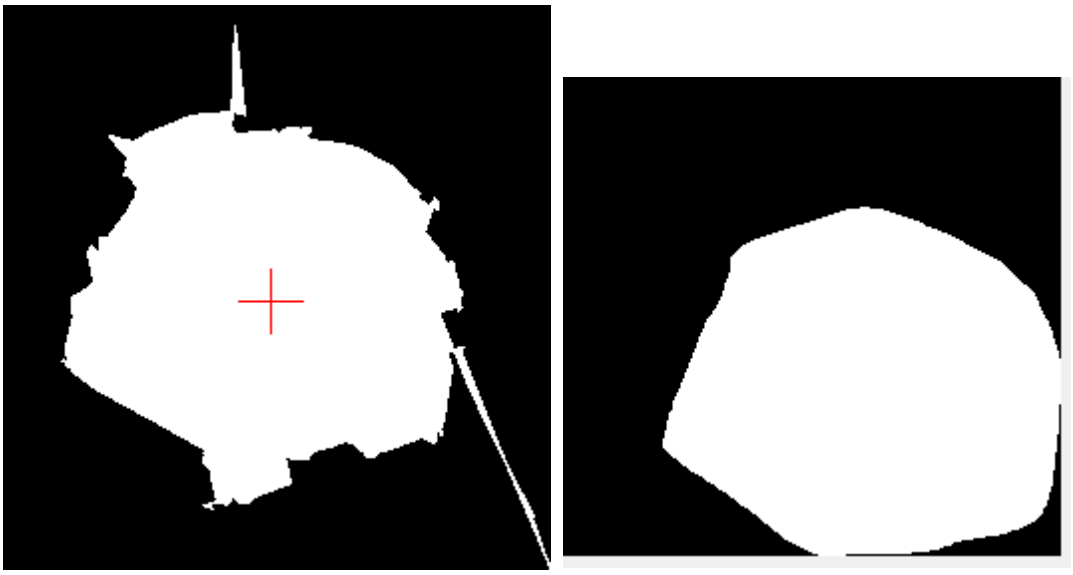


Figure 4. Smoothing Example

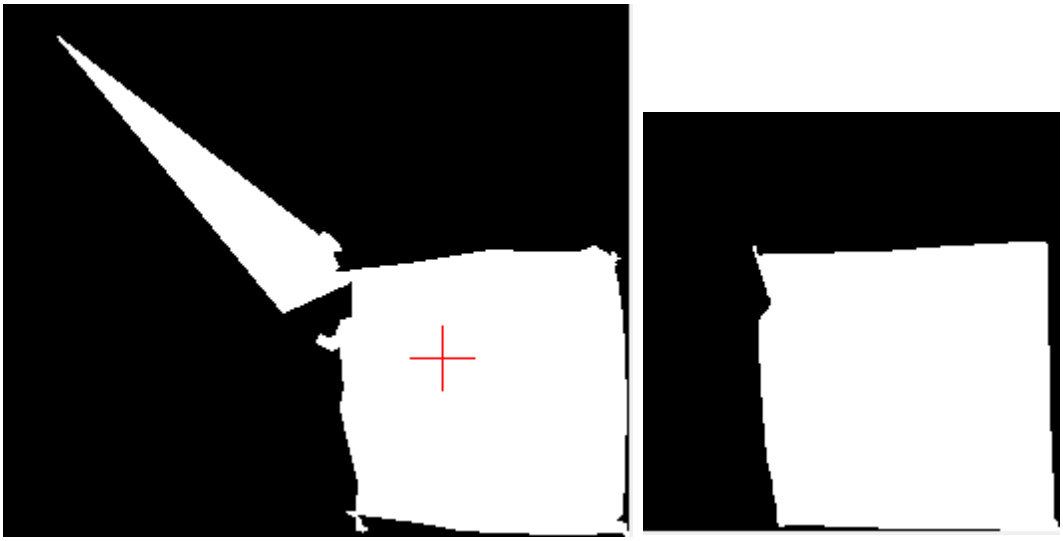


Figure 5. Saccade Example

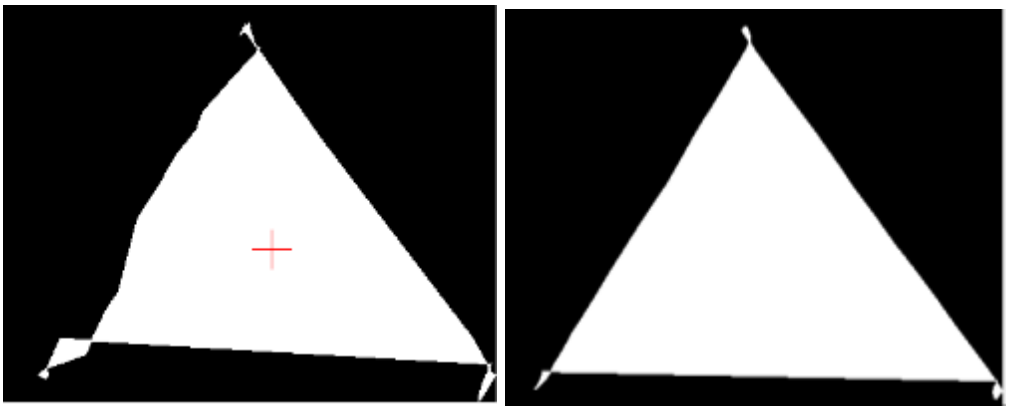


Figure 6. Smooth Shape Example

v. BEST FIT RADIUS ALGORITHM

ShapeRecogn reads the x and y coordinates of gaze data for the shape and the position of interest. The maximum and minimum distance from the position of interest (centroid) are found, and these values are used to construct bounding shapes as illustrated in Figure 7. A subroutine then counts the points that fall between the two shapes or bounds and calculates a score. The score is a percentage of points that fell within the bounds. The best length scale (e.g. radius, side length, hereafter referred to as radius for

all shapes) to fit the data is found for the three shapes, along with a percentage output representing the amount of data that fits within those bounds. These bounds can be adjusted in the settings.ini file depending on the degree of the disability and personal preference of the user. However, it should be noted that adjusting the bounds for each shape differently may affect the detection success negatively, as with large bounds squares may be detected as circles. The results are returned to the main function ShapeRecogn. This function executes quickly for small radius values, but as the radius increases the computational time also increases. The same issue occurs for poorly drawn shapes as the minimum radius may be quite small and the maximum radius quite large due to the roughness of the drawn shape.

Figure 7. Drawn a shape with bounds for best-fit radius algorithm

vi. CORNER FINDING ALGORITHM

The next metric uses Harris corner detection [58]. This metric uses the data for each radius from the best-fit radius algorithm and checks the corners at "expected" locations (based on the shape templates used in the menu structure). A buffer can be adjusted to allow for the sharpness of the corner to detect and the zone to search for each shape. In the case of circles, the entire profile is checked to check to see if any corners are detected on the shape; this is best seen in Figure 8. In Figure 8, asterisks are placed where the corner would be for each radius value; in the circle case, a circle profile is used

instead of asterisks. The number of corners detected then becomes a metric for SPSE that is returned to ShapeRecogn.

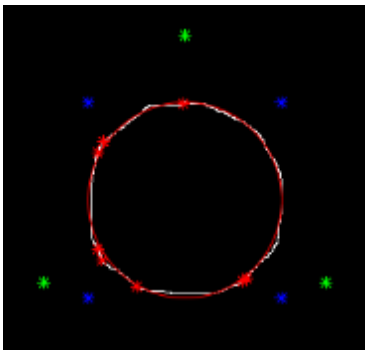


Figure 8. Corner Locations for the Circle

vii. HOUGH TRANSFORM METHODS

The next subroutine uses the Hough transform for shape detection. The Hough transform is designed to detect lines in a binary image. The function uses the parametric representation of a line [59]:

$$\rho = xcos(\theta) + ysin(\theta) \quad (2)$$

The function returns ρ, the distance from the origin to the line along a vector perpendicular to the line, and θ, the angle in degrees between the x-axis and this vector. These data are used to create the histograms in Figure 10, with ρ related to Hough matrix intensity. This process is used in several shape detection methods which follow.

$$H_{jk} = \begin{cases} 1, H_{jk} = i \\ 0, H_{jk} \neq i \end{cases}$$

$$d_i = \sum_{j=1}^{\#\ of\ rows\ of\ H} \sum_{k}^{\#\ of\ columns\ of\ H} H_{jk}\,, where\ i = 1{:}\max(\max(H) \quad (3)$$

This produces a graph where the x-axis roughly represents an increase of radius by 1 and the y-axis is the sum of the number of points in the Hough transform matrix that correspond to that radius. A data set might consist of 283 data points from the original 493 points in x and 493 points in y that created a Hough matrix of size 2371x360. The size of these parameters will change for every shape drawn but share similar shape characteristics.

a. PREPROCESSING:

A preprocessing step was implemented to speed up the runtime of the program. This preprocessing step was to create large tables of data for perfect shapes at various radii; this decreases program runtime by using reference values from tables (a “lookup” approach) rather than calculating the Hough transform for each possible shape and comparing data against the drawn shape. All of these calculations otherwise would create more processing time that the possibly disabled user has to wait as their gaze vectors are being calculated. This preprocessing step is a unique aspect of this project, as the image only consists of the shape drawn, whereas in most other applications of the Hough transform further processing and filtering are done to achieve accurate Hough data.

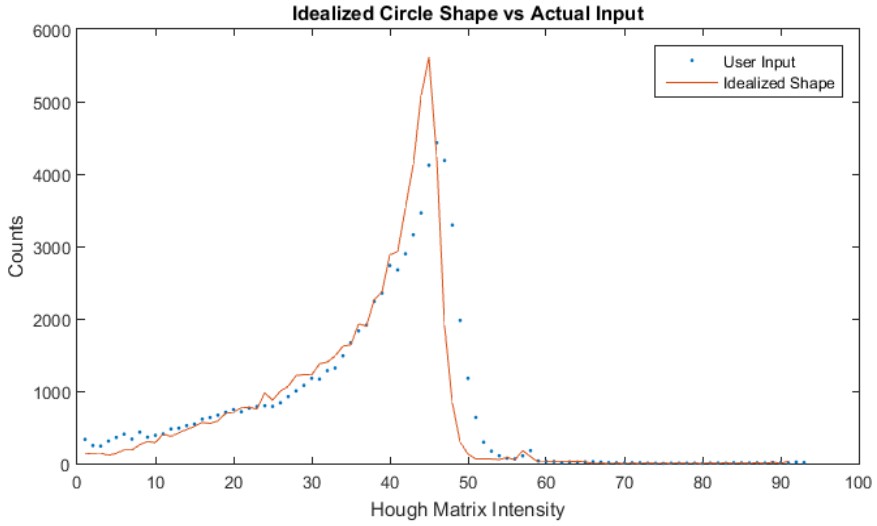
b. SHAPE DETECTION ALGORITHM:

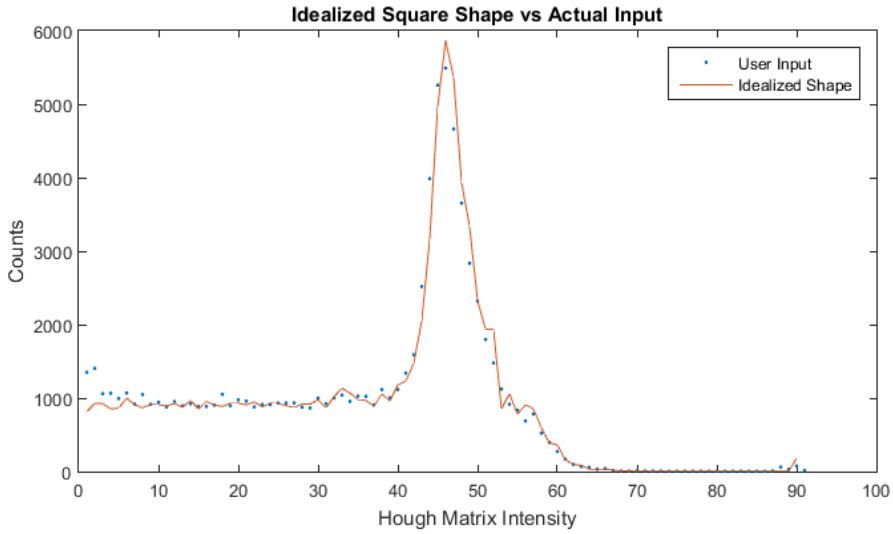
The Shape Detection Algorithm subroutine starts with the best radius for each shape and the drawn image. The drawn image consists of coordinate vectors that are connected to create an enclosed shape; this can be seen in Figure 9. This shape is then

filled to be able to apply the Hough transform efficiently. Due to preprocessing, the perfect Hough transform data is retrieved from tables, and only one Hough transform is required (for the traced image), speeding up the process. The values from the table are compared with the input data to determine the different metrics for each shape.

Figure 9: Enclosed Shapes

The drawn shape after the Hough transform is compared to the data from idealized shapes in preprocessing. The data can be represented in graph form as seen in Figure 10. The parameters for comparison are the area under the curve, the median, the average value at low Hough intensity, and the slope of the data. The slope of the data was calculated using representative points near the lowest intensity and the peak of the intensity histogram. The peak point location is calculated by the highest intensity of the ρ value (eq. 2). Figure 10 shows the Hough graphs for Circle, Square, and Triangle from the shapes in Figure 9 respectively, along with corresponding idealized shapes.


Idealized Circle Shape vs Actual Input
User Input
Idealized Shape
Counts
Hough Matrix Intensity
0
1000
2000
3000
4000
5000
6000
0
10
20
30
40
50
60
70
80
90
100


Idealized Square Shape vs Actual Input
User Input
Idealized Shape
Counts
Hough Matrix Intensity
0
1000
2000
3000
4000
5000
6000
0
10
20
30
40
50
60
70
80
90
100

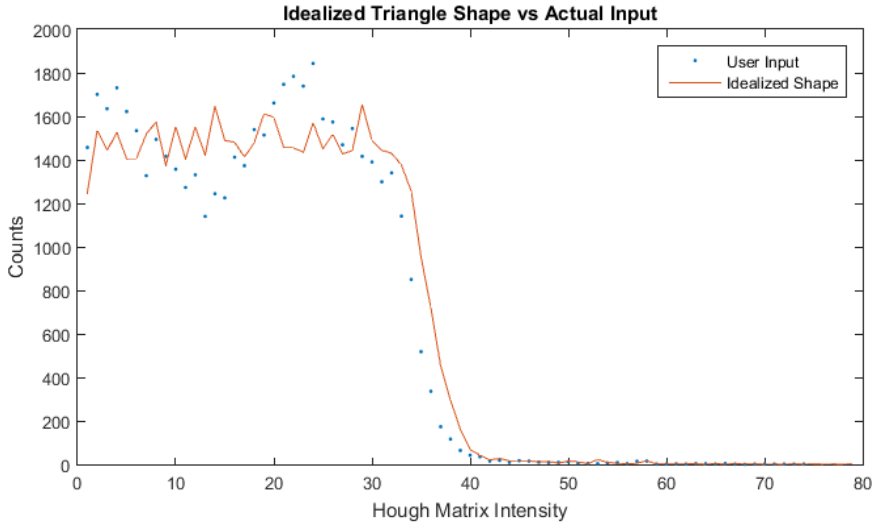


Figure 10. Unique Graphs for Respective Shapes from Fig. 8

viii. FINAL RESULT:

The result of the shape detection algorithm is determined by combining the different metrics from all of the subroutines (eq. 1). These functions create a score for each shape. The highest score is the shape that the program chooses as the correct shape. If the score is below a minimum threshold, no shape is chosen. All shapes receive a score due to normalization. The results of each test can be saved. This is helpful for testing and further analysis of the data to create a better fit for each user and each disability (patient-specific tuning of the algorithm). With the results of this program, commands can be sent to the robot to perform various assistive tasks.

CHAPTER 4: RESULTS:

A total of 120 traced shapes were analyzed (20 for each of the three test shapes, repeated using both gaze and mouse input). The weights for one user were calibrated from a small data set and are presented in Table 1. Table 2 and Table 3 show the average SPSE score of 20 results for each shape, using gaze and mouse input respectively. The rows represent the average value for each intended shape that was drawn, whereas the columns represent the shape-detection results. It is interesting to note the differences between the experiments done with the eyes compared to the experiments done using mouse input. The gaze-based SPSE values are all lower than the SPSE for the mouse-drawn results. This is further explained by looking at each metric individually to check the validity of the solution. It is important to note that the bold numbers represent the intended shape and intended result. These are always highest for final results, indicating that the valid results are returned.

Table 1. WEIGHTS USED FROM CALUBRATION

Metric	Radius	Corner	Area	Mean	Slope	Median
Weights	10	3	4	4.5	3	3

Table 2. SHAPE DETECTION FINAL RESULTS USING EYE GAZE

		Shape Detected		
		Circle	Square	Triangle
Shape Drawn	Circle	**18.6067**	14.89301	9.617871
	Square	11.92702	**18.54621**	9.597459
	Triangle	12.43349	9.56102	**18.42686**

Table 3. SHAPE DETECTION FINAL RESULTS USING MOUSE INPUT

		Shape Detected		
		Circle	Square	Triangle
Shape Drawn	Circle	**21.79355**	13.6458	5.932328
	Square	14.08615	**21.06246**	5.226813
	Triangle	6.929033	10.48045	**20.13124**

i. BEST FIT RADIUS ALGORITHM:

The first metric was from the percentage of data points that fell within the bounding shapes in the best--fit radius algorithm and then multiplying by the weights. By setting all weights, other than that for radius, to zero in (eq. 1), the individual contribution to SPSE can be seen in Table 4 and Table 5.

Table 4. POINT MATCH USING BOUNDING SHAPES WITH GAZE

		Shape Detected		
		Circle	Square	Triangle
Shape Drawn	Circle	**7.04515**	5.275	5.04847
	Square	5.15393	**5.19377**	3.58795
	Triangle	4.67693	3.45145	**5.82683**

Table 5. POINT MATCH USING BOUNDING SHAPES WITH MOUSE

		Shape Detected		
		Circle	Square	Triangle
Shape Drawn	Circle	**7.87036**	5.88343	2.19373
	Square	0.573813	**7.10807**	1.72813
	Triangle	0.35031	2.99144	**6.71473**

The results for the mouse input compared to the gaze-based input are interesting as they do not have the same patterns. One large weakness of the best-fit radius algorithm is the false detection of circles when squares are intended. This can be seen in Table 4. The reason for this is due to the increased roughness of the shape when using gaze compared to the smooth strokes of using a mouse. This roughness causes the values to be outside of the bounding shapes, which is what is used to check to see if the data is compliant to the model. Although the accuracy is lower, this metric still accomplishes the goal of finding the best-fit radius value.

ii. CORNER FINDING ALGORITHM:

The next metric is the results from the corner-finding algorithm. The idea is to match the number of corners detected with the corresponding shape (circle = 0, triangle = 3, square = 4). The weighted corner finder results (based on zero weights in (eq. 1) for other metrics) can be seen in Table 6 and Table 7. One thing to notice is that there is a large increase in the number of corners discovered in the gaze data compared to the mouse data. This is due to the extra bumps and curves in the image drawn by the eye being detected as corners. The maximum number of corners that the corner-finding algorithm looks for can be set to different values; in these experiments, it was set to 8 and only checked near expected corner locations. In the instance of a circle, the program checked the whole shape for corners. This is a reason why the circle results are so high.

Table 6. CORNER DETECTION BY SHAPE WITH GAZE

		Shape Detected		
		Circle	Square	Triangle
Shape Drawn	Circle	**1.95**	0.7	0.75
	Square	1.7	**1.9**	0.75
	Triangle	1.65	0.6	**1.9**

Table 7. CORNER DETECTION BY SHAPE WITH MOUSE

		Shape Detected		
		Circle	Square	Triangle
Shape Drawn	Circle	**1.0**	0.1	0
	Square	.7	**1.7**	0.3
	Triangle	0	0.5	**1.1**

iii. HOUGH METHODS:

The next set of metrics are based on the Hough transform and the approximations of the Hough transforms that were previously derived. By using the Hough transform each shape had unique properties that were separated into different calculable results contributing to the overall SPSE score.

a. AREA OF HOUGH TRANSFORM METRIC:

The first Hough-based metric is the difference in the area compared to the area found for a perfect shape with equal radius. In other words, it is the basic integral under the curve in Fig. 8. This area is subtracted from the perfect shape area. All weights except that for the area are set to zero in (eq. 1), and the results are seen in Table 8 and Table 9. Something worth noting is that the area for a circle shape and a square shape are nearly identical, but this is not the case for the triangle (although poorly drawn circles can be mistaken for triangles). This can be seen in the triangle drawn shape in Table 8 as the circle was incorrectly chosen by the program. Otherwise, the numbers are as expected for both the mouse and the gaze-based trials.

Table 8. AREA OF HOUGH TRANSFORM- GAZE CASE

		Shape Detected		
		Circle	Square	Triangle
Shape Drawn	Circle	**2.547442**	1.903898	1.419948
	Square	1.536639	**2.485485**	1.695392
	Triangle	3.183223	0.440902	**2.513845**

Table 9. AREA OF HOUGH TRANSFORM- MOUSE CASE

		Shape Detected		
		Circle	Square	Triangle
Shape Drawn	Circle	**2.958929**	0.947028	2.456028
	Square	1.999308	**2.422011**	1.421321
	Triangle	0.84204	2.646811	**2.777951**

b. MEAN OF HOUGH TRANSFORM METRIC:

The second metric for Hough-based shape detection is finding the mean of the first one-third of the Hough histogram. In the first third of the data, the mean tells quite a lot about what shape has been drawn (see Figure 10). However, one weakness is that the average value of square and circle are again very close to each other, but triangles are quite distinguishable, as seen in Table 10 and Table 11. In both the mouse and the gaze-based cases, the triangle stands out easily.

Table 10. MEAN OF FIRST ONE-THIRD OF HOUGH TRANSFORM-GAZE CASE

		Shape Detected		
		Circle	Square	Triangle
Shape Drawn	Circle	**2.626377**	3.770403	0.647659
	Square	1.571241	**3.828741**	1.244637
	Triangle	0.835423	2.005884	**3.846952**

Table 11. MEAN OF FIRST ONE-THIRD OF HOUGH TRANSFORM-MOUSE CASE

		Shape Detected		
		Circle	Square	Triangle
Shape Drawn	Circle	**4.185824**	3.09798	0.237311
	Square	2.279997	**4.253781**	0.619156
	Triangle	0.684837	2.134503	**4.240643**

c. SLOPE OF HOUGH TRANSFORM METRIC:

The third metric is the slope evaluated from the beginning of the Hough transform data to the peak in the data, as shown in Figure 11. The peak location varies by shape but represents a characteristic radius similar to an inscribed circle. The peak value was calculated using an averaging window with 5 data points immediately preceding the highest matrix intensity value (eq. 3). These values are used to determine the difference between square results and circle results. Square and triangle have similar slopes and should appear to have similar numbers if the variance of the data is not too high. Slopes

were compared using (eq. 1) with all weights other than for slope equal to zero, yielding Table 12 and Table 13. The circle's variance continued to be a problem as it affected the results for both circle and square readings. Otherwise, the data are still fairly comparable and show correct results for square and triangle.

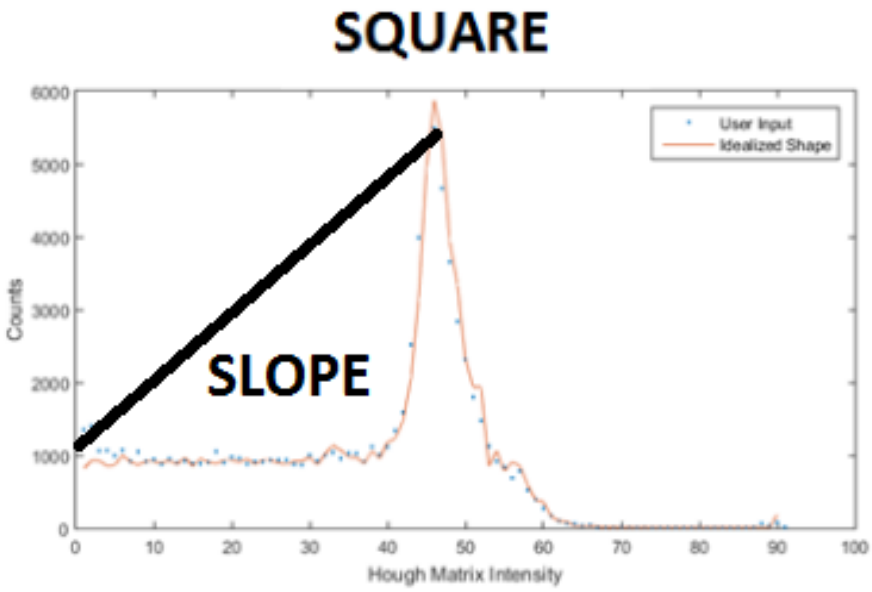


Figure 11. Example of Slope and Peak Location for Square

Table 12. SLOPE OF HOUGH TRANSFORM -GAZE CASE

		Shape Detected		
		Circle	Square	Triangle
Shape Drawn	Circle	**1.437732**	2.24371	0.7518
	Square	0.965202	**2.138216**	1.319481
	Triangle	0.387915	1.862779	**2.239229**

Table 13. SLOPE OF HOUGH TRANFORM- MOUSE CASE

		Shape Detected		
		Circle	Square	Triangle
Shape Drawn	Circle	**2.778436**	2.617368	0.045264
	Square	2.368715	**2.578591**	0.158205
	Triangle	0.699052	1.2077	**2.497919**

d. MEDIAN OF HOUGH TRANSFORM METRIC:

The fourth metric is slightly different from the other metrics. This metric is based on the median Hough data value. The median always follows an average trend where the triangle has the smallest median, the circle has the middle median, and finally, the square has the largest median. The corresponding data are shown in Table 14 and Table 15. Although the median value is slightly different when comparing gaze to mouse-based input, the trend is the same.

Table 14. MEDIAN OF HOUGH TRANSFORM-GAZE CASE

Circle Results	478
Square Results	816.9
Triangle Results	261.5

Table 15. MEDIAN OF HOUGH TRANSFORM-MOUSE CASE

Circle Results	251.1
Square Results	836.9
Triangle Results	58.3

iv. RESULTS SUMMARY:

All of these weighted metrics from the Hough transform data (slope, median, area, mean) are added to the data returned by the other subroutines (percentage of points, corners) to determine the detected shape. The final results can be seen comparing gaze to mouse data in Table 2 and Table 3. The average values for all of the test trials showed that the SPSE values showed the correct shape on average. In fact, all cases gave the correct shape according to the SPSE. To ensure no false positives, the program can be set to not find a result for anything less than a threshold value of SPSE; for example, in Table 2 the value could be set to 15 based on the second highest SPSE score.

v. HUMAN TESTING:

To improve the robustness of the algorithms, human testing was needed. The next improvement was the inclusion of a GUI for the user to interface with, shown in Figure 12. The entire algorithm was implemented in the ShapeRecogn() function, only requiring x and y coordinate vectors and a center position to calculate the shape and SPSE scores to output to the user. This advancement allows for the broader application of the shape recognition software.

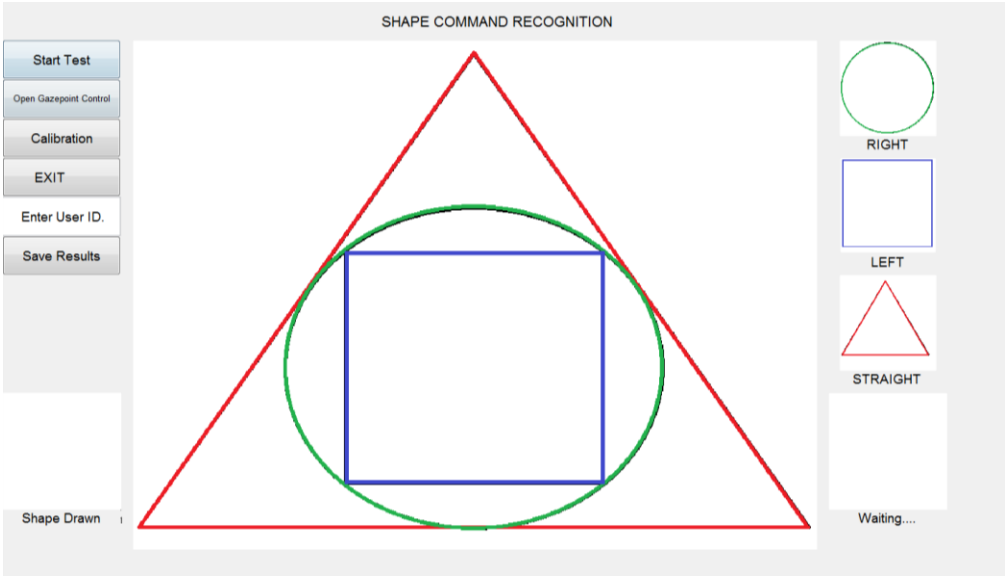


Figure 12. User Interface

The interface seen in Figure 12, is part of an ongoing study intended to obtain pilot data, determine potential problems, gauge the robustness of the program, and have the ability to tune the system to future users by establishing common features. These common features would include the speed the user draws each shape, the amount of wavering around lines, amount of saccades, and many other features. Data was collected under an IRB-approved protocol with informed consent of the participants.

Participants were asked to sit in a chair and look at a computer screen. They were then asked to "draw" a few test shapes (circle, square, triangle) using their eyes, so they became familiar with the task and software. Once familiar with the system, each participant was asked to draw a total of 30 shapes (10 each of the three types of shape, in random order). The gaze coordinate vectors were recorded and saved in text files. The participants were then asked to complete a survey consisting of questions asking if they

encountered any discomfort, how easy it was to draw shapes, if their eyes became dry, how well they felt the system identified their intended shapes, and any pros and cons of usability.

The preliminary SPSE results, as seen in Table 16 for the first six users, along with the subjective user feedback, will help make future improvements to tune the shape recognition system. The preliminary results show that on average the correct shape is being drawn even by inexperienced users. It is worth noting that the SPSE scores are lower than those in Table 2 and Table 3. This is due to some inexperience, the system not being individually tuned to their preferred settings, and other potential issues discussed later.

Table 16. SHAPE DETECTION FINAL RESULTS USING EYE GAZE.

		Shape Detected		
		Circle	Square	Triangle
Shape Drawn	Circle	**15.5720**	11.3761	7.6389
	Square	11.8692	**14.0893**	8.79702
	Triangle	8.86922	10.8502	**14.1145**

Although further testing is needed to fully understand the results, the current user base information provides insight on potential correlations (see Table 17).

Table 17. USER BASE STATISTICS

Question	Units	Result
Discomfort?	% Yes	80
Easy to draw shapes?	Scale (1-5)	3.533333
Eyes become dry?	% Yes	100
Correct Identification?	Scale (1-5)	3.8

The most common problem is an issue with eyes becoming dry. This occurred around shape 20 of 30 for most users. The user was allowed to take a small break if they wanted but only one user ever did. All notes on discomfort were from dryness in the eyes. Currently, no users with vision or mobility disabilities have tested the system. The most common suggestion was that when the shapes intersected on the GUI, their eyes would wander on the wrong shape for a while before correcting. Another helpful suggestion from multiple users was to change the background GUI to be a different color other than white, for example black, to decrease the brightness on the eyes, which in turn would decrease eye dryness. The last common complaint or suggestion was users' perception that the program smoothed the corners of the shapes. This is a side effect of the filtering and the time spent looking at each corner. This should not be an issue in processing, but it may have caused some user dissatisfaction.

vi. TESTING WITH NAO ROBOT:

The ShapeRecogn() program was used to control a NAO robot; (Figure 13). The robot was given a list of shape-based commands representative of a command sub-menu; this can be seen in Figure 13. Three healthy subjects were asked to draw 30 shapes, 10 for circle, square, and triangle respectively. Each shape then triggered the NAO's action. The percentage of correct shapes that were detected for the three test subjects were 90%, 97%, and 100%, for circle, square, and triangle respectively. This was without changing values in the settings.ini file to help further improve the percentages (tuning the algorithm for the specific users). These results demonstrate that the NAO is responsive to the program output and performs the intended function.

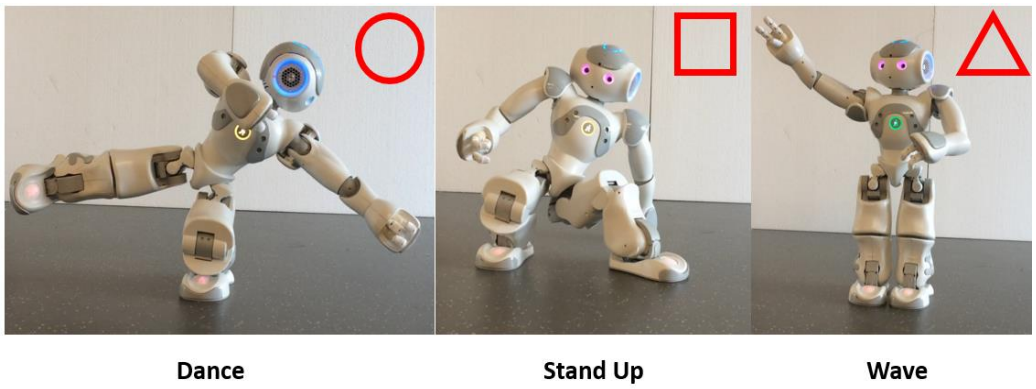


Figure 13. The NAO robot is responding to gaze-based commands

vii. TESTING WITH LOCATION DEPENDENT COMMANDS:

To demonstrate the idea of controlling robot navigation, seen in Figure 14, using shape-based commands, a grid was created within which a robot can navigate. The grid, as seen in Figure 14, had four sections labeled A, B, C, and H for home. The user was

given three options for travel to locations (A Circle, B Square, C Triangle), using the interface from Figure 12. The robot then moved to the designated location. Once at this location, it gave the user three options for turning on LEDs (Blue Circle, Red Square, Yellow Triangle). The robot has now completed its task and returns to the center location, ready for a new command. Performance was judged by the robot navigating to the correct zone 10 times using only commands given with eye gaze patterns. One mistake occurred where the wrong shape was detected, the right shape was then drawn and the robot moved to the correct location.

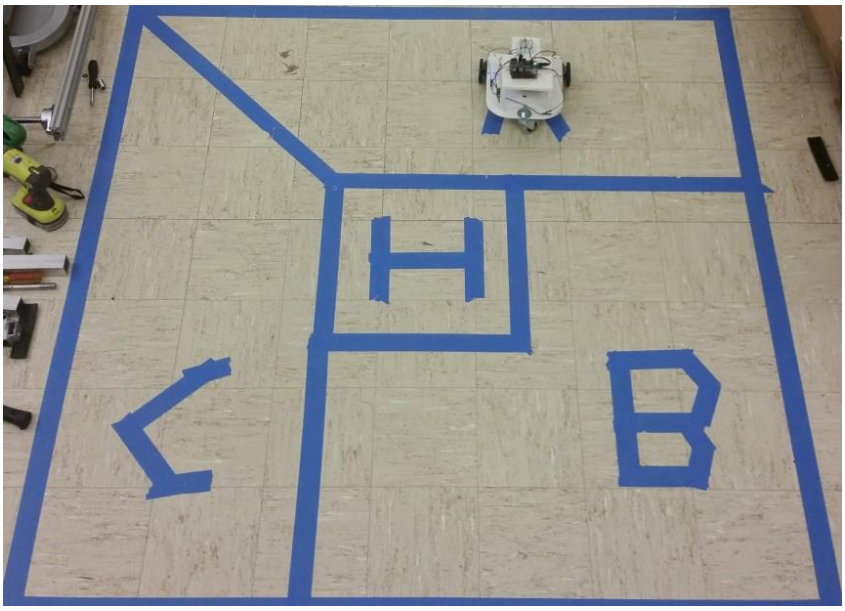


Figure 14. Robot at Position A

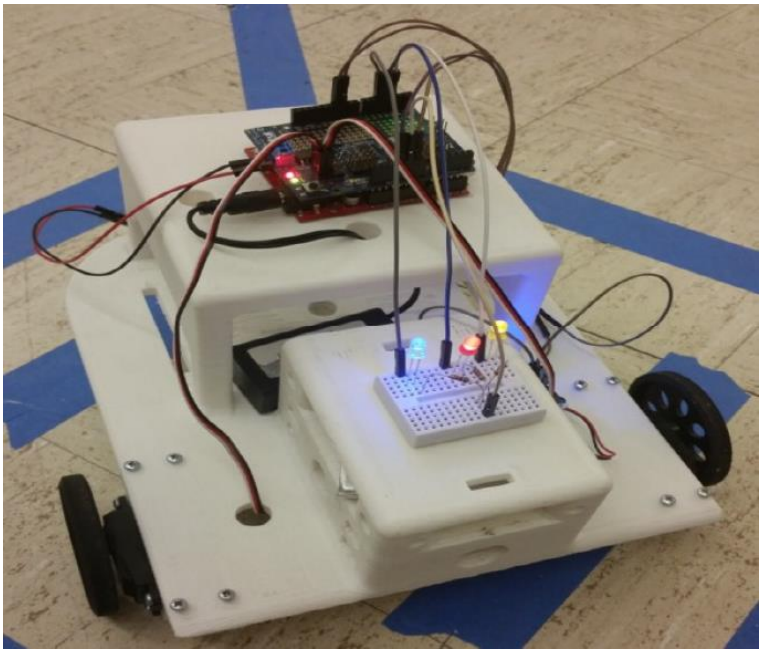


Figure 15. Robot with 3 LEDs

Although the demonstration is quite simple and lacks complexities such as obstacle avoidance, the relevant applications can be quite varied. For example, location A could represent a kitchen and each LED a command, perhaps to retrieve pills, water, or turn off the stove. The robot then returns after completion of the task and the user can issue a new command. The state dependency refers to a physical location in this example but could be states of time, temperature, or velocity.

CHAPTER FIVE: STOCHASTIC ANALYSIS

The shape solving algrorithm is fast and efficient but can still be improved. One area of improvement would be to reduce the overall processing time. The size of the Hough matrix lends itself to being the most time intestive part due to processing. The current method is to process all of the Hough transform matrices to get the intensity data that can then be compared to the new data intensity. The comparison looks at the slope, mean value, median value, and area under the curve to produce a score. All of these calculations create more processing time that the possibly disabled user has to wait as their gaze vectors are being calculated; this is due to the different requirements in processing based on the resulting drawn image size. The objective of this data modeling will be to create a dynamic model against which new gaze vector data can be compared to see if it fits the shape model. Potential challenges include different locations of shape start and end points, outliers where the eye traveled to a new point quickly, and different size of the shapes being drawn. An ARMA (Auto Regressive Mean Average) model can be made to represent different parameters of the eye and the eye gaze data. These models will be constructed for all the Hough transform models and the original x and y shape data. This will not only provide greater insight on how the eye is tracing shapes but be able to predict what shape is being drawn without excessive processing.

ARMA models can be found by using methods of Data Dependent Systems (DDS). A DDS methodology begins with knowledge and data collected from a system, in this case gaze coordinate vectors. From the observed data the development of statistically adequate models can be started without full knowledge of all system parameters. From

this data alone forecasting gaze coordinates is possible [60]. The method of DDS can provide a differential equation that governs the system [61] and can provide information on the things such as the frequency of saccades or other factors that affect eyes movement. The DDS approach uses least-squares techniques to fit a series of difference equations until a statistically adequate model is achieved; this is the basis of the ARMA model. The modeling is of the form [61]:

$$x_t - \phi_1 x_{t-1} - \phi_2 x_{t-2} \ldots - \phi_n x_{t-n} = a_t - \theta_1 a_{t-1} \ldots \theta_{n-1} a_{t-n+1} \quad (4)$$

$$\bar{x} = \frac{\sum \dot{x}_t}{N} \quad (5)$$

$$x_t = \dot{x}_t - \bar{x} \quad (6)$$

where x_t are the data responses at a certain time, and a_t is the associated white noise. ϕ_i is the auto regressive coefficient of lag i, and θ_j is the moving average coefficient of lag j [62]. N is the number of samples. In other words N is the number of gaze coordinates, and ϕ_i and θ_j represent the linearity of the gaze coordinate values over time. This information will be beneficial in path prediction as it can predict where the eyes will be several points ahead, allowing for faster processing potential. The model can also provide a non-biased dual verification method as to what shape was drawn and where the shape was started.

The first data set will be strictly the x coordinates gathered by the GP3 that correlate to the circle shape. This vector will be substitued into the equations for a DDS and evaluated at different ARMA models until a minimum Residual Sum of Squares

(RSS) is found. The different ARMA models will start small, such as ARMA(1,1) which represents only one autoregressive part (ϕ) and one mean average part (θ), and increase until the F-test is satisfied. It should be noted that a higher order model may be required dependent on shape of the data. For more explanation of the equations or of the process see Appendix 1.

Using the rules of the F test, Table 18 was created.

Table 18. F-Test of Differnt ARMA Models

Order	RSS	#UAC>3	F0<>Fcrit
0,0	4295584	105	
1,0	5832.615	24	361854.5054>3.8604
2,1	4622.992	4	42.6495>2.6231
4,3	4554.955	3	1.8111<2.3903
6,5	4480.427	3	1.9132<1.9576

The model of interest is the ARMA(2,1) model. This was chosen due to the ARMA (4,3) model showing not much improvement in the reduction of errors proven with the F test (1.8111<2.3903). The model being an ARMA(2,1) means we will have two ϕ and one θ. These values are found by iterating until RSS is as small as statistically possible. Note that the unified autocorrelation (UAC) is not 0, meaning that there is correlation among the residuals.

After finding the parameters of the equation the values are $\phi_1 = 1.4518$, $\phi_2 = -.4528$ and $\theta_1 = 3.2289 * 10^{-5}$. Looking at a ARMA(2,1) model that appears in this form:

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + a_t - \theta_1 a_{t-1} \text{ ARMA(2,1)}$$

Substituting the values from the equations for gaze vectors in X:

$$x_t = 1.4518 x_{t-1} - .4528 x_{t-2} + a_t - 3.2289 * 10^{-5} a_{t-1}$$

We can use this model to estimate many parameters that will be helpful in forecasting and analysis. Figure 16 and Figure 17 were also made to help better explain the different parameters.

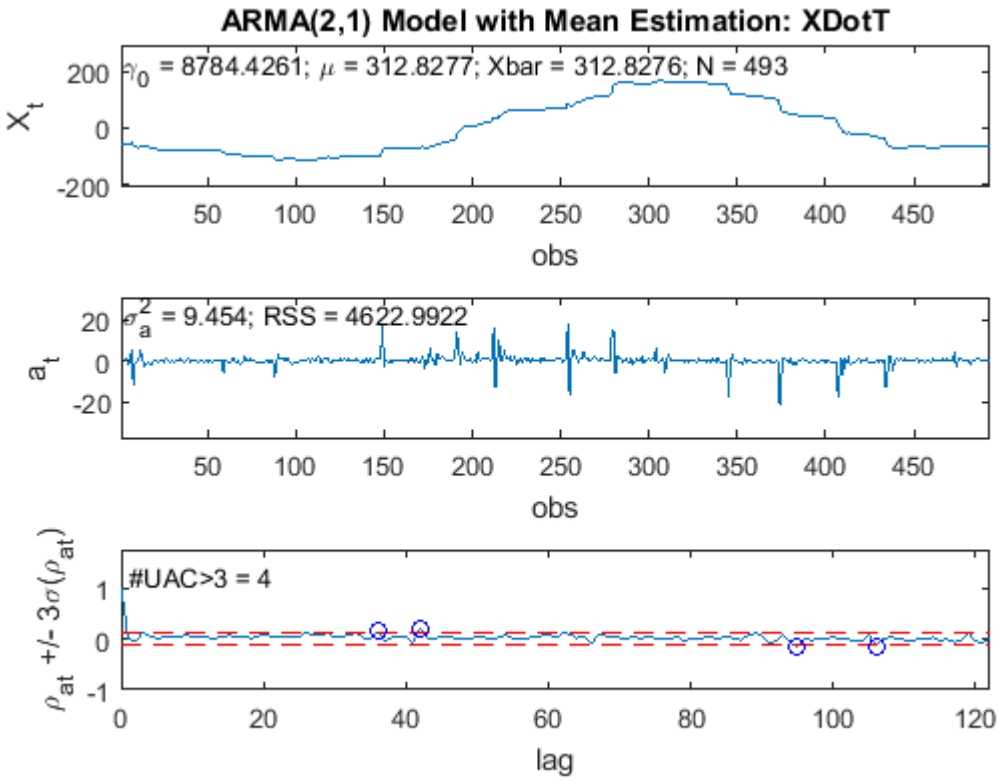


Figure 16: ARMA (2,1) Model Circle X Coordinate

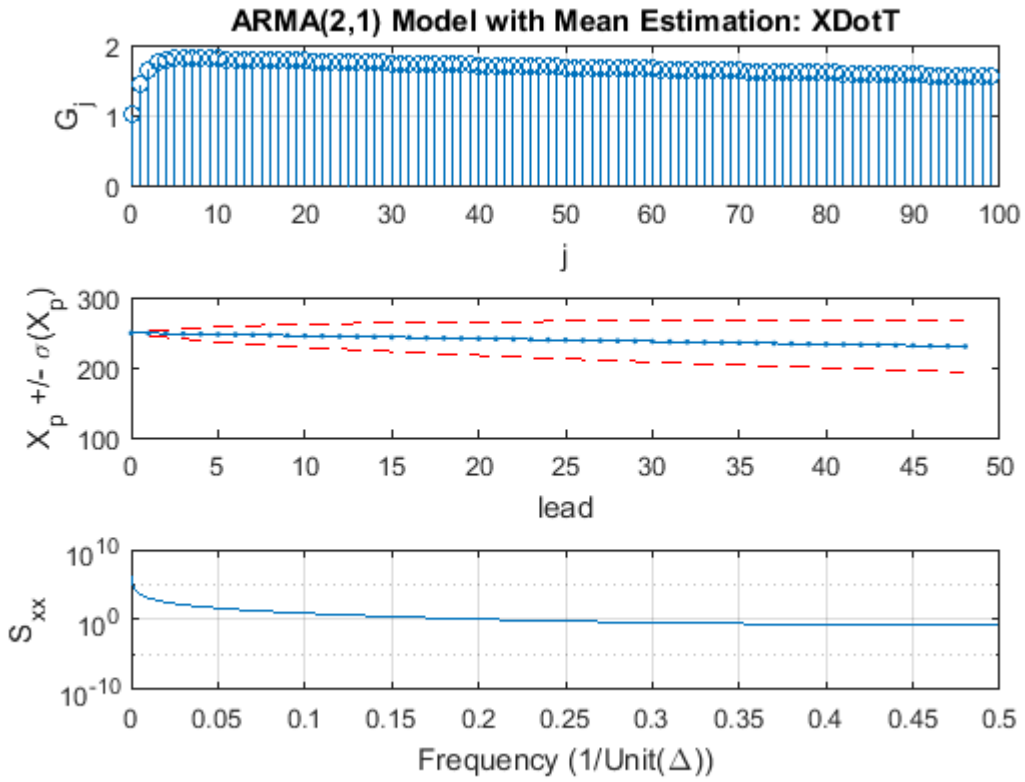


Figure 17: ARMA (2,1) Model Circle X Coordinate Future Predictions

Because we are looking at ARMA(2,1) models for this model we can use the explicit equation for the Green's functions. The equations are given below (Eqs. 7-9), but can be received through most DDS software, as seen in Figure 17.

$$G_j = g_1\lambda_1^j + g_2\lambda_2^j \text{ for real } \lambda_1 \text{ and } \lambda_2 \quad (7)$$

$$g_1 = \frac{\lambda_1-\theta_1}{\lambda_1-\lambda_2}, g_2 = \frac{\lambda_2-\theta_1}{\lambda_2-\lambda_1} \quad (8)$$

$$\lambda_1, \lambda_2 = \frac{1}{2}(\phi_1 +/-\sqrt{\phi_1^2 + 4\phi_2}) \quad (9)$$

Knowing the values of λ allow for the user to see if the system is stable, unstable, or asymptotically stable following these conditions.

Table 19. Stability Conditions

$\lvert\lambda_1\rvert$<1	And	$\lvert\lambda_2\rvert < 1$	Asymptotically Stable
$\lvert\lambda_1\rvert$>1	And/Or	$\lvert\lambda_2\rvert > 1$	UNSTABLE
$\lvert\lambda_1\rvert$=1	And	$\lvert\lambda_2\rvert < 1$	STABLE
λ_1=1	And	$\lambda_2 = -1$	STABLE

In this first case $\lambda_1 = .9983, \lambda_2 = .4535$ this condition means that the system will be asymptotically stable. θ_1 is equal to $3.2289 * 10^{-5}$ which is very close to zero allowing some simplifications.

$$g_1 = \frac{.9983 - 3.2289 * 10^{-5}}{.9983 - .4535} = 1.8324, g_2 = \frac{.4535 - 3.2289 * 10^{-5}}{.4535 - .9983} = -.8324$$

$$G_j = 1.8324(.9983)^j - .8324(.4535)^j$$

It can be seen in Figure 17 that the Green's function is very slowly approaching 0. This is mostly due to the fact that the equation is subtracting from itself at almost the same rate that it is adding. When j=10000 we achieve a point at $7.5 * 10^{-8}$, and when j=1000 we achieve a point at .33. This means that the system has a very strong memory. In other words the effects from previous data strongly affect the future system values as well; this is not necessarily bad as it depends on how you are expecting your values to change. Since the x coordinate could change at any time to a new position quickly it

helps smooth the data, but if the data keeps moving unexpectedly it may be a bad fit for the current model. This would be due to the values not responding fast enough to the current model. This can be seen when forecasting.

i. FORECASTING:

With the ARMA(2,1) model can now forecast values for the system. Forecasted values could be utilized in the system to predict the next few points of where a user was drawing with their eyes. These predicted points could be compared with the real points and the longer processing tasks could be started early if the beginning of a vector was following the forecasts for a certain shape. The basics of forecasting an ARMA(2,1) model as follows.

$$\hat{x}_t(1) = E_t[x_{t+1}]$$

$$\hat{x}_t(1) = E_t[\phi_1 x_t + \phi_2 x_{t-1} + a_{t+1} - \theta_1 a_t]$$

$$\hat{x}_t(1) = \phi_1 x_t + \phi_2 x_{t-1} - \theta_1 a_t$$

$$\hat{x}_t(2) = E_t[\phi_1 x_{t+1} + \phi_2 x_t + a_{t+2} - \theta_1 a_{t+1}]$$

$$\hat{x}_t(2) = \phi_1 \hat{x}_t(1) + \phi_2 x_t$$

$$\hat{x}_t(3) = E_t[\phi_1 x_{t+2} + \phi_2 x_{t+1} + a_{t+3} - \theta_1 a_{t+2}]$$

$$\hat{x}_t(3) = \phi_1 \hat{x}_t(2) + \phi_2 \hat{x}_t(1)$$

Any values past $l \geq 3$ steps ahead that will be

$$\hat{x}_t(l) = \phi_1 \hat{x}_t(l-1) + \phi_2 \hat{x}_t(l-2) \text{ for } l \geq 3$$

So taking the model

$$x_t = 1.4518x_{t-1} - .4528x_{t-2} + a_t - 3.2289 * 10^{-5}a_{t-1}$$

and looking one step ahead

$$\hat{x}_t(1) = 1.4518x_t - .4528x_{t-1} - 3.2289 * 10^{-5}a_t$$

One can look at a data point near the start of the data series to simulate a computer guessing the next point. We are going to be le looking at $x_{50} = -78.8276$, $x_{49} = -78.8276$, $a_{50} = -.0730$ and evaluating success after looking 4 steps ahead.

$$\hat{x}_{50}(1) = \phi_1 x_{50} + \phi_2 x_{49} - \theta_1 a_{50}$$

$$\hat{x}_{50}(1) = 1.4518(-78.827) - .4528(-78.827) - 3.228 * 10^{-5}(-.073) = -78.7488$$

$$\hat{x}_{50}(2) = \phi_1 \hat{x}_{50}(1) + \phi_2 x_{50} = 1.4518(-78.7488) - .4528(-78.8276) = -78.6344$$

$$\hat{x}_{50}(3) = \phi_1 \hat{x}_{50}(2) + \phi_2 \hat{x}_{50}(1) = 1.4518(-78.634) - .4528(-78.748) = -78.5040$$

$$\hat{x}_{50}(4) = \phi_1 \hat{x}_{50}(3) + \phi_2 \hat{x}_{50}(2) = 1.4518(-78.504) - .4528(-78.634) = -78.3665$$

Table 20: Summary of Predicted Points

	Forecast		Real	Percent Difference (%)
$\hat{x}_{50}(1)$	-78.7488	x_{51}	-79.8276	1.360588714
$\hat{x}_{50}(2)$	-78.6344	x_{52}	-79.8276	1.505958919
$\hat{x}_{50}(3)$	-78.504	x_{53}	-80.8276	2.916667388
$\hat{x}_{50}(4)$	-78.3665	x_{54}	-80.8276	3.091931699

The forecasted and real values are surprisingly very close, only being off by 1.3605% for the one step ahead, and 3.09% off for the 4 steps ahead. The error could be further reduced by updating the forecast with the new known values using the Green's function. If we look back at the graph for X_p DDS shows the prediction of points with fairly large error bars. The prediction points are estimated at the end of the data series with errors growing the farther from the predicted point. This prediction is not as useful as the data of interest is within the data set and anything after should not be contributing to shape drawing aspects.

ii. FREQUENCY, DAMPING RATIO, POWER (VARIANCE) CONTRIBUTION:

Table 21. Frequency, Dampening Ratio, and Power Parameters

λ	0.9983	0.4535
ω_n	2.6646*10^-4	0.1258
P	0.6943	1.5284
d	9.4697*10^3	-18.1036
ζ	1	1

The parameters of the eye gaze coordinate vectors are similar to a spring-damper system (Table 21); (see Appendix 1). The value of ζ being equal to one represents a critically damped system. The frequency may show the eye doing saccades over a portion of the shape outline it is tracing. In this case the system is fairly quick reacting, with a small frequency, and a critically damped system. It should be noted that the values of d_1 and d_2 give us insight on the variance. There is a very large power contribution in d_1 with a value equal to $9.4697 * 10^3$; this is not preferred as it represents that our variance is correlated with the system. This could be potentially lower with higher-order models.

iii. DISCUSSION:

The explanation of the DDS method in this thesis looked at the x gaze coordinate data of a circle, other shapes can be found in Appendix 1. We fitted an ARMA(2,1) model to the system but higher order models could have been used to help reduce some of the errors and make a more compliant model. The model allows data collected just from the x gaze coordinate values to predict the starting point for an x/y system and therefore predict the shape without knowledge of the other vectors. The frequencies of the system do not match the sampling frequency, meaning that we are sampling at high enough rates to see effects of unknown frequencies. Theses frequencies may be from the twitching of the user's eyes or from some other unknown effect. High-order models may first need to be run to see what the frequencies represent. The Green's function also leaves room for improvement, as it is slow to reach 0. A more ideal system would have a quicker response with the Green's function. Another advantage of the system is the ability to predict a point midway into the data set. This can be implemented by processing a collected data point while the system is still collecting data and then comparing the future value to the predicted value to see which model it fits. Although the discussion for this thesis is mostly based on the circle x coordinates the other parameters were looked at as well, please see Appendix 1 for further discussion and processes on this data. The Hough Transform was looked at as well and is summarized below with ARMA(4,3) models; this discussion can be seen in Appendix 1 as well.

Summarizing all Shapes:

Circle:

$$x_t = 1.4518x_{t-1} - .4528x_{t-2} + a_t - 3.2289 * 10^{-5}a_{t-1} \text{ X}$$

$$x_t = 1.4280x_{t-1} - .4299x_{t-2} + a_t - .0346a_{t-1} \text{ Y}$$

Square:

$$x_t = 1.4605x_{t-1} - .4616x_{t-2} + a_t + .0264a_{t-1} \text{ X}$$

$$x_t = 1.3803x_{t-1} - .3818x_{t-2} + a_t - .0627a_{t-1} \text{ Y}$$

Triangle:

$$x_t = 1.4971x_{t-1} - .4977\, x_{t-2} + a_t - .0180a_{t-1} \text{ X}$$

$$x_t = 1.6836x_{t-1} - .6840\, x_{t-2} + a_t - .2779a_{t-1} \text{ Y}$$

Summarized Hough Transform Results:

Circle:

$$x_t = .9378\ x_{t-1} - .0512x_{t-2} + .0208\, x_{t-3} - .0055x_{t-4} + a_t + 1.0223a_{t-1} + .6001a_{t-2} - 2.6224 * 10^{-6}a_{t-3}$$

Square:

$$x_t = 1.4977\ x_{t-1} - .8875x_{t-2} + .6115x_{t-3} - .2867\, x_{t-4} + a_t - .0063\, a_{t-1} + 1.9988 * 10^{-5}a_{t-2} - 8.6367 * 10^{-9}a_{t-3}$$

Triangle:

$$x_t = .2432x_{t-1} + .6255\,x_{t-2} + .1589x_{t-3} - .0380\,x_{t-4} + a_t + .5676\,a_{t-1} + .0815\,a_{t-2} + 2.6194 * 10^{-4}a_{t-3}$$

In summary, ARMA models were created for 3 shapes for both the x and y. The Hough transforms had ARMA models created as well. They hide information that the user can extract from using intuition on the system. This information could be the saccades of the eye, or the frequency of the eye focusing on and off a point. The models are also useful for real time forecasting. Using the developed models can help reduce the amount of data that needs to be stored to represent these models, lending to faster processing time. Although this thesis only touched on the surface of what is possible using DDS, more can be done in future work.

CHAPTER SIX: EPISCLERAL VENOMANOMETER

i. INTRODUCTION:

The purpose of an episcleral venomanometer (**Error! Reference source not found.**) is to draw information from the episcleral veins located in the eye. The episcleral veins serve as collector channels for the outflow of aqueous humor and therefore an increased pressure in the veins, which is correlated to an elevation of intraocular pressure [63]. The episcleral venomanometer inflates a small air balloon, made from transparent silicone rubber (General Electric RTV 615A), which makes contact with the surface of the eye. The pressure is increased, affecting the air balloon, until the vessel becomes half blanched. The half blanched point is when the color in the vein becomes pale; this is slightly subjective (Figure 19, picture from source [63]). In Figure 19 the EV stands for espicleral vien, the image on top is before the half blanched point and the image on the bottom is when the half blanched point is reached. M is the meniscus of tear film and R is the aiming ring associated with the surgical membrane. The pressure can be read from a dial on the side of the device and is adjusted using an air sealed piston. The pressure range can be accurately read between 5-30 mm Hg, with in vitro reproducibility at 2.4% and in vitro intraobserver reproducibility at .7mm Hg for normal pressure [63].

Figure 18. Episcleral Venomanometer

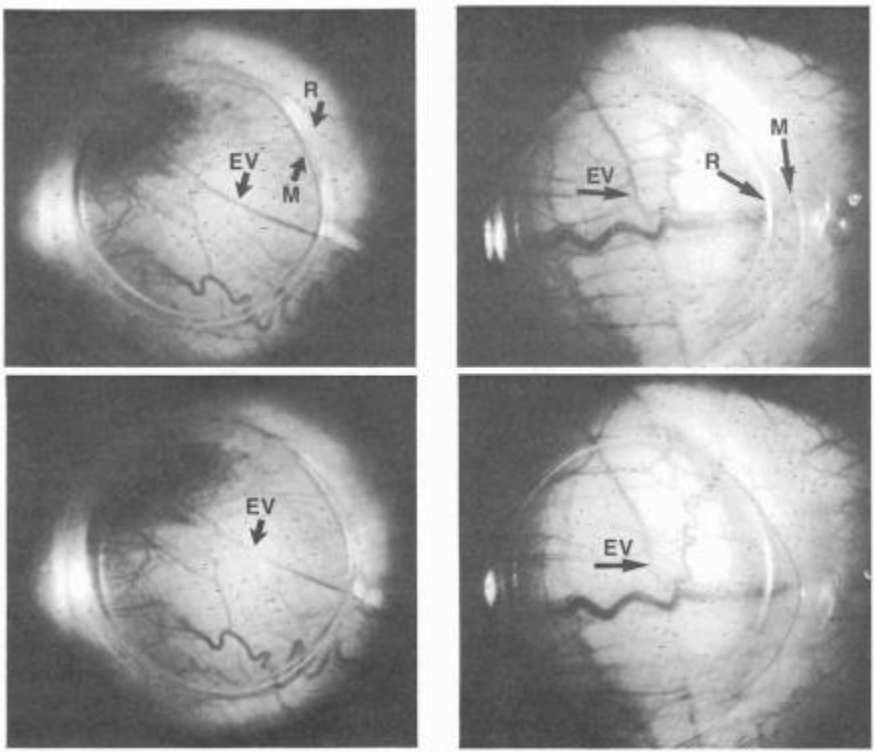


Figure 19. Espiceral Venous Pressure Pictures

Episcleral venous pressure (EVP) can be a sign of many health related issues. A high EVP may indicate glaucomatous damage. High EVP may also be a sign of brain damage or concussions. A portable venomanometer would therefore be useful at sporting events or on a battlefield. The goal of creating an improved episcleral venomanometer is that it can be easily operated by one observer, require little to no calibration, be a compact size, easy to mount, and provide good reproducibility. Current devices attach to a slit lamp which is large and cumbersome. Another issue is getting consistent readings between doctors for identification of the episcleral veins. The episcleral veins are identified are vessels that are typically less mobile, deeper, and straighter than the conjunctival vessels. This makes it easier to find the veins for the measurement.

ii. PROBLEM DESCRIPTION:

A problem with the current technique is that the observer may have repeatability and reproducibility but the difference between two observers may be significant [64]. In a clinical study the interobserver reproducibility test shows that the results of the two observers may differ by 1.2 mm Hg and that overall on average one observer obtained values that were .7 mm Hg higher [64]. The major issue with measuring EVP is that it has an anatomic nature. Between observers they may select different blood vessels to focus on which may create different readings. The reason this occurs is due to the higher pressure to collapse superficial veins along with deep-seated vessels that are located near the exiting of the sclera (see Figure 20).

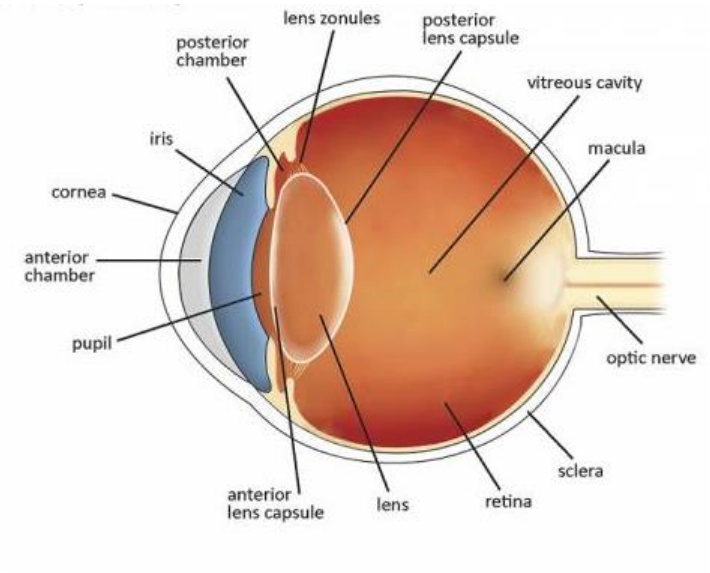


Figure 20. Basic Anatomy of the Human Eye

The other issue is selecting the contact point between the EVP and the eye. This is usually a personal choice, and most doctors choose when it is half blanching at the end point. When you measure past the halfway point the measurements are prone to variability and falsely elevated readings. In order to account for these issues a standard should be set. For refrence look at the difficulty of finding the half blanched point in Figure 19. This standard could be in the form of the exact color for the when the vessel was blanched using methods of computer vision.

iii. SOLUTION:

a. ITERATION 1:

In order to overcome the challenges and requirements of getting consistent readings; an Episcleral Venomanometer EV-310 was obtained from EyeTech. Under the supervision of University of Nebraska Medical Center (UNMC) ophthalmology faculty the device was retrofitted to allow improvements. These improvments were designed in a way to record what measurments the doctor was using when the half blanched point was reached, with the hope of being able to use recorded pressure and images for comparison without the need for large machinery that affected previous recording methods.

The first improvement was the ability to capture images of the eye without needing to use large and expensive equipment. In order to accomplish this goal a Raspberry Pi Zero (Figure 21) was chosen for the computing in a Linux environment. The Raspberry Pi Zero is a small board (65 mm x 30 mm x5 mm), which allows the device to be mounted on the back of the Episcleral Venomanometer. The camera chosen was the Raspberry Pi Camera Board v2 at 8 megapixels. This is a special type of camera that allows direct communication with the Raspberry Pi, while also being quite small. It is capable of 3280x2464 pixel static images and supports video at 1080p 30 fps, 720p 60 fps, and 640x480p and 90 fps.

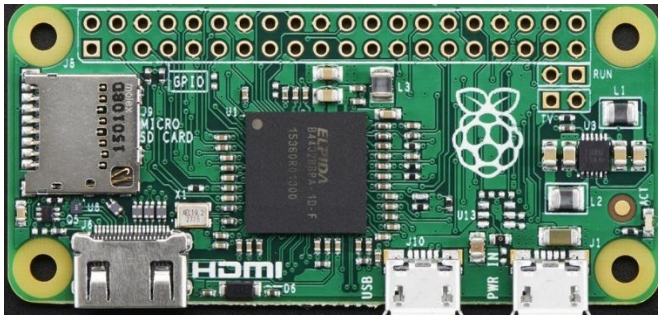


Figure 21: Raspberry Pi Zero

The placement of the camera needed to be in line with the view through the episcleral venomanometer; this obstructed the view point of the user. In order to account for this a touchscreen was added to the back of the episcleral venomanometer to allow live footage of what the camera is seeing. The chosen screen is the Adafruit PiTFT 2.8” touchscreen display (Figure 22); this was chosen for its size and ability to directly communicate with the Raspberry Pi over Serial Peripheral Interface (SPI). A nice feature of this screen is that it allows communication through 4 push buttons and the resistive touchscreen, meaning you don’t need a keyboard and mouse to communicate with the device.

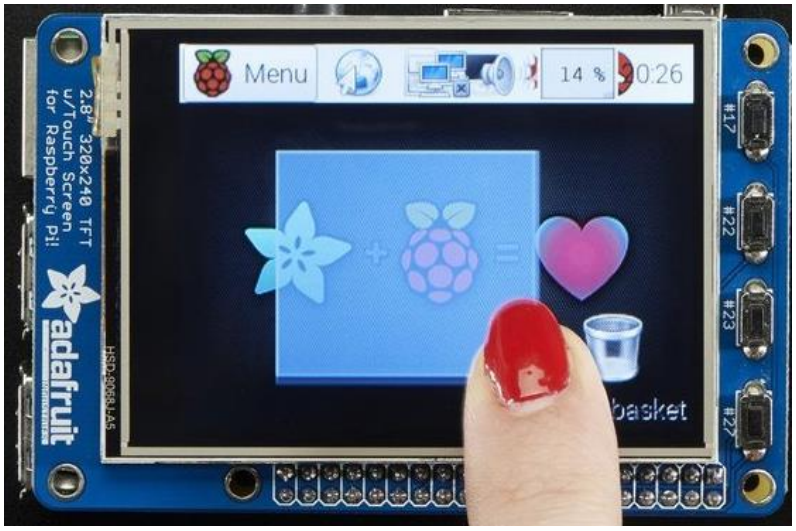


Figure 22: Adafruit PiTFT 2.8" Touchscreen

The add-ons to the device were made from 3D printed plastic and designed to fit to the existing device through the attachment of only one bolt. This would allow for easy assembly and would work on any existing devices without modification.

Originally the device was just configured to take photos that saved to system memory. Also this version had a very large 10X zoom lens that allowed movement of both the camera and lens; (Figure 23). The movement ended up being more than required and the lens was too large in size and without sufficient zoom. Images, however, were clear and the idea was a good proof of concept.

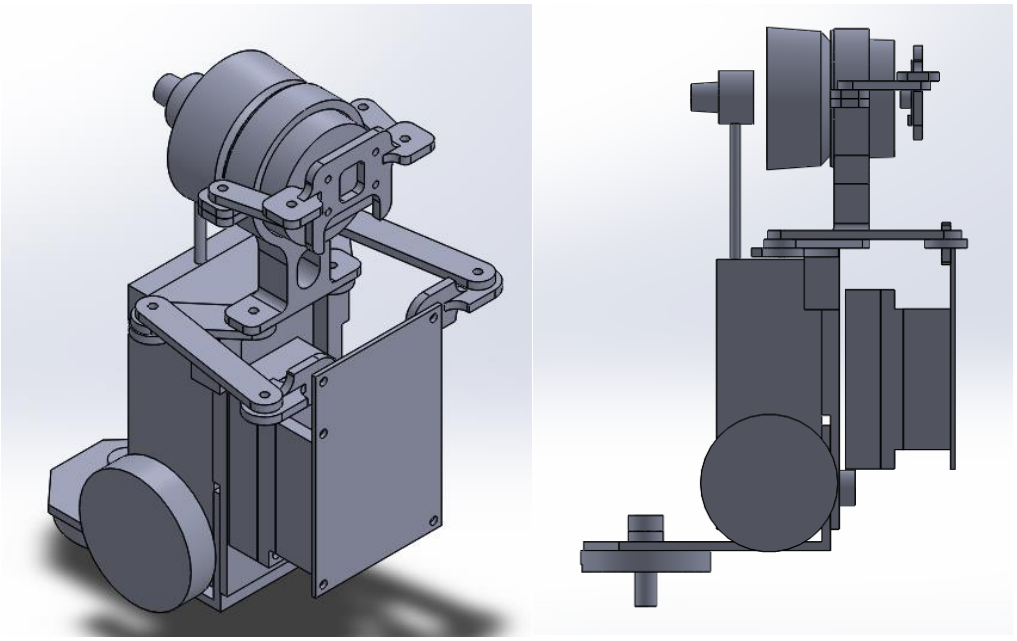


Figure 23. Iteration 1

b. ITERATION 2:

The next iteration aimed to fix the movement of camera and lens to be concentric and allow only small movements axially. To ensure these requirements a new 3D-printed body was made. This new body attached very similarly to iteration 1 with the difference of including some lips that allowed the device to be supported also from the front, Figure 24. This increased overall stability of the base. The electronics bay was also printed to be removable and allow different electronics to be slid onto the back of the device, Figure 25. Easy attachment points were added for future attachments.

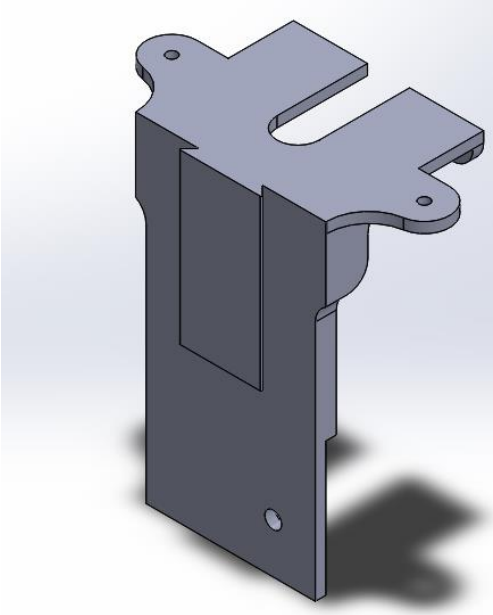


Figure 24: Lipped and Groove Back Plate

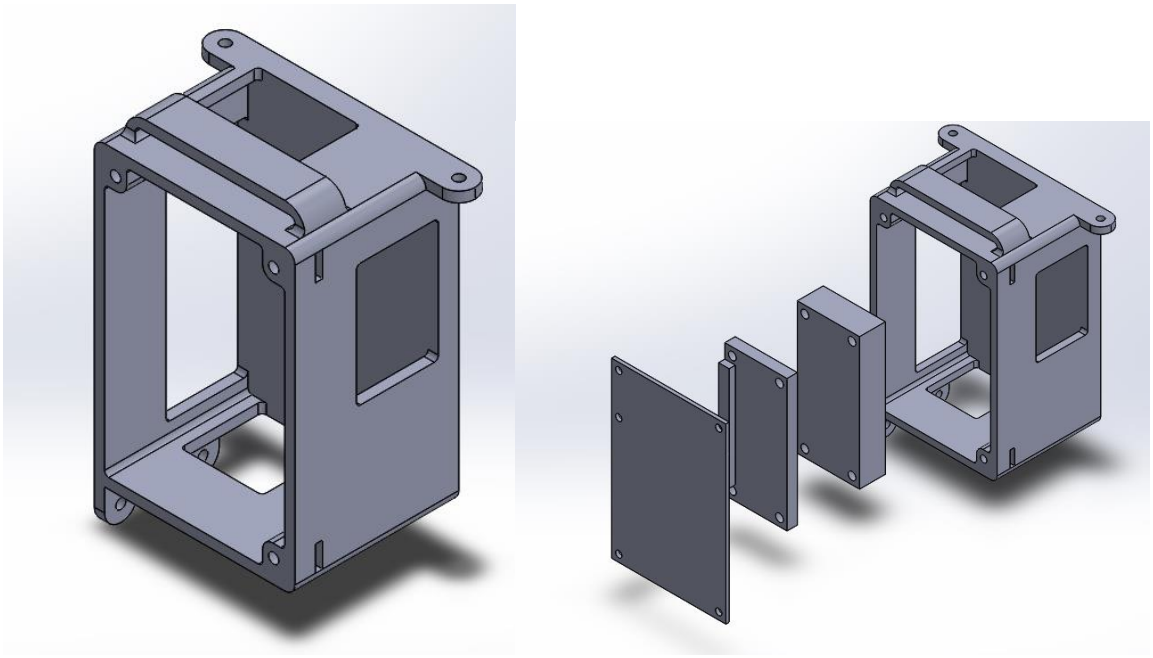


Figure 25: Electronics Bay

The zoom was increased to 25X, and a smaller lens was chosen. This version suffered from problems with both lighting and focus of the camera. At this point the camera was changed from taking pictures to taking video. This change required an external USB drive to store video footage, along with heavy reworking of the programming.

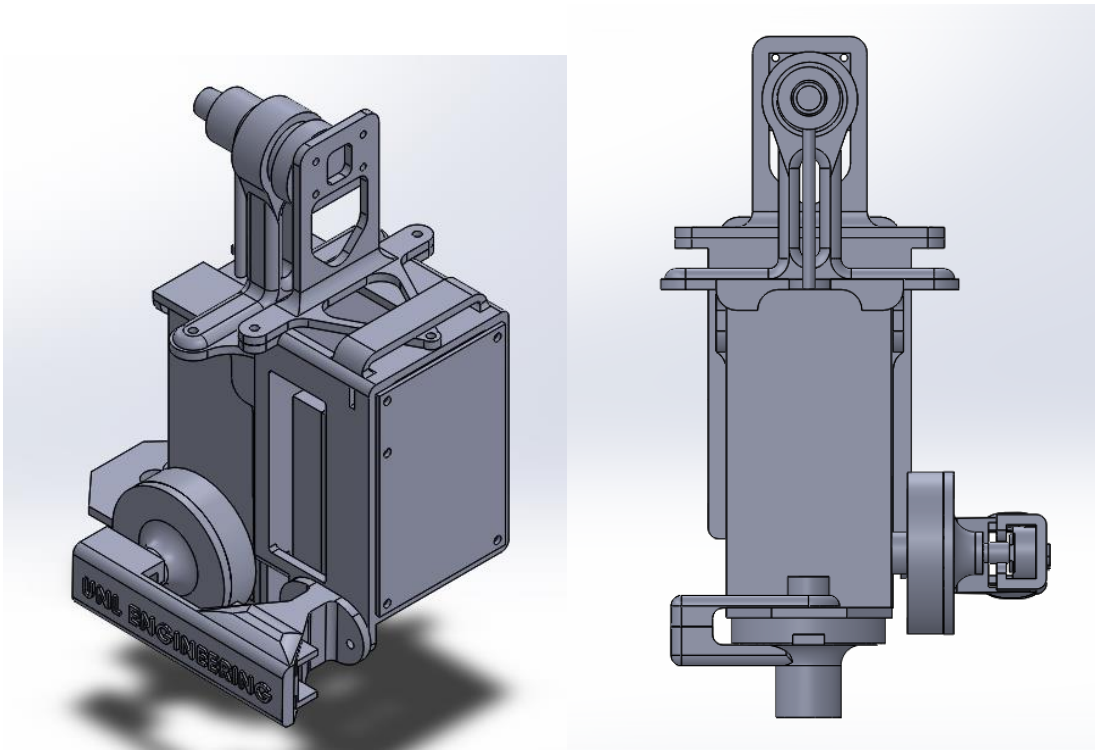


Figure 26: Iteration 2

In these changes a new feature was added, providing a digital readout of what pressure the device was reading. This was done by making an attachment that mounted a potentiometer with the axis of the pressure adjustment wheel, as seen in Figure 27. Functiong like an encoder, when the potentiometer wheel turned, a small custom Arduino read the potentiometer and then using serial communication, told the Raspberry Pi the pressure value. The buttons on the touchscreen were set up to allow easy calibration of the potentiometer to get accurate pressure readings each time. The buttons in order from left to right mean exit program, set low pressure (0 mm Hg), set high pressure (30 mm Hg), and record video.

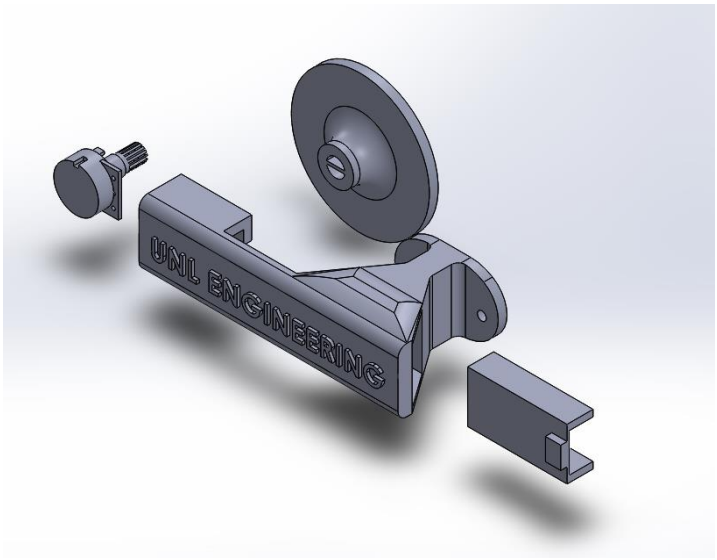


Figure 27. Potentiometer Arm

Another improvement of this iteration was the inclusion of an adapter attachment to a tripod. The stand was a camera tripod that had custom mounting to attach the device via a loc-line tube. Loc-line tube is a plastic modular tube that can hold its shape and be easily moved and configured into new positions, Figure 28. The idea for this tube was to hold the device steady on the tripod while allowing the doctor to move the device into position. The tubing, although powerful, was not strong enough at long lengths. To reinforce the tubes, small bendable metal rods were inserted into the tube. Although this fixed the issue with the tubes being too easily bent, it now made the tubes more difficult to do fine alignment. Another setback was that the wires' added weight created too much moment on the tripod and would cause it to tip. The tripod also proved to be too large and was often in the way during testing.

Figure 28: Loc-Line Tubing

The next issue was lighting. This was a consistent issue throughout the whole project. Too much light would overexpose the camera and make all the images too white. Another issue was that due to the contact of the eye with the inflated balloon and the contact of the camera with the device, when the eye is pressed hard into the device all light is blocked, making the image too dark to see. Through the various different iterations the best lighting has been achieved using a diffused LED light at an angle of 45 to 60 degrees from the contact point with the eye. Due to the requirement that only one person is conducting the measurements; this can create an awkward testing procedure without a slit lamp, as the doctor would need to adjust the light and pressure while keeping the device in place.

c. ITERATION 3:

The next iteration set out to fix some of the issues related to focus and zoom. To address the zoom and focus a pocket microscope with adjustable focus and zoom was purchased. The pocket microscope was a Carson MicroBrite led lighted 20X-40X zoom pocket microscope that was of small form factor, Figure 29. The pocket microscope was then mounted with the camera.

Figure 29. Carson MicroBrite Pocket Microsocpe

The issue with this version was that if the device was shifted off its intended placement, the view through the microscope was in the wrong area, Figure 30. Lighting was good and so was focus when it was in view, but it was hard to keep it consistently in one place with the size of the image on screen. To solve this issue, digital zoom was

simulated by turning off portions of the camera sensor and expanding the resulting image on the screen. This allowed a small improvement in frame-rate without a noticeable difference in perceived quality.

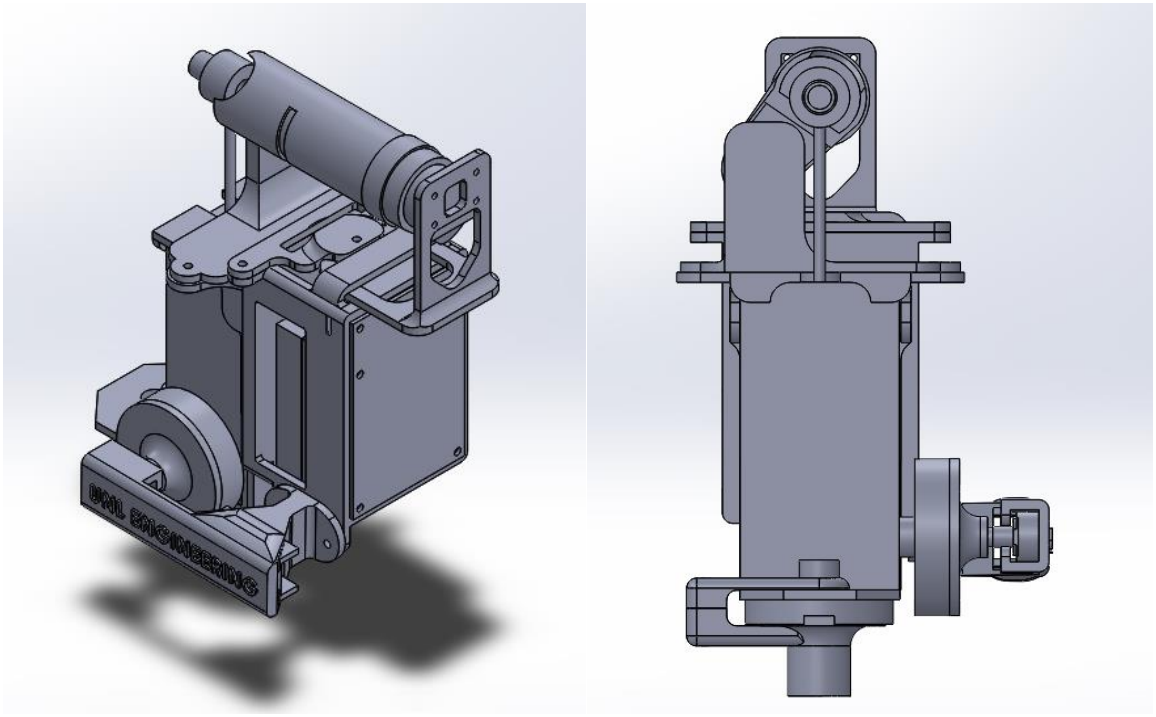


Figure 30. Iteration 3

d. ITERATION 4:

This iteration focused on fixing the stand and keeping the microscope in one place to allow for easier and more consistent viewing. The microscope was disassembled and rebuilt in a custom 3D-printed housing that mounted directly to the frame of the device. This meant that we were now always getting the same view. The stand was replaced with a microphone stand, specifically the JamStands Mic Stand with round base, Figure 31. The round base was chosen over the tripod style base due to tripping hazards and the excess of room the tripod needed would make it difficult to put close to an operating table. A 19" metal gooseneck was chosen to replace the loc-line to provide more stability, Figure 31. The gooseneck advertises the ability to hold up to 2.5 lb at its end, after weighing the device it came in at just 2.2 lb meaning the stand could support it. To support the base of the microphone stand a 9 lb weight was added to the base. The weight was needed more as reassurance to ensure the device not to tip over, even at extreme angles.

Figure 31. JamStand and Gooseneck

This version of the device also incorporated the features of “zoom” and all of the software updates from previous versions while also including a side-mounted light to allow for illumination of the eye. The focus and zoom mechanics were preserved from iteration 3, but both were stiffened to not allow accidental turning of the wheels.

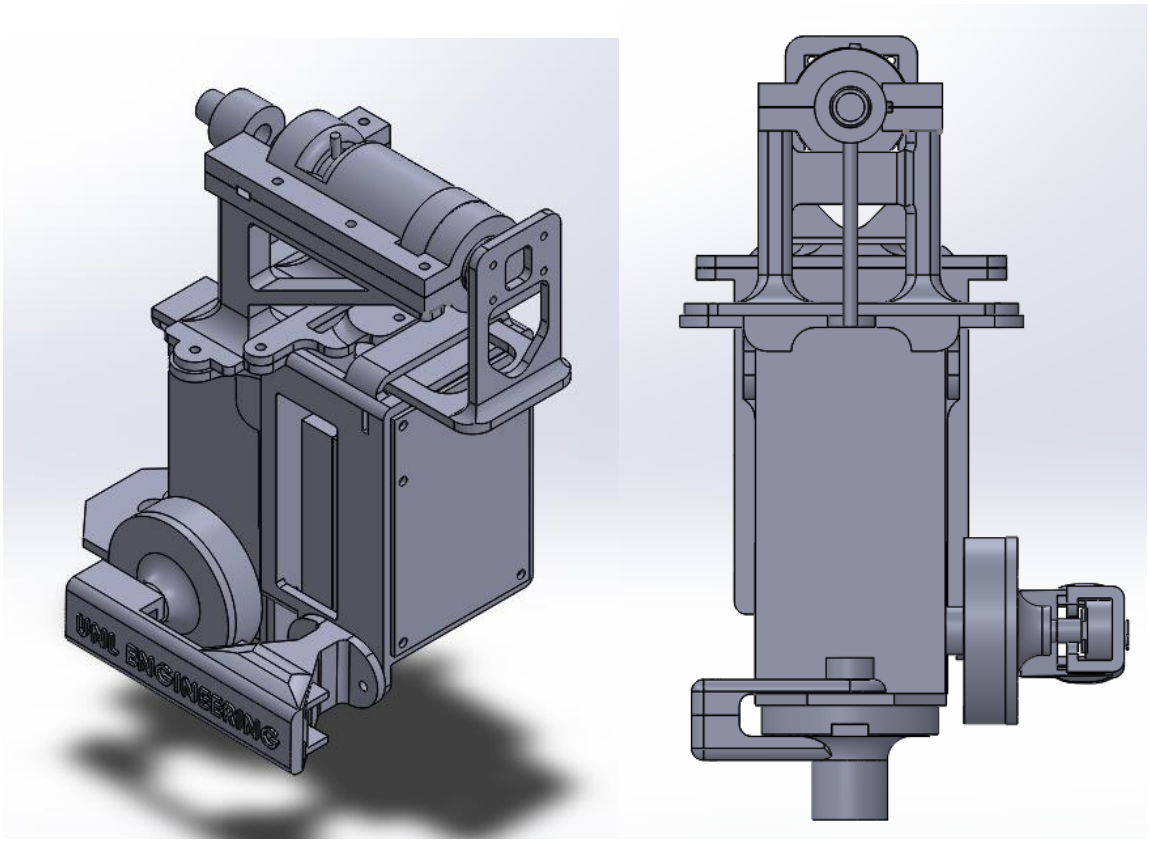


Figure 32. Iteration 4

After testing of this iteration it was discovered that the most useful information is taken from the 0X to around 12X zoom range. This meant that our current device was zooming in too far and becoming blurry for the portions of the eye we wished to view as we were doing around 20X-40X manual zoom with a small amount of “digital zoom” on top of that. Also during testing the correct lighting was discovered to be along the membrane balloon that touches the eye with a direct light source instead of a diffuse light source. More parameters were also decided such as the need for the gooseneck to stop in place after moving. The other critique was that the ball joint near the device allowed for too much movement. The want for an autofocus was also expressed; this is difficult as the

PiCam being used is a fixed focus camera and display using H.264 video file streaming techniques might not allow other cameras to be compatible.

e. ITERATION 5:

The main goal of this iteration was to decide if a new camera was needed or if the PiCam could be reworked to meet the standards set out. After researching compatible cameras it was found that the majority of the compatible cameras were web cameras, with a small subsection of some cameras being USB digital microscopes. Finding a camera that allowed zoom that did not allow manual adjustment was unsuccessful with the exception of a few products that were in the $300-400 range that had little to no documentation of working with the Raspberry Pi. This lead to research on how the rest of the webcams were handling zoom. A majority of the standard webcams were handling the zoom optically with some using a software focus and others using hardware focus. Another caveat of many of these cameras was that the maximum resolution was only 2 MP before using zoom. Knowing the other cameras were only going to be at 2 megapixels lead to the reconsideration of using the PiCam, which has a Sony IMX219 8 megapixel sensor, for its higher resolution.

Using the original PiCam, the microscope and extra lens were removed. The digital zoom was adjusted in software to allow a more customized zoom using the following function (Eq. 10).

camera.crop(x,y,w,h) (10)

where x and y are the location of the start of a rectangle of width w and height h. All parameters are between 0 and 1. To achieve true zoom Eq. 10 is modified and the following formula is used.

camera.crop(z,z,1-2*z,1-2*z), where z is the percentage of zoom.

This formula was used to get the camera zoom to be useful for the 0-12X zoom that was required. A small shift in the x and y position was also implemented to focus on a certain portion of the screen. The focus of the camera still being fixed needed to be at a distance away to still allow the image to be in focus. The PiCam V2 has a lens of f=3.0 mm, f/2.0 with a field of view of 62.2 x 48.8 degrees. The full frame SLR lens equivalent is 29 mm; [65]. The lens can be adjusted for better focus by carefully scrapping off the glue that holds the threads of the lens and turning the lens assembly clockwise. This allows focusing as close as 25 mm or .94252 inches with the side effect of becoming a wide angle lens. This adjustment was not needed as simply moving the lens farther by a few centimeters adjusted the focus enough to be adequate.

Improvements were also done to the stand. The gooseneck was supported with rubber heat shrink tubing. Also the stand height was shortened to allow easier access to a patient on a table. To shorten the stand height the orginal microphone stand was cut with a Dremel tool and then placed back together. Iteration 5 was successful and more simple than other iterations; (see Figure 33 and Figure 34). Lighting was included via a flexiable reading light.

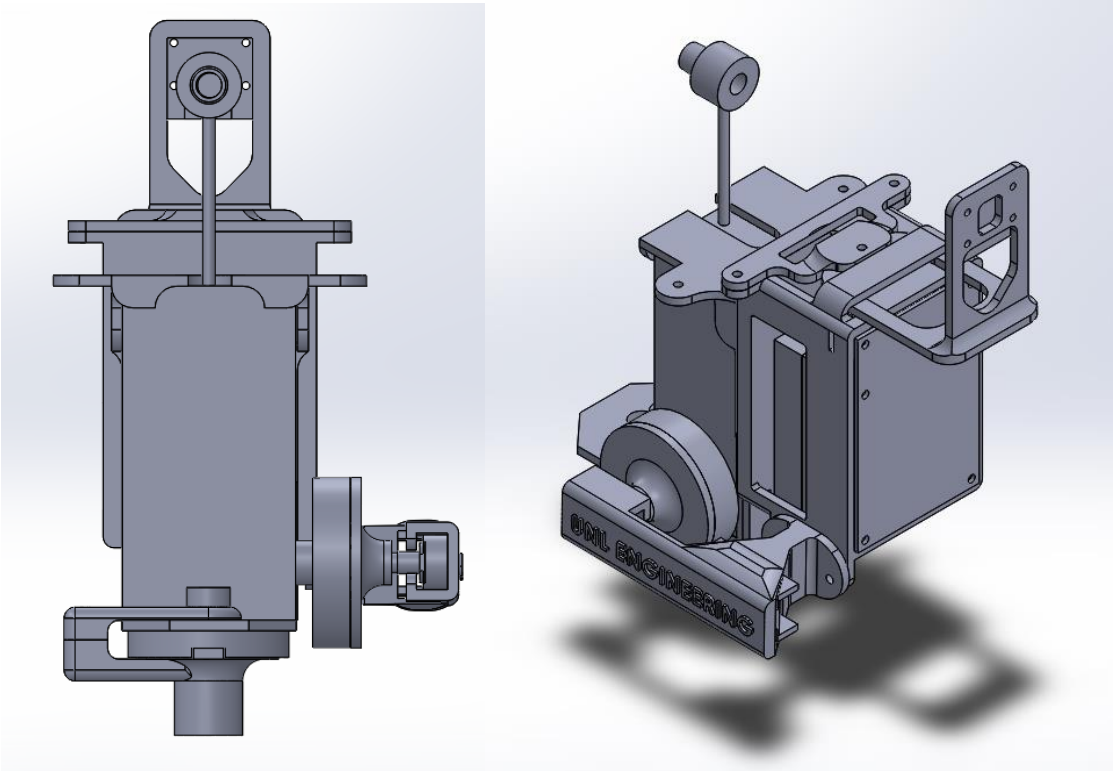


Figure 33. Iteration 5

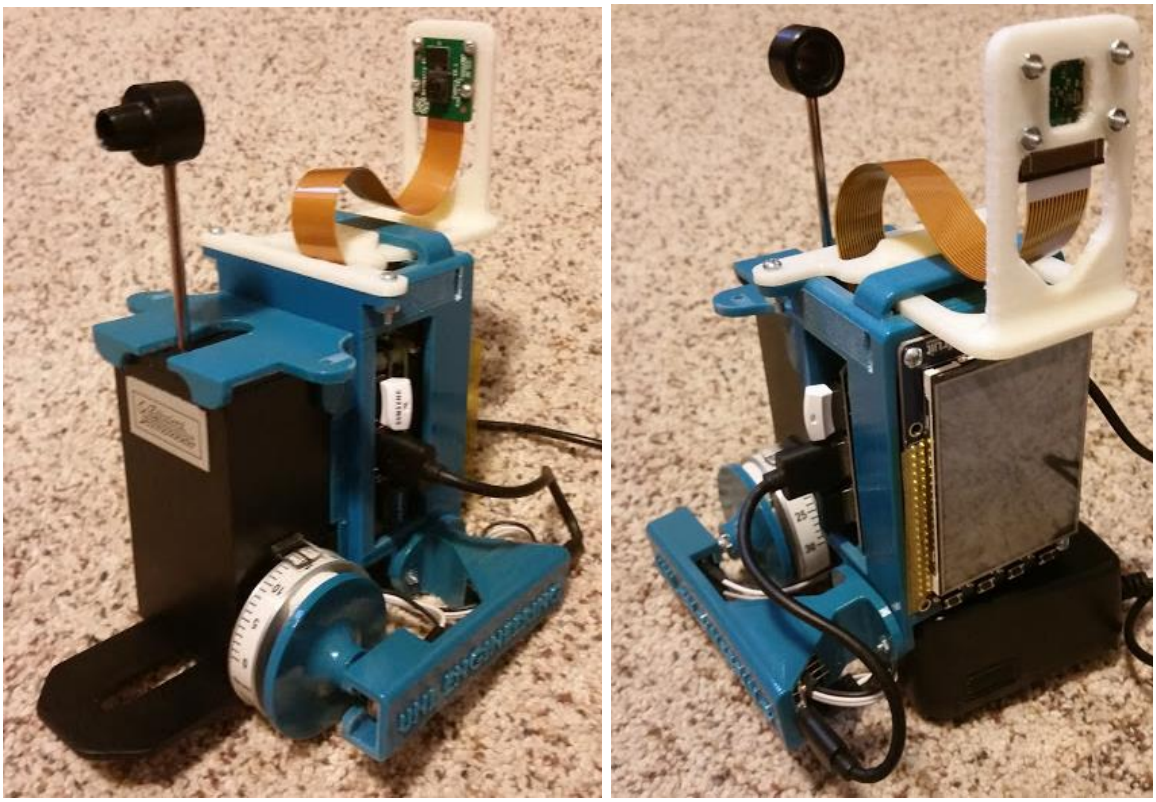


Figure 34. Iteration 5 Physical Model

CHAPTER SEVEN: CONCLUSIONS

i. CONCLUSION OF GAZE VECTOR SHAPE BASED RECOGNITION

In this thesis, the concept of gaze-based HRI was expanded to allow the inclusion of shape-based command recognition. The experiments showed that the detection of mouse-drawn shapes and eye gaze-drawn shapes were comparable. The method was validated on a small humanoid robot with a set of shape-based commands, as well as on a custom-built wheeled robot. Eye-based commands will prove to be beneficial in interfacing with robots. This approach has many advantages over blink-based command input. Blinks can be difficult to count and easily confused, whereas shapes are more natural and can vary in size for the intensity of the command. This system is easily accessible for disabled and elderly people who may lack the motor skills necessary to control technology by other means. By combining the gaze-based shape method with blinks for program indicators, the amount of commands and interactions grow to provide greater command options and independence for the user. Another option for adding more commands is the location of the starting corner of a shape. For a square that would allow 4 different starting locations, and with the addition of drawing clockwise or counterclockwise with size differnces, allows for a wide variety of shape possibilities from only a small amount of shape based patterns.

More users and results are being processed to allow a large database of user shapes to be logged. From these logged results different techniques and specifications can be tested on large populations of data. Machine learning or Taguchi method may be

implemented to determine the best tuning procedure. Future work will include testing with impaired subjects, along with testing procedures to properly adjust weights in the algorithm for each subject.

ii. CONCLUSION OF DESIGN OF AN IMPROVED EPISCLERAL VENOMANOMETER:

In conclusion the design of an improved episcleral venomanometer took many iterations. The cross discipline project required direct interation between ophthalmologist, researchers, and engineering. Although the parameters and requirements changed and adapted as the project went on, further knowledge on what was required and needed in a typical operation was gained. The final product is only a small fraction of the cost of the full device, as seen in Table 22. For only $263.54 any episcleral venomanometer can be easily improved to allow live recording of both pressure and what the surgeon sees. When the device itself is already $950.00 and the alternatives require large bulky machinery, this is a relative bargain. Assembly to the device is easy and only required one existing screw to attach the electronics.

The success of iteration 5 lead to the creation of an identical device for further subject testing. New footage will be recorded and there is the possibility to have greater accuracy and understanding of the episcleral pressure with the video recordings.

Table 22. Bill of Materials for Venomanometer Iteration 5

Name	Producer	Price	Quanity	Total Price
Electronics:				
Raspberry Pi Zero - Version 1.3	Adafruit	$5.00	1	$5.00
32Gb FlashDrive	Samsung	$10.99	1	$10.99
16 Gb Micro SD	SandDisk	$8.24	1	$8.24
Raspberry Pi Zero v1.3 Camera Cable	Adafruit	$5.95	1	$5.95
Raspberry Pi Camera Board v2 - 8 Megapixels	Adafruit	$29.95	1	$29.95
Short USB Cable 15cm	CableCreation	$4.99	1	$4.99
5V 2.4A Switching Power Supply w/ 20AWG 6' MicroUSB Cable	Adafruit	$7.95	1	$7.95
Zero4U - 4 Port USB Hub for Raspberry Pi Zero v1.3	Adafruit	$9.95	1	$9.95
USB Wifi Adapter	Edimax	$7.99	1	$7.99
Innovation Board	UNL Makers Club	$9.00	1	$9.00
PiTFT Plus Assembled 320x240 2.8" TFT + Resistive Touchscreen	Adafruit	$34.95	1	$34.95
Total Electronics Cost				$134.96
Construction:				
Coolant Hose Connector 3/4"	Loc-Line	$6.42	1	$6.42
LDR 3/4-in dia Black Iron Cap Fitting	Lowe's	$1.74	1	$1.74
JS-MSCRB100 Mic Stand	JamStands	$27.99	1	$27.99
19" Gooseneck	OnStage	$7.19	1	$7.19
#4 3/4" Bolts	Lowe's	$1.25	1	$1.25
Gardner Bender 22.2mm 6-in Heat Shrink Tubing	Lowe's	$3.96	1	$3.96
10 lb Weight	Cap Barbell	$8.42	1	$8.42
3D Printing Cost	Stratasys Mojo	$5.50	13.02	$71.61
Total Construction Cost				$128.58
Surgical Instuments:				
Cost of Episcleral Venomanometer	EyeTech	$950.00	1	$950.00
Total Cost of Electronics and Construction				$263.54
Total Cost of Device				$1,213.54

ACKNOWLEDGEMENT:

I would like to thank my advisor Dr. Nelson. Dr. Nelson has shown time and time again that he is committed to making, creating, and inspiring students to reach new heights of academic excellence. He does this in all aspects of life, both in and out of the classroom. He helped me do my best throughout my academic career. Through his help we were able to submit a paper and present at the IEEE/ASME MESA 2016 conference in Auckland, New Zealand. Much of the content from that paper made its way into my thesis. These sort of opportunities and professional advancement were things that I would not have been able to achieve without his expertise in both the field, and his continual involvement in me as a student. I am so thankful for what he has contributed, and am excited to see the greatness that he will continue to inspire in others.

I would also like to thank the NSF for the funding that went into the eye gaze shape based recognition research, as well as the NRI funding and UNMC involvement in the research and study of an improved Episcleral Venomanometer.

Works Cited

[1] M. Bollini, S. Tellex, T. Thompson, N. Roy and D. Rus, "Interpreting and Executing Recipes with a Cooking Robot," *In Experimental Robotics,* pp. 481-495, 2013.

[2] M. Beetz, U. Klank, I. Kresse, A. Maldonado, L. Mosenlechner, D. Pangercic, T. Ruhr and M. Tenorth, "Robotic Roommate Making Pancakes," *11th IEEE-RAS International Conference on Humanoid Robots (Humanoids),* pp. 529-536, 2011.

[3] S. Miller, J. V. D. Berg, M. Fritz, T. Darrell, K. Goldberg and P. Abbeel, "A Geometric Approach to Robotic Laundry Folding," *The International Journal of Robotics Research,* vol. 32, no. 2, pp. 249-267, 2012.

[4] A. Ramisa, G. Alenya, F. Moreno-Noguer and C. Torras, "Using Depth and Appearance Features for Informed Robot Grasping of Highly Wrinkled Clothes," *IEEE International Conference on Robotics and Automation (ICRA),* pp. 1703-1708, 2012.

[5] M. Ciocarlie, K. Hsiao, A. Leeper and D. Gossow, "Mobile Manipulation through an Assistive Home Robot," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* pp. 5313-5320, 2012.

[6] H. Nguyen, A. Jain, C. Anderson and C. C. Kemp, "A Clickable World: Behavior Selection through Pointing and Context for Mobile Manipulation," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* pp. 787-793, 2008.

[7] C.-H. King, T. L. Chen, A. Jain and C. C. Kemp, "Towards an Assistive Robot that Autonomously Performs Bed Baths for Patient Hygiene," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* pp. 319-324, 2010.

[8] S. K. Banala, S. H. Kim, S. K. Agrawal and J. P. Scholz, "Robot Assisted Gait Training with Active Leg Exoskeleton (ALEX)," *IEEE Transactions on Neural Systems and Rehabilitation Engineering,* vol. 17, no. 1, pp. 2-8, 2009.

[9] P. Malcolm, W. Derave, S. Galle and D. D. Clercq, "A Simple Exoskeleton that Assists Plantarflexion can Reduce the Metabolic Cost of Human Walking," *PloS one,* vol. 8, no. 2, 2013.

[10] R. Sparrow and L. Sparrow, "In the Hands of Machines? The Future of Aged Care," *Minds*

and Machines, vol. 16, no. 2, pp. 141-161, 2006.

[11] Y. S. Choi, "A Study of Human-Robot Interaction with an Assistive Robot to Help People with Severe Motor Impairments," 2009.

[12] A. Jain and C. C. Kemp, "EL-E: An Assistive Mobile Manipulator that Autonomously Fetches Objects from Flat Surfaces," *Autonomous Robots,* vol. 28, no. 1, pp. 45-64, 2010.

[13] L. P. Reis, R. A. Braga, M. Sousa and A. P. Moreira, "A Flexible Interface for an Intelligent Wheelchair," *Robot Soccer World Cup,* pp. 296-307, 2009.

[14] Researchnester.com, "Mechanical Keyboard & Switch Market : Global Demand Analysis & Opportunity Outlook 2023," ICT & Electronics, 1 Febuary 2017. [Online]. Available: http://www.researchnester.com/reports/mechanical-keyboard-switch-market-global-demand-analysis-opportunity-outlook-2023/190. [Accessed 6 June 2017].

[15] R. Mead and M. J. Mataric, "Toward Robot Adaptation of Human Speech and Gesture Parameters in a Unified Framework of Proxemics and Multimodal Communication," *IEEE International Conference on Robotics and Automation (ICRA) Workshop on Machine Learning for Social Robots,* 2015.

[16] A. Aly and A. Tapus, "A Model for Synthesizing a Combined Verbal and Nonverbal Behavior based on Personality Traits in Human-Robot Interaction," *8th ACM/IEEE International Conference on Human-robot interaction,* pp. 325-332, 2013.

[17] K. Anderson and P. W. McOwan, "A Real-Time Automated System for the Recognition of Human Facial Expressions," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics),* vol. 36, no. 1, pp. 96-105, 2006.

[18] J. Saldien, K. Goris, B. Vanderborght, J. Vanderfaeillie and D. Lefeber, "Expressing Emotions with the Social Robot Probo," *International Journal of Social Robotics ,* vol. 2, no. 4, pp. 277-289, 2010.

[19] M. V. d. Bergh, D. Carton, R. D. Nijs, N. Mitsou, C. Landsiedel, K. Kuehnlenz, D. Wollherr, L. V. Gool and M. Buss, "Real-Time 3D Hand Gesture Interaction with a Robot for Understanding Directions from Humans," *IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN),* pp. 357-362, 2011.

[20] L. Sartori, C. Becchio and U. Castiello, "Cues to Intention: The Role of Movement Information," *Cognition,* vol. 119, no. 2, pp. 242-252, 2011.

[21] P. K. Artemiadis and K. J. Kyriakopoulos, "EMG-based Control of a Robot Arm Using Low-Dimensional Embeddings," *IEEE Transactions on Robotics,* vol. 26, no. 2, pp. 393-398, 2010.

[22] N. Bu, M. Okamoto and T. Tsuji, "A Hybrid Motion Classification Approach for EMG-based Human–Robot Interfaces Using Bayesian and Neural Networks," *IEEE Transactions on Robotics,* vol. 25, no. 3, pp. 502-511, 2009.

[23] L. Bi, X.-A. Fan and Y. Liu, "EEG-based Brain-Controlled Mobile Robots: A Survey," *IEEE Transactions on Human-Machine Systems,* vol. 43, no. 2, pp. 161-176, 2013.

[24] F. Galan, M. Nuttin, W. Lew, P. W. Ferrez, G. Vanacker, J. Philips and J. D. R. Millan, "A Brain-Actuated Wheelchair Asynchronous and Non-invasive Brain-Computer Interface for Continuous Control of Robots," *Clinical Neurophysiology,* vol. 119, no. 9, pp. 2159-2169, 2008.

[25] marketsandmarkets.com, "Eye Tracking Market by Type (Mobile & Remote), by Application (Medical Diagnostics, HCI, Research, & Virtual Reality), by Industry (Marketing, Healthcare, Transportation, Communication & Entertainment) and by Geography - Global Trend & Forecast to 2020," marketsandmarkets.com, November 2015. [Online]. Available: http://www.marketsandmarkets.com/Market-Reports/eye-tracking-market-144268378.html. [Accessed 6 June 2017].

[26] D. Cox and J. DiCarlo, "Device and Method for Tracking Eye Gaze Direction". United States of America Patent 11/386.878.

[27] C. H. Morimoto and M. R. Mimica, "Eye Gaze Tracking Techniques for Interactive Applications," *Computer Vision and Image Understanding,* vol. 98, no. 1, pp. 4-24, 2005.

[28] Z. Zhu and Q. Ji, "Eye and Gaze Tracking for Interactive Graphic Display," *Machine Vision and Application,* vol. 15, pp. 139-148, 2004.

[29] K. R. Park, "A Real-Time Gaze Position Estimation Method based on a 3-D Eye Model," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics,* vol. 37, pp. 199-212, 2007.

[30] E. D. Guestrin and M. Eizenman, "General Theory of Remote Gaze Estimation Using the Pupil Center and Corneal Reactions," *IEEE Transactions on Biomedical Engineering,* vol. 53, pp. 1124-1133, 2007.

[31] R. Barea, L. Boquete, L. M. Bergasa, E. Lopez and M. Mazo, "Electro-Oculographic Guidance of a Wheelchair Using Eye Movements Codification," *The International Journal of Robotics*

Research, vol. 22, no. 7-8, pp. 641-652, 2003.

[32] E. A. Hoffman and J. V. Haxby, "Distinct Representations of Eye Gaze and Identity in the Distributed Human Neural System for Face Perception," *Nature Neuroscience,* vol. 3, no. 1, pp. 80-84, 2000.

[33] N. J. Emery, "The Eye Have It: The Neuroethology, Function and Evolution of Social Gaze," *Neuroscience & Biobehavioral Reviews,* vol. 24, no. 6, pp. 581-604, 2000.

[34] K. K. Kampe, C. D. Frith, R. J. Dolan and U. Frith, "Psychology: Reward Value of Attractiveness and Gaze," *Nature,* vol. 413, no. 6856, pp. 589-589, 2001.

[35] B. Fink and I. Penton-Voak, "Evolutionary Psychology of Facial Attractiveness," *Current Directions in Psychological Science,* vol. 11, no. 5, pp. 154-158, 2001.

[36] R. Jacob and K. S. Kam, "Eye Tracking in Human-Computer Interaction and Usability Research: Ready to Deliver the Promises," *Mind,* vol. 2, no. 3, p. 4, 2003.

[37] J. H. Goldberg and X. P. Kotval, "Computer Interface Evaluation using Eye Movements: Methods and Constructs," *International Journal of Industrial Ergonomics,* vol. 24, no. 6, pp. 631-645, 1999.

[38] T. W. Victor, J. L. Harbluk and J. A. Engstrom, "Sensitivity of Eye-Movement Measures to in-Vehicle Task Difficulty," *Transportation Research Part F: Traffic Psychology and Behaviour,* vol. 8, no. 2, pp. 167-190, 2005.

[39] N. B. Sarter, R. J. Mumaw and C. D. Wickens, "Pilots' Monitoring Strategies and Performance on Automated Flight Decks: An Empirical Study Combining Behavioral and Eye-Tracking Data," *Human Factors: The Journal of the Human Factors and Ergonomics Society,* vol. 49, no. 3, pp. 347-457, 2007.

[40] C. S. Lin, C. Ho, W. Chen, C. Chiu and M. Yeh, "Powered Wheelchair Controlled by Eye-Tracking System," *Optica Applicata,* vol. 36, no. 2-3, p. 401, 2006.

[41] P. S. Gajwani and S. A. Chhabria, "Eye Motion Tracking for Wheelchair Control," *International Journal of Information Technology,* vol. 2, pp. 185-187, 2006.

[42] J. Ma, Y. Zhang, A. C. A and F. Matsuno, "A Novel EOG/EEG Hybrid Human-Machine Interface Adopting Eye Movements and ERPS: Application to Robot Control," *IEEE Transactions on Biomedical Engineering,* vol. 62, no. 3, pp. 876-889, 2015.

[43] B. H. Kim, M. Kim and S. Jo, "Quadcopter Fight Control Using a Low-Cost Hybrid Interface with EEG-based Classification and Eye Tracking," *Computers in Biology and Medicine,* vol. 51, pp. 85-92, 2014.

[44] C. Staub, S. Can, B. Jensen, A. Knoll and S. Kohlbecher, "Human-Computer Interfaces for Interaction with Surgical Tools in Robotic Surgery," *4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob),* pp. 81-86, 2012.

[45] N. P. Noonan, G. P. Mylonas, J. Shang, C. J. Payne, A. Darzi and G. Z. Yang, "Gaze Contingent Control for an Articulated Mechatronic Laparoscope," *3rd IEEE RAS and EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob),* pp. 759-764, 2010.

[46] X. Zhang, S. Li, J. Zhang and H. Williams, "Gaze Contingent Control for a Robotic Laparoscope Holder," *Journal of Medical Devices,* vol. 7, no. 2, p. 020915.

[47] S. Li, J. Zhang, L. Xue, F. J. Kim and X. Zhang, "Attention-Aware Robotic Laparoscope for Human-Robot Cooperative Surgery," *IEEE International Conference on Robotics and Biomimetics (ROBIO),* pp. 792-797, 2013.

[48] S. Li, X. Zhang, F. J. Kim, R. D. d. Silva, D. Gustafson and W. R. Molina, "Attention-Aware Robotic Laparoscope based on Fuzzy Interpretation of Eye-Gaze Patterns," *Journal of Medical Devices,* vol. 9, no. 4, p. 041007, 2015.

[49] C. A. Nelson, X. Zhang, J. Webb and S. Li, "Fuzzy Control for Gaze-Guided Personal Assistance Robots: Simulation and Experimental Application," *International Journal on Advances in Intelligent Systems,* vol. 8, no. 1-2, pp. 77-84, 2015.

[50] D. Puanhvuan and Y. Wongsawat, "Semi-automatic P300-based brain-controlled wheelchair," *ICME International Conference on Complex Medical Engineering (CME), Kobe, Japan,* pp. 455-460, 2012.

[51] P. P. Caffier, U. Erdmann and P. Ullsperger, "Experimental evaluation of eye-blink parameters as a drowsiness measure," *European Journal of Applied Physiology,* vol. 89, no. 3-4, pp. 319-325, 2003.

[52] G. Cardona and N. Quevedo, "Blinking and driving: the influence of saccades and cognitive workload," *Current Eyes Research,* vol. 39, no. 3, pp. 239-244, 2014.

[53] K. Grauman, M. Betke, J. Gips and G. R. Bradski, "Communication via eye blinks - detection and duration analysis in real time," *IEEE Computer Society Conference on Computer Vision*

and Pattern Recognition (CVPR), vol. 1, pp. 1010-1017, 2001.

[54] "Gazepoint GP3 Eye Tracker- Gazepoint," Gazepoint, [Online]. Available: http://www.gazept.com/product/gazepoint-gp3-eye-tracker/. [Accessed 6 April 2016].

[55] B. Li, B. Mettler and J. Andersh, "Classification of Human Gaze in Spatial Guidance and Control," *IEEE International Conference on Systems, Man, and Cybernetics (SMC) in Kowloon, China,* pp. 1073-1080, 2015.

[56] "Find out more about NAO," Aldebaran Robotics, [Online]. Available: https://www.aldebaran.com/en/cool-robots/nao/find-out-more-about-nao. [Accessed 4 April 2016].

[57] "Recognizing objects — NAO Software 1.14.5 documentation," Aldebaran Robotics, [Online]. Available: http://doc.aldebaran.com/1-14/software/choregraphe/tutos/recognize_objects.html. [Accessed 4 April 2016].

[58] C. Harris and M. Stephens, "A combined corner and edge detector," *Alvery Vision Conference,* vol. 15, p. 50, 1988.

[59] J. Illingworth and J. Kittler, "A survey of the Hough transform," *Computer Vision Graphics Image Process,* vol. 44, no. 1, pp. 87-116, 1988.

[60] S. M. Pandit and K. P. Rajurkar, "Data-Dependent Systems Approach to Solar Energy Simulation Inputs," *Journal of Solar Energy Engineering,* vol. 105, pp. 461-463, 1983.

[61] K. P. Rajurkar and J. L. Nissen, "Data-Dependent Systems Approach to Short-Term Load Forecasting," *IEEE Transaction on Systems, Man, and Crybernetics,* Vols. SMC-15, no. 4, p. 532, 1985.

[62] H. Xin, J. A. DeShazer, J. J. R. Feddes and K. P. Rajurkar, "Data Dependent Systems Analysis of Stochastic Swine Energetic Responses," *J therm Biol,* vol. 17, no. 4.5, pp. 225-234, 1992.

[63] M. Mori, *Episcleral Venomanometer EV-310,* Boca Raton, Florida: EYETECH LTD., 2016.

[64] R. C. Zeimer, D. K. Gieser, J. T. Wilensky, J. M. North, M. M. Mori and E. E. Odunukwe, "A Practical Venomanometer: Measurment of Episcleral Venous Pressure and Assessment of the Normal Range," *Arch Ophthalmol,* vol. 101, pp. 1447-1449, 1983.

[65] "Rpi Camera Module," elinux.org, 11 July 2017. [Online]. Available: http://elinux.org/Rpi_Camera_Module#Technical_Parameters_.28v.1_board.29.

APPENDIX 1:

Stocachstic Equations:

The roots of the equation can be found as follows

$$(1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_n B^n) = (1 - \lambda_1 B)(1 - \lambda_2 B) \ldots (1 - \lambda_n B)$$

$\lambda_i = e^{\mu i \Delta}$, i=1,2,….n, where n is the number of data points

with the complex conjugate pair of roots giving the natural frequency and damping ratio.

$$\mu_1, \mu_2 = -\zeta\omega_n \pm j\omega_n\sqrt{1 - \zeta^2}$$

A spring-damper system can be represented as follows which can help give the required parameters for a dynamic system.

$$(D^2 + 2\zeta\omega_n D + \omega_n^2)x(t) = \frac{1}{M}f(t)$$

The dynamics of the difference equation model are seen in the Green's function:

$$G_j = g_1\lambda_1^j + g_2\lambda_2^j + \cdots g_n\lambda_n^j$$

with the distinct roots of λ_i given in the coefficients g_k calculated by

$$g_k = \frac{(\lambda_k^{n-1} - \lambda_k^{n-2}\theta_1 - \lambda_k^{n-3}\theta_2 - \cdots - \theta_{n-1})}{\prod_{i=1, i\neq k}^{n}(\lambda_k - \lambda_i)}$$

The Green's function can also be expressed as:

$$x_t = \sum_{j=0}^{\infty} G_j a_{t-j}$$

To compare multiple models against each other the RSS and F0 and Fcrit are calculated and used in the F-test, explained below.

Comparing ARMA(2n,2n-1) model with ARMA(2n+2,2n+1)

Hypothesis: $H_0 = \phi_{2n+1} = 0 = \phi_{2n+2} = \cdots = \theta_{2n} = \theta_{2n+1}$ …

$F = \frac{A_1 - A_0}{A_0/(N-\gamma)}$: F(s,N-γ)

N=# of observations or data

A_1=Sum of Squares of Errors (SSE) of lower order model

A_0=SSE of higher order model

F(s,N-γ) →F distribution with S and N- γ DOF

γ =total # of parameters to be estimated for higher order model

s=Difference in # of parameters of two models

The equations of a spring-damper system fit closely with the base A(2) model equation. The A(2) model can then be expressed similar within the spring-damper equations to helps better describe the parameters.

$$x_t = 1.4518x_{t-1} - .4528x_{t-2} + a_t - 3.2289 * 10^{-5}a_{t-1}$$

$$(D^2 + 2\zeta\omega_n D + \omega_n^2)x(t) = \frac{1}{M}f(t)$$

$$u_1, u_2 = -\zeta\omega_n \pm \omega_n\sqrt{\zeta^2 - 1}$$

$$G(t) = \frac{e^{u_1 t} - e^{u_2 t}}{u_1 - u_2}$$

$$\gamma(s) = \frac{\sigma_z^2}{2u_1u_2(u_1^2 - u_2^2)}[u_2e^{u_1 s} - u_1e^{u_2 s}], where\ s = \Delta K$$

$$\gamma(0) = \frac{\sigma_z^2}{4\omega_n^3\zeta}$$

VarArma(2,1) $\gamma_k = d_1\lambda_1^k + d_2\lambda_2^k$

$$d_1 = \frac{\sigma_z^2}{2u_1(u_1^2 - u_2^2)}$$

$$\lambda_1 = e^{u_1\Delta}$$

$$d_2 = \frac{-\sigma_z^2}{2u_2(u_1^2 - u_2^2)}$$

$$\lambda_2 = e^{u_2\Delta}$$

$$\phi_1 = \lambda_1 + \lambda_2 = e^{u_1\Delta} + e^{u_2\Delta}$$

$$\phi_2 = -\lambda_1\lambda_2 = e^{(u_1+u_2)\Delta}$$

$$\theta_1^2 + 2P\theta_1 + 1 = 0$$

$$\theta_1 = -P \pm \sqrt{P^2 - 1}$$

$$P = \frac{-u_1(1+\lambda_1^2)(1-\lambda_2^2) + u_2(1+\lambda_2^2)(1-\lambda_1^2)}{u_1\lambda_1(1-\lambda_2^2) - u_2\lambda_2(1-\lambda_1^2)}$$

Another method that can be used for updating is using the Green's function with these equations:

$$\hat{x}_t(l) = G_l a_t + G_{l+1} a_{t-1} + G_{l+2} a_{t-2} + \cdots$$

$$\hat{x}_{t+1}(l) = G_l a_{t+l} + \hat{x}_t(l+1)$$

Circle Y:

Order	RSS	#UAC>3	F0<>Fcrit
0,0	208407.3	105	
1,0	4408.698	8	232082.63408>3.8604
2,1	3678.01	1	32.3822>2.6231
4,3	3665.435	1	.41598<2.3903
6,5	3648.789	1	.48151<1.9576

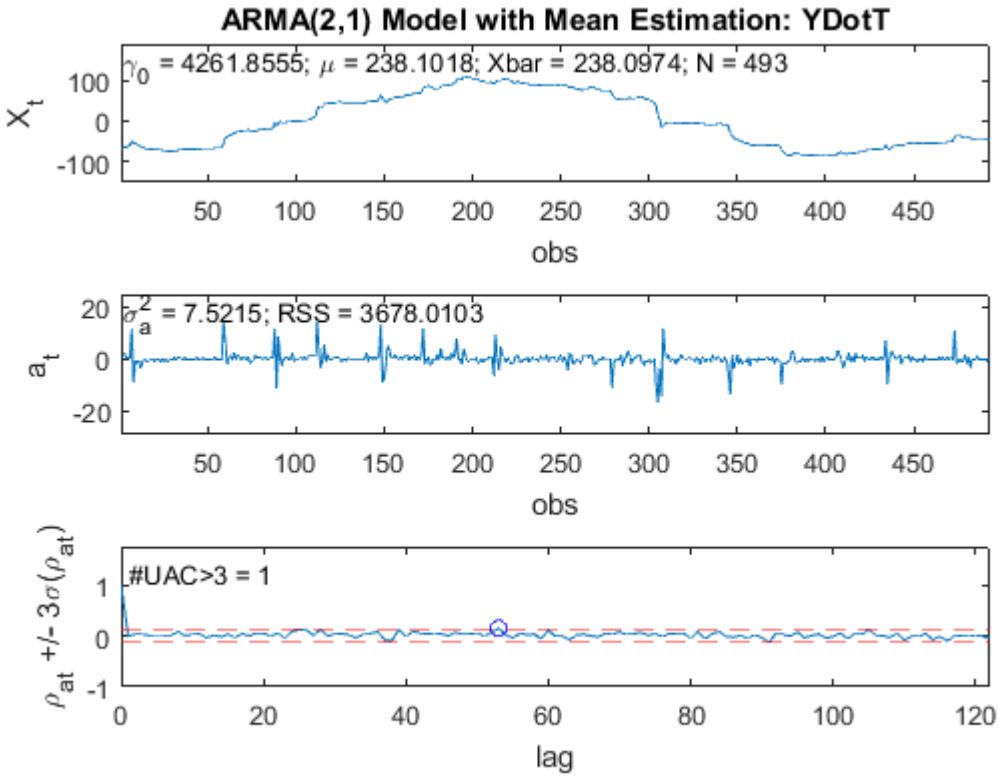
The model of interest is the ARMA(2,1) model. This was chosen due to the ARMA (4,3) model showing not much improvement in the reduction of errors proven with the F test .41598<2.3903.

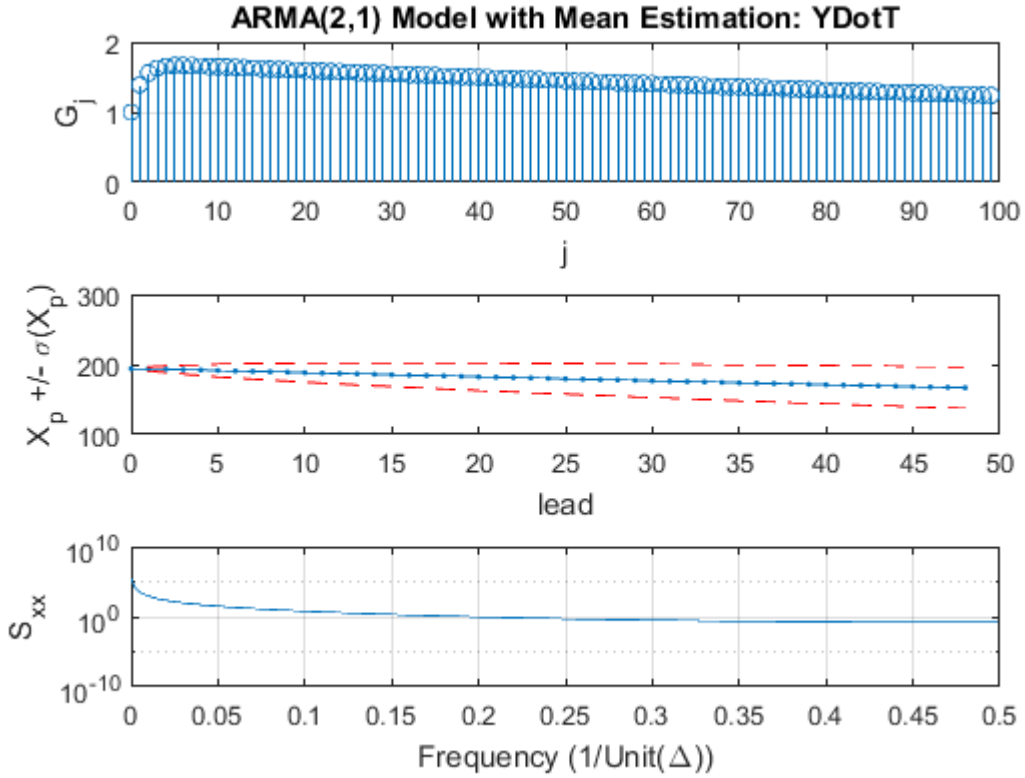
After finding the parameters of the equation the values are $\phi_1 = 1.4280$, $\phi_2 = -.4299$ and $\theta_1 = .0346$. Looking at a ARMA(2,1) model that appears in this form:

$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + a_t - \theta_1 a_{t-1}$ ARMA(2,1)

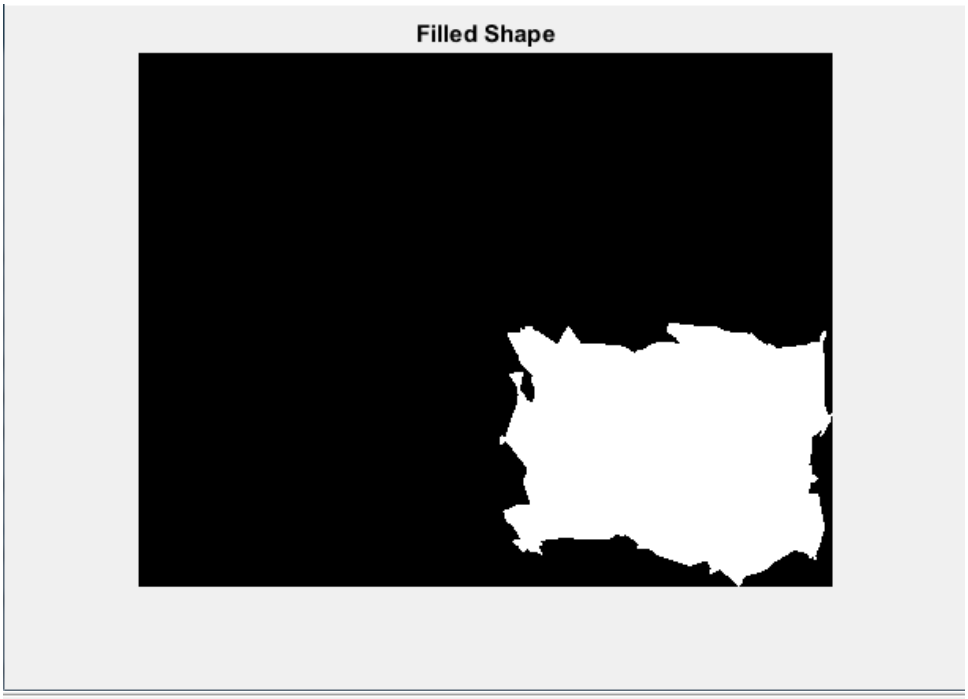
Plugging in the values from the program:

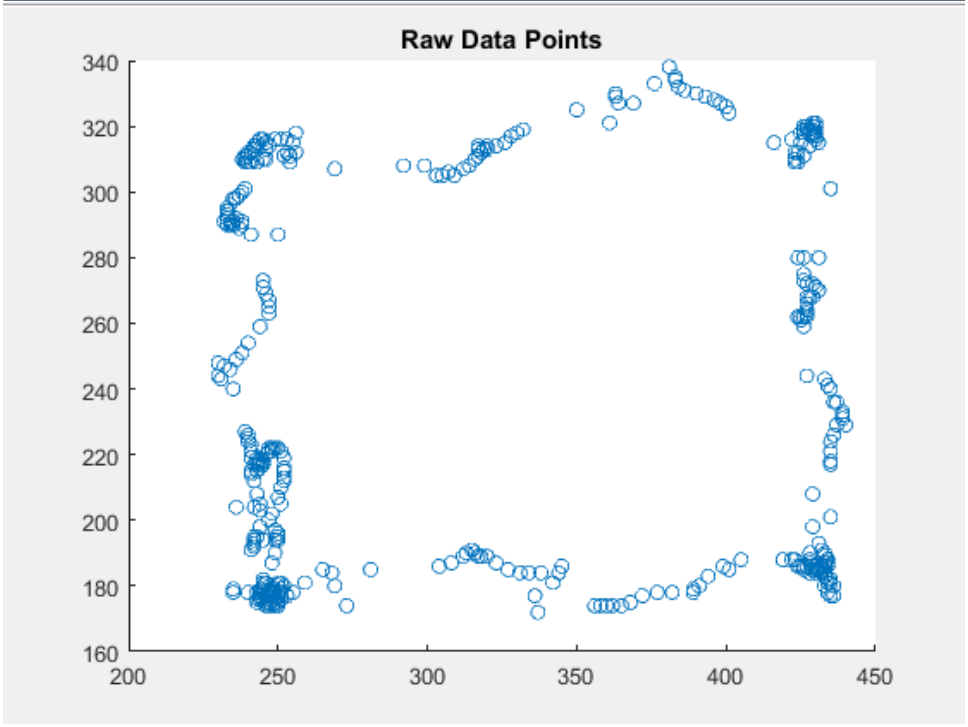
$x_t = 1.4280 x_{t-1} - .4299 x_{t-2} + a_t - .0346 a_{t-1}$ ARMA(2,1) Y

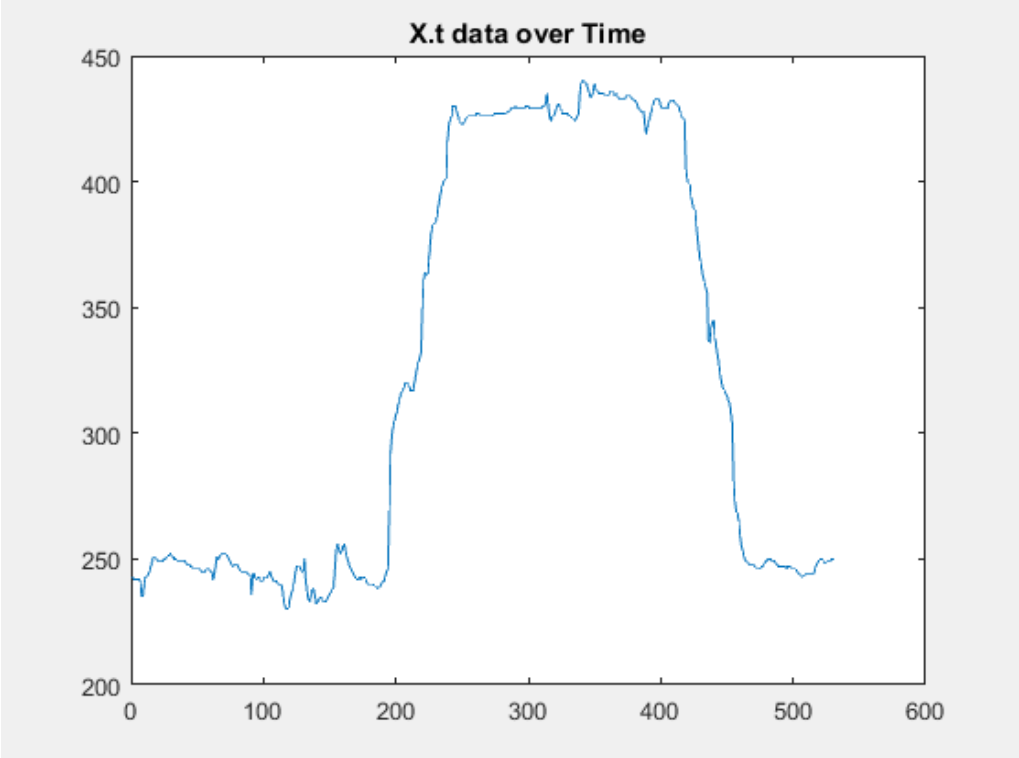

ARMA(2,1) Model with Mean Estimation: YDotT
γ_0 = 4261.8555; μ = 238.1018; Xbar = 238.0974; N = 493
X_t
100
0
-100
50
100
150
200
250
300
350
400
450
obs
σ_a^2 = 7.5215; RSS = 3678.0103
a_t
20
0
-20
50
100
150
200
250
300
350
400
450
obs
#UAC>3 = 1
ρ_{at} +/- 3σ(ρ_{at})
1
0
-1
0
20
40
60
80
100
120
lag

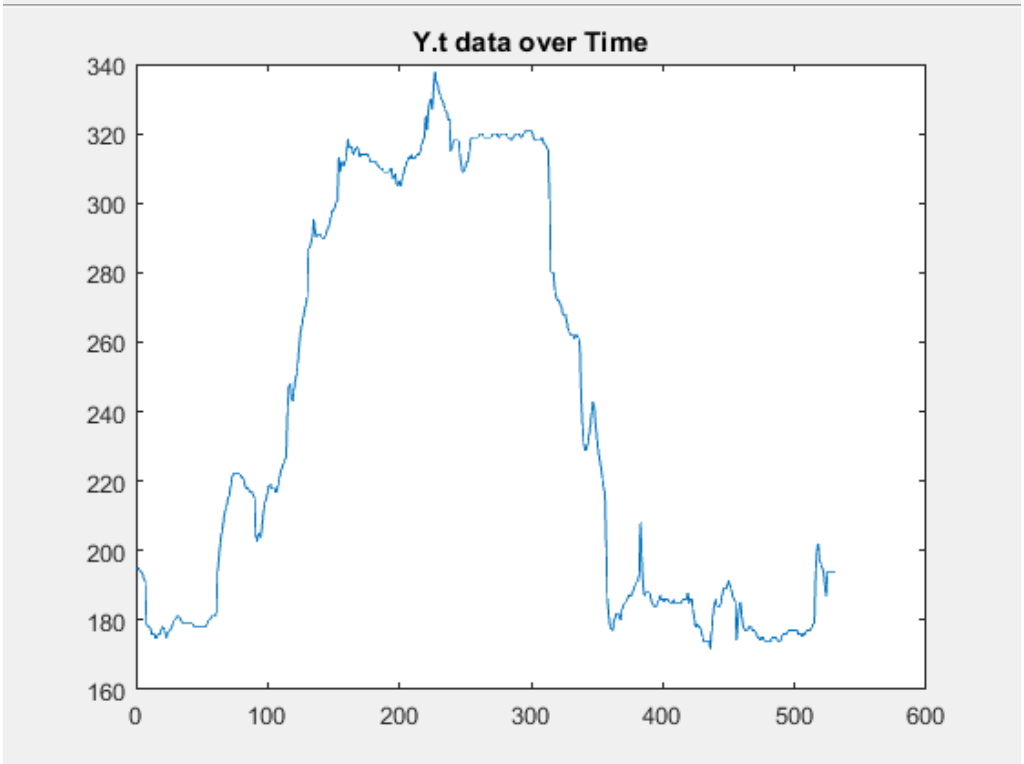

ARMA(2,1) Model with Mean Estimation: YDotT
G_j
j
X_p +/- $\sigma(X_p)$
lead
S_{xx}
Frequency (1/Unit(Δ))

Square X:


Filled Shape


Raw Data Points
340
320
300
280
260
240
220
200
180
160
200
250
300
350
400
450





Looking at the x portion:

Order	RSS	#UAC>3	F0<>Fcrit
0,0	3823161	105	
1,0	4998.585	24	404839.7739>3.8591
2,1	3812.761	5	54.6349>2.6218
4,3	3813.234	4	-0.016204<2.389
6,5	3692.074	5	2.1207>1.9562
20,19	3535.125	3	.77853<1.4993

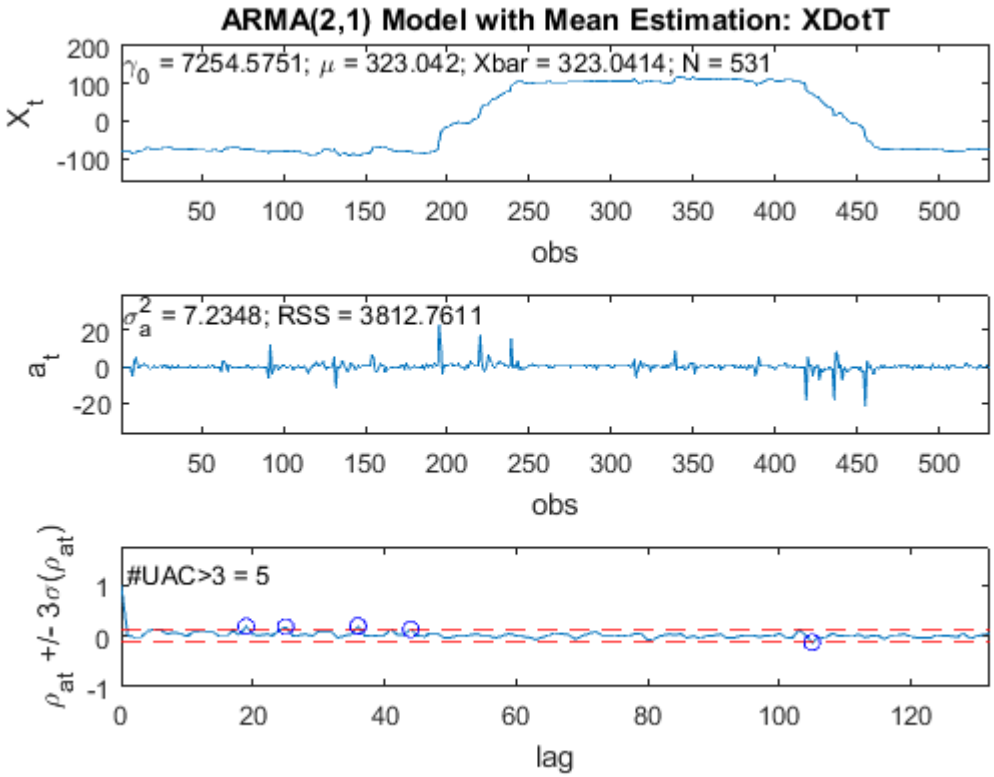
The model of interest is the ARMA(2,1) model. This was chosen due to the ARMA (4,3) model showing not much improvement in the reduction of errors proven with the F test -.016204 <2.389.

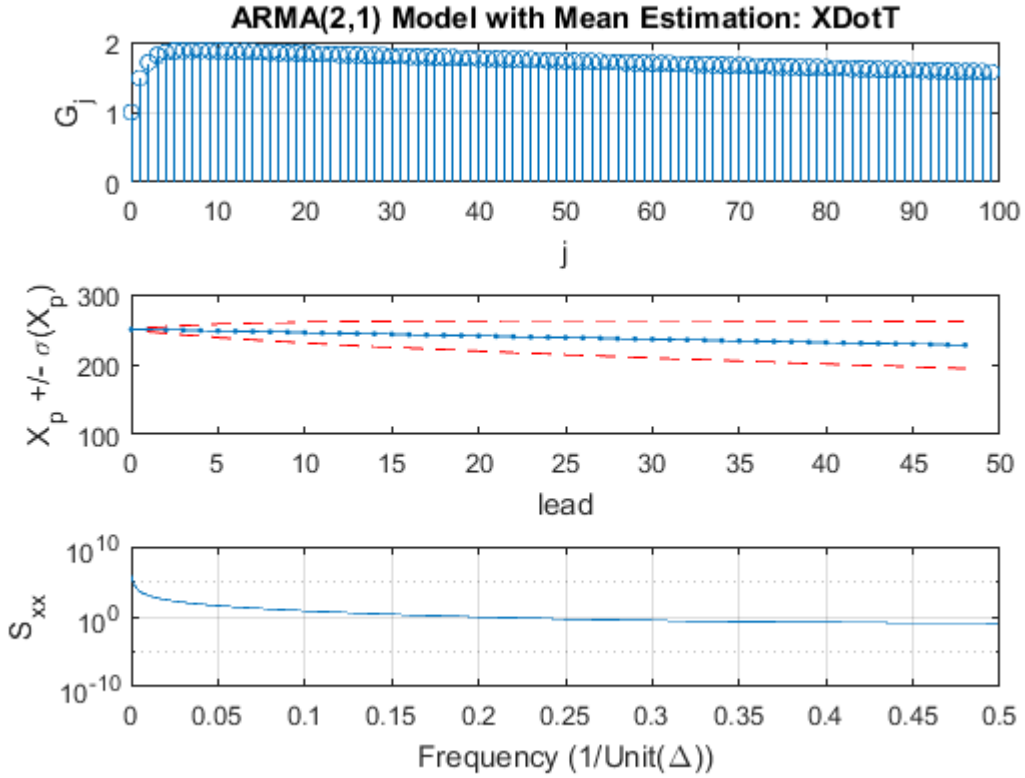
After finding the parameters of the equation the values are $\phi_1 = 1.4605$, $\phi_2 = -.4616$ and θ_1 =-.0264 Looking at a ARMA(2,1) model that appears in this form:

$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + a_t - \theta_1 a_{t-1}$ ARMA(2,1)

Plugging in the values from the program:

$x_t = 1.4605 x_{t-1} - .4616 x_{t-2} + a_t + .0264 a_{t-1}$ ARMA(2,1) X


ARMA(2,1) Model with Mean Estimation: XDotT
γ_0 = 7254.5751; μ = 323.042; Xbar = 323.0414; N = 531
X_t
obs
σ_a^2 = 7.2348; RSS = 3812.7611
a_t
obs
#UAC>3 = 5
ρ_{at} +/- $3\sigma(\rho_{at})$
lag



Square Y:

Order	RSS	#UAC>3	F0<>Fcrit
0,0	1882863	112	
1,0	4588.306	4	216961.4329>3.8591
2,1	4084.694	1	21.6583>2.6218
4,3	4062.313	1	.72038<2.389
6,5	3973.808	1	1.8103<1.9562

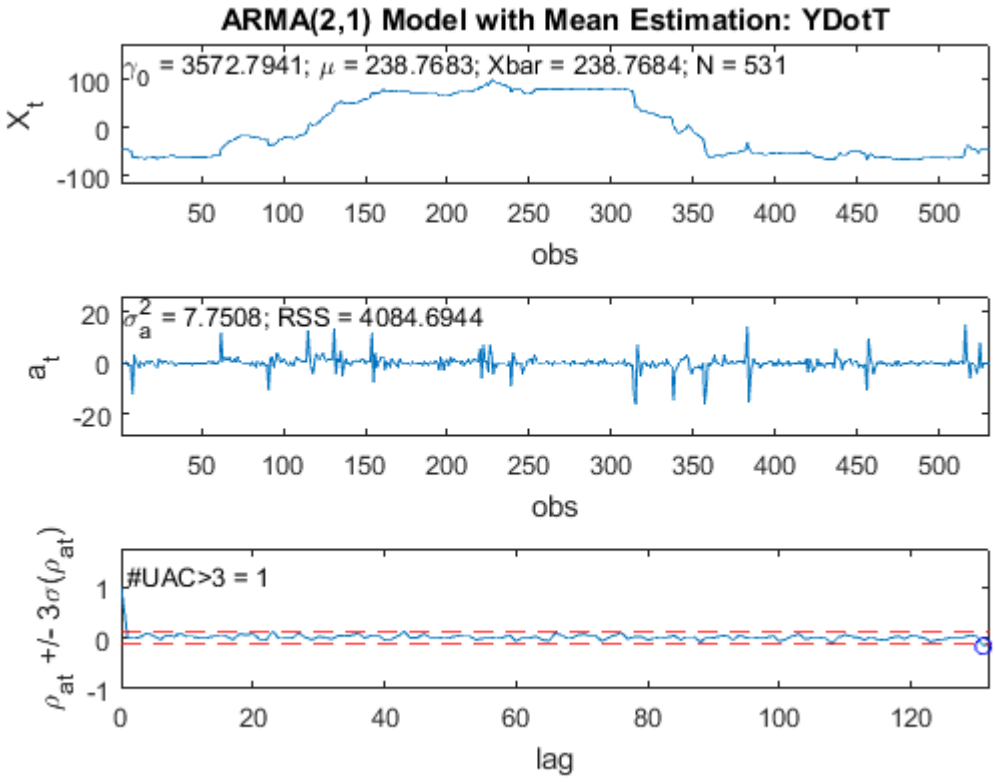
The model of interest is the ARMA(2,1) model. This was chosen due to the ARMA (4,3) model showing not much improvement in the reduction of errors proven with the F test .72038<2.389.

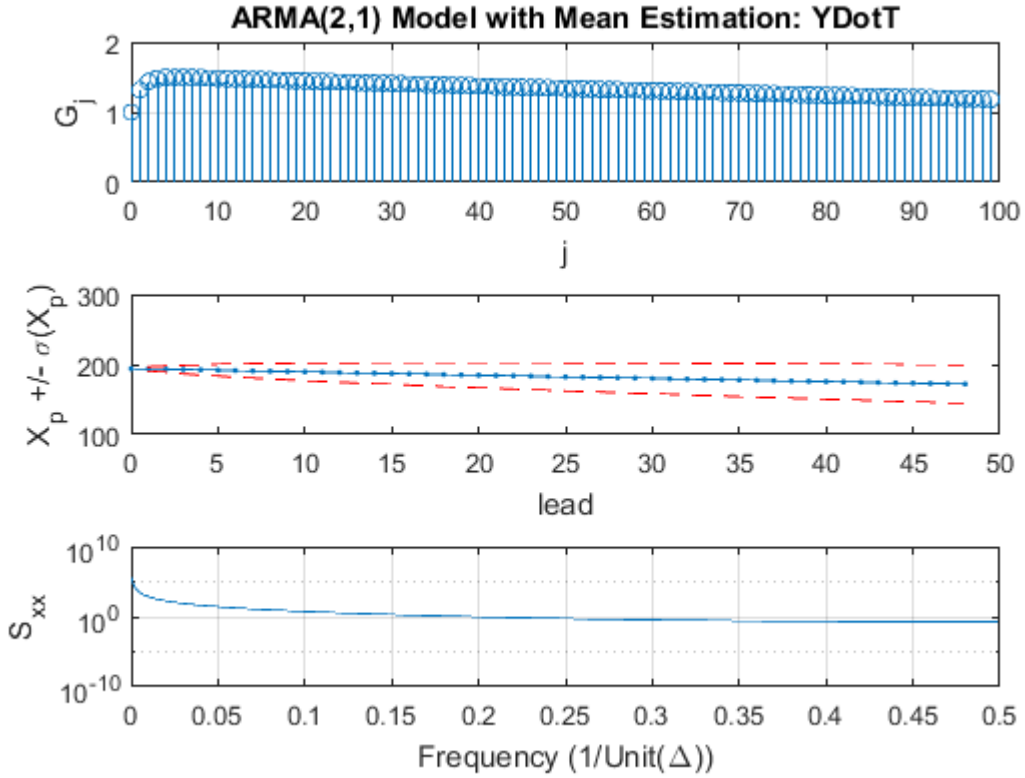
After finding the parameters of the equation the values are $\phi_1 = 1.3803$, $\phi_2 = -.3818$ and $\theta_1 = .0627$. Looking at a ARMA(2,1) model that appears in this form:

$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + a_t - \theta_1 a_{t-1}$ ARMA(2,1)

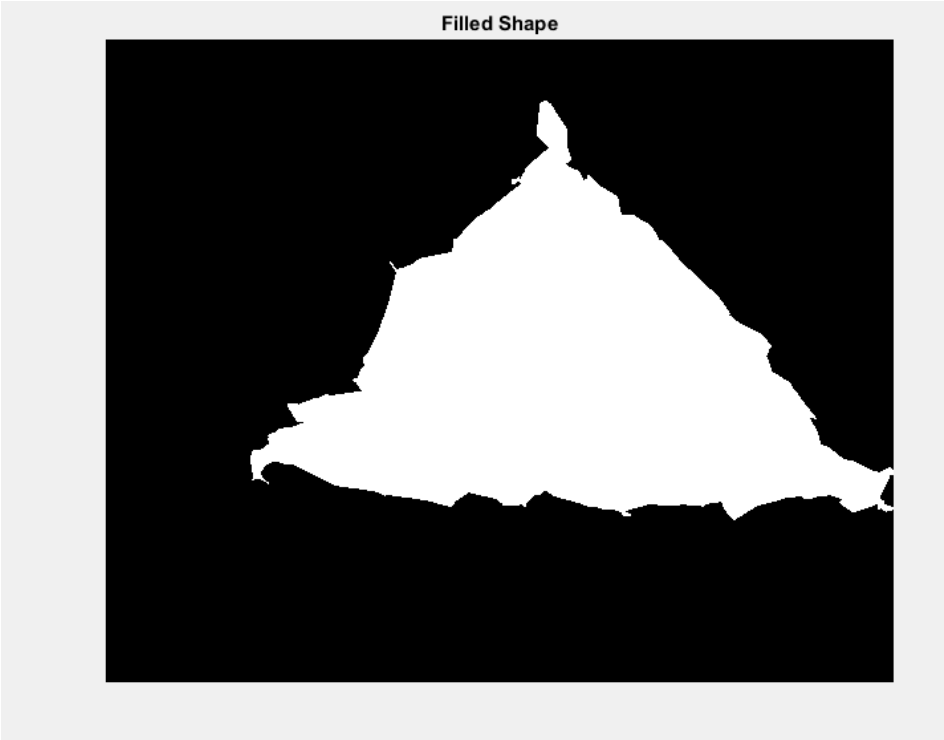
Plugging in the values from the program:

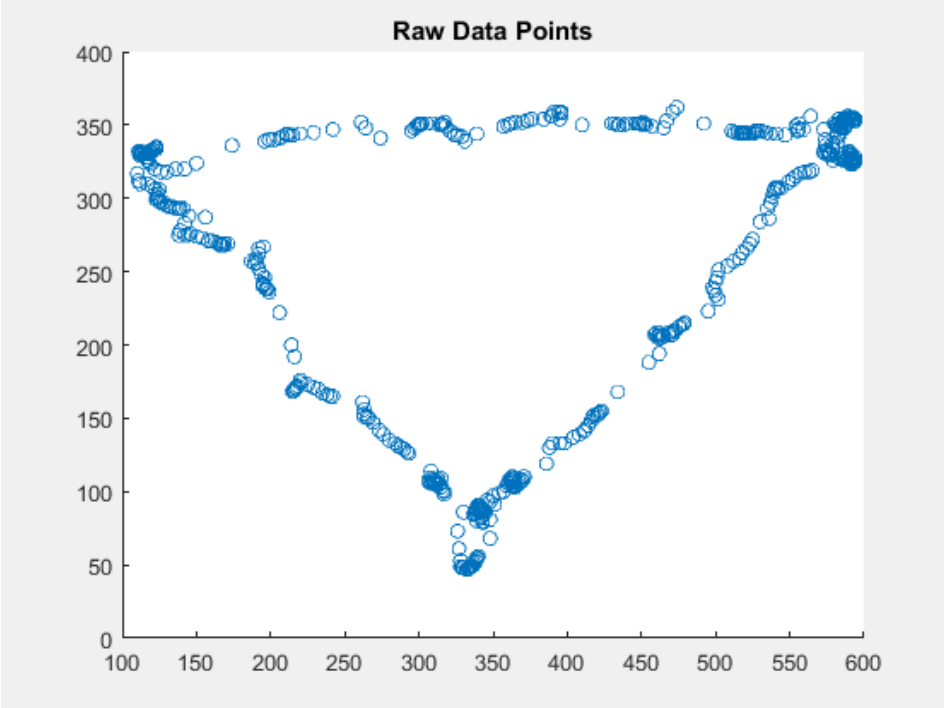
$x_t = 1.3803 x_{t-1} - .3818 x_{t-2} + a_t - .0627 a_{t-1}$ ARMA(2,1) Y

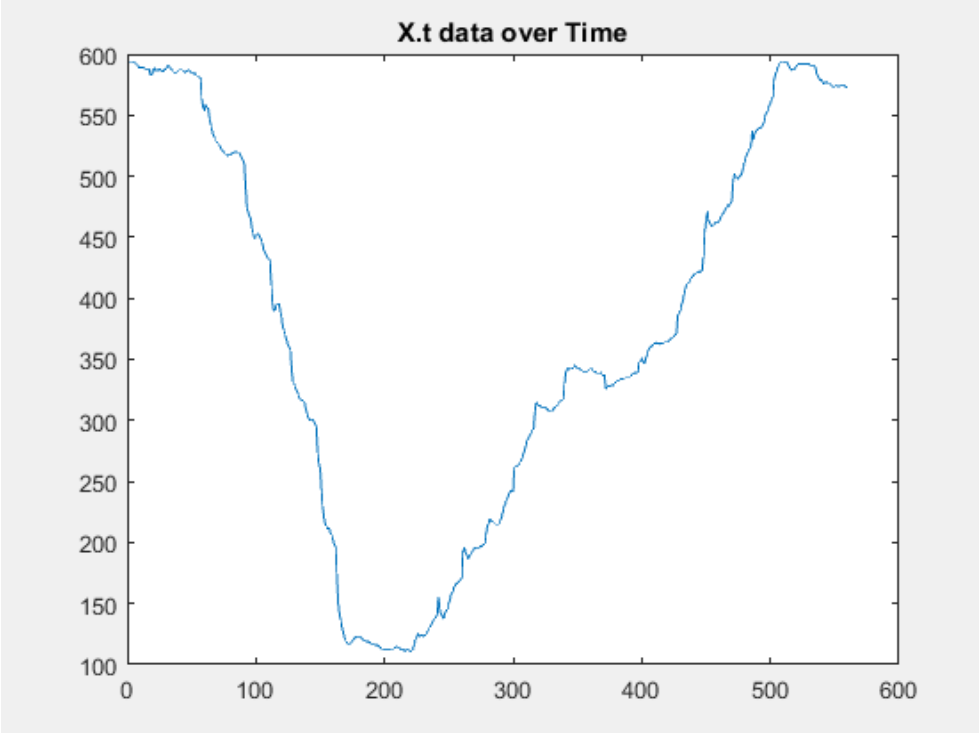


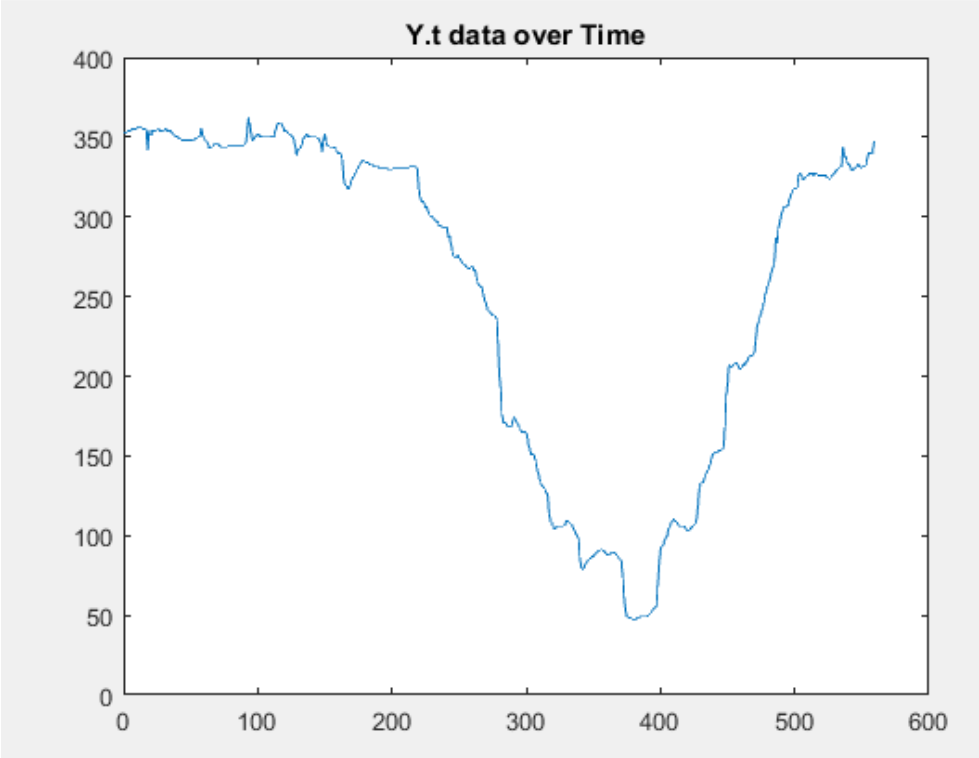

ARMA(2,1) Model with Mean Estimation: YDotT
G_j
j
X_p +/- $\sigma(X_p)$
lead
S_{xx}
Frequency (1/Unit(Δ))

Triangle X:


Filled Shape


Raw Data Points
400
350
300
250
200
150
100
50
0
100
150
200
250
300
350
400
450
500
550
600


X.t data over Time
600
550
500
450
400
350
300
250
200
150
100
0
100
200
300
400
500
600


Y.t data over Time
400
350
300
250
200
150
100
50
0
0
100
200
300
400
500
600

Order	RSS	#UAC>3	F0<>Fcrit
0,0	14977486	117	
1,0	1090.939	62	822464.498>3.8581
2,1	7648.735	6	61.7099>2.6209
4,3	7631.536	4	.31157<2.3881
6,5	7551.766	2	.8818<1.9553

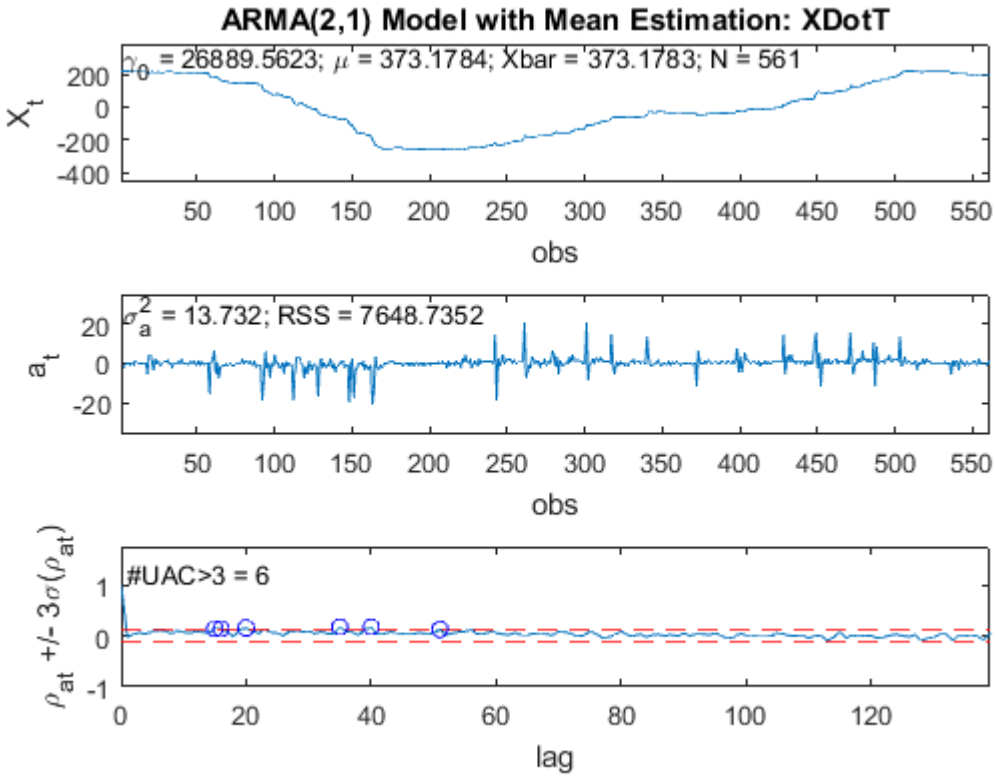
The model of interest is the ARMA(2,1) model. This was chosen due to the ARMA (4,3) model showing not much improvement in the reduction of errors proven with the F test .31157<2.3881.

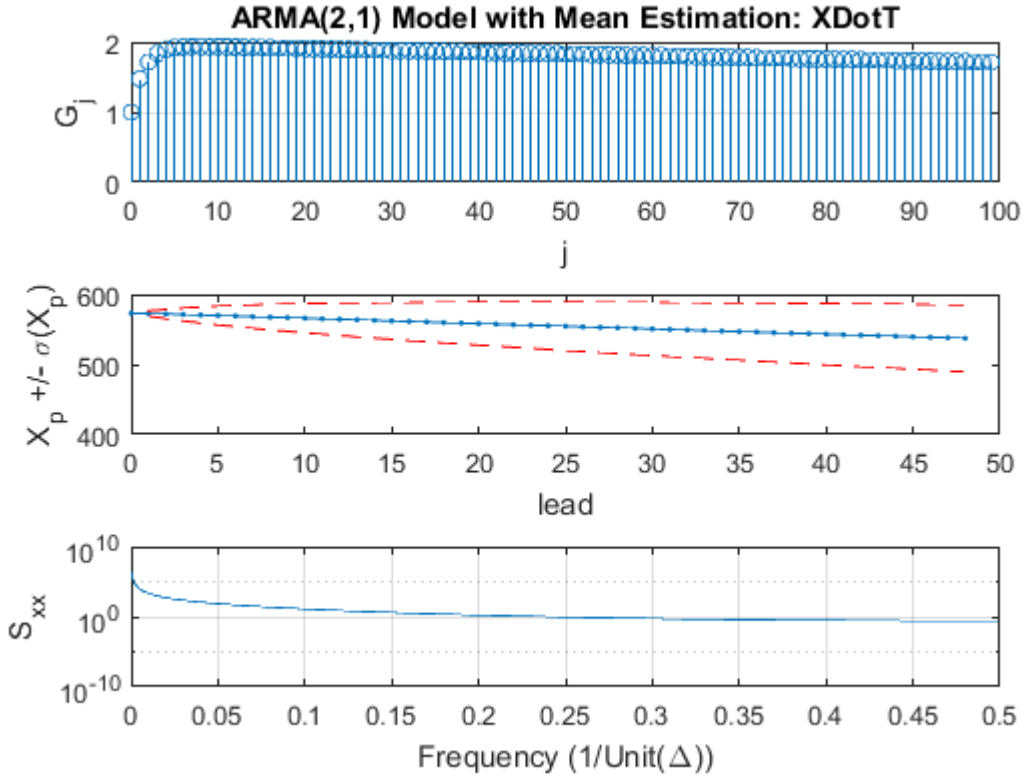
After finding the parameters of the equation the values are $\phi_1 = 1.4971$, $\phi_2 = -.4977$ and $\theta_1 = .0180$. Looking at a ARMA(2,1) model that appears in this form:

$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + a_t - \theta_1 a_{t-1}$ ARMA(2,1)

Plugging in the values from the program:

$x_t = 1.4971 x_{t-1} - .4977\ x_{t-2} + a_t - .0180 a_{t-1}$ ARMA(2,1) X


ARMA(2,1) Model with Mean Estimation: XDotT
γ_0 = 26889.5623; μ = 373.1784; Xbar = 373.1783; N = 561
X_t
obs
σ_a^2 = 13.732; RSS = 7648.7352
a_t
obs
#UAC>3 = 6
ρ_{at} +/- 3σ(ρ_{at})
lag


ARMA(2,1) Model with Mean Estimation: XDotT
G_j
j
X_p +/- $\sigma(X_p)$
lead
S_{xx}
Frequency (1/Unit(Δ))

Triangle Y:

Order	RSS	#UAC>3	F0<>Fcrit
0,0	6086454	117	
1,0	6025.892	42	565068.1576>3.8581
2,1	4519.056	2	61.9088>2.6209
4,3	4545.927	2	-0.8172<2.3881
6,5	4555.786	2	-0.55328<1.9553

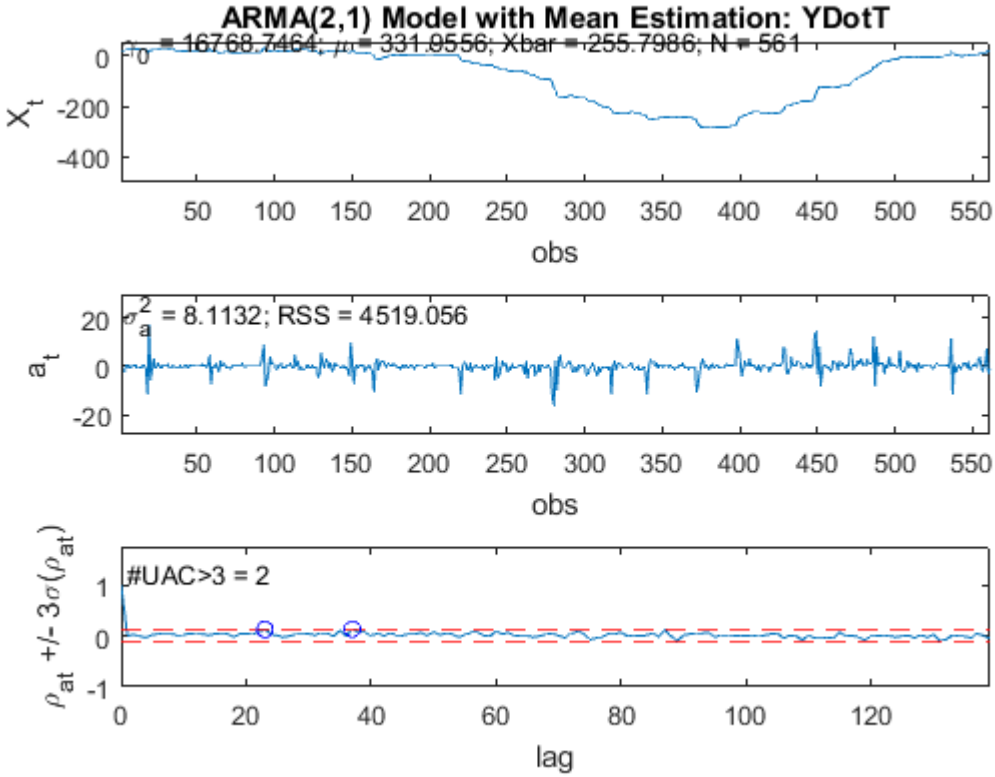
The model of interest is the ARMA(2,1) model. This was chosen due to the ARMA (4,3) model showing not much improvement in the reduction of errors proven with the F test -.8172 <2.3881.

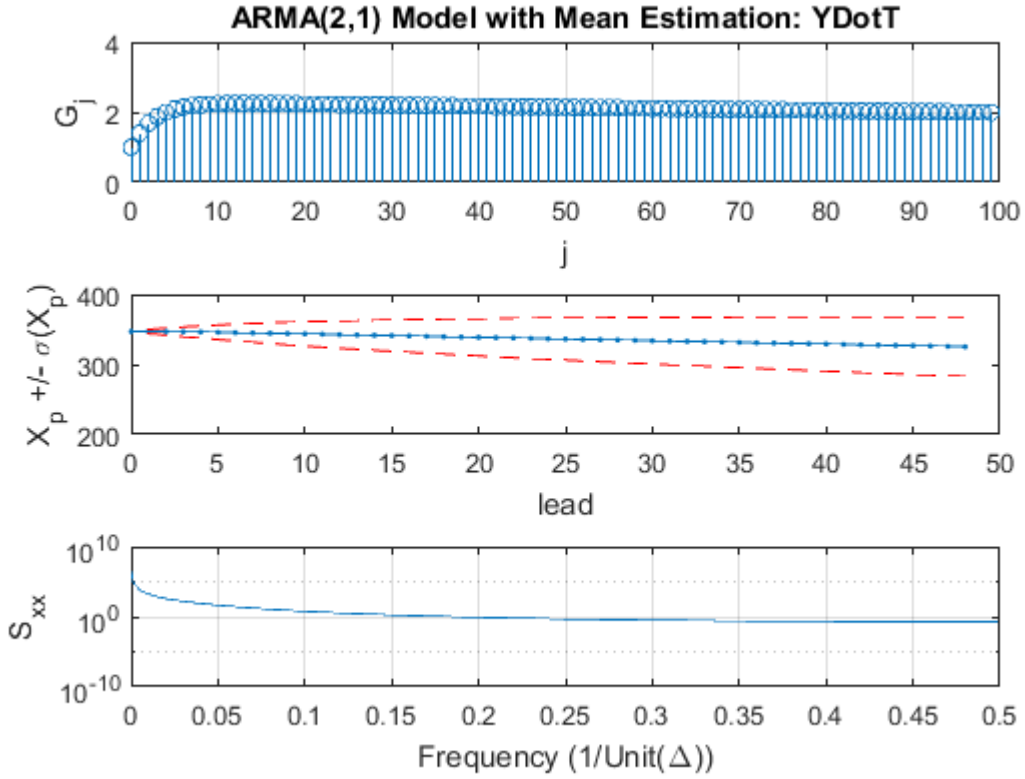
After finding the parameters of the equation the values are $\phi_1 = 1.6836$, $\phi_2 = -.6840$ and $\theta_1 = .2779$. Looking at a ARMA(2,1) model that appears in this form:

$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + a_t - \theta_1 a_{t-1}$ ARMA(2,1)

Plugging in the values from the program:

$x_t = 1.6836 x_{t-1} - .6840\, x_{t-2} + a_t - .2779 a_{t-1}$ ARMA(2,1) Y


ARMA(2,1) Model with Mean Estimation: YDotT
σ0 = 16768.7464; μ = 331.9556; Xbar = 255.7986; N = 561
Xt
obs
σ2a = 8.1132; RSS = 4519.056
at
obs
#UAC>3 = 2
ρat +/- 3σ(ρat)
lag


ARMA(2,1) Model with Mean Estimation: YDotT
G_j
j
X_p +/- $\sigma(X_p)$
lead
S_{xx}
Frequency (1/Unit(Δ))

Summary:

Circle:

$$x_t = 1.4518x_{t-1} - .4528x_{t-2} + a_t - 3.2289 * 10^{-5}a_{t-1}$$ X

$$x_t = 1.4280x_{t-1} - .4299x_{t-2} + a_t - .0346a_{t-1}$$ Y

Square:

$$x_t = 1.4605x_{t-1} - .4616x_{t-2} + a_t + .0264a_{t-1}$$ X

$$x_t = 1.3803x_{t-1} - .3818x_{t-2} + a_t - .0627a_{t-1}$$ Y

Triangle:

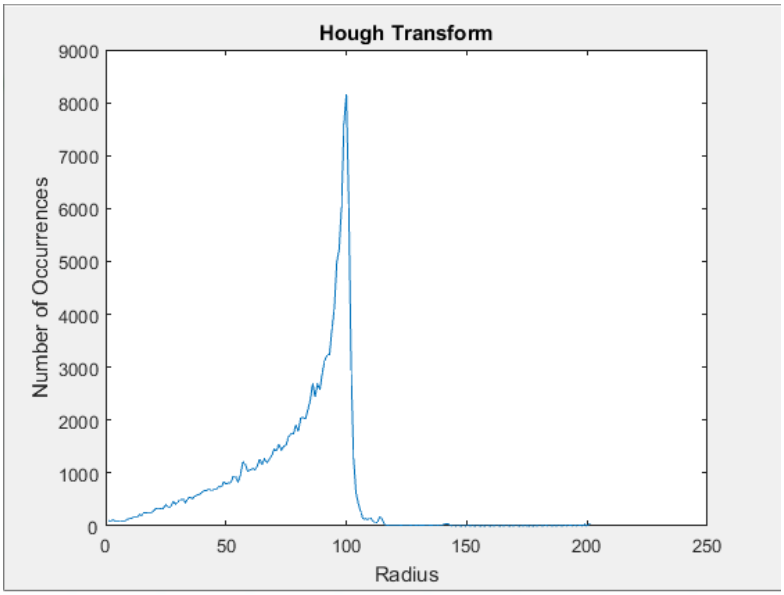
$$x_t = 1.4971x_{t-1} - .4977\ x_{t-2} + a_t - .0180a_{t-1}$$ X

$$x_t = 1.6836x_{t-1} - .6840\ x_{t-2} + a_t - .2779a_{t-1}$$ Y

HOUGH TRANSFORMS:

Now to look at the information from the Hough Transforms. For this we will be looking at the perfect transforms to try and make a model from them.

Circle:



Order	RSS	#UAC>3	F0<>Fcrit
0,0	337927303.6	19	
1,0	2266248.977	1	20782.2621>3.8884
2,1	10855181.76	1	71.4264>2.6504
4,3	9552763.11	0	6.5784>2.4184
6,5	9702242.796	0	-0.072797<2.4194

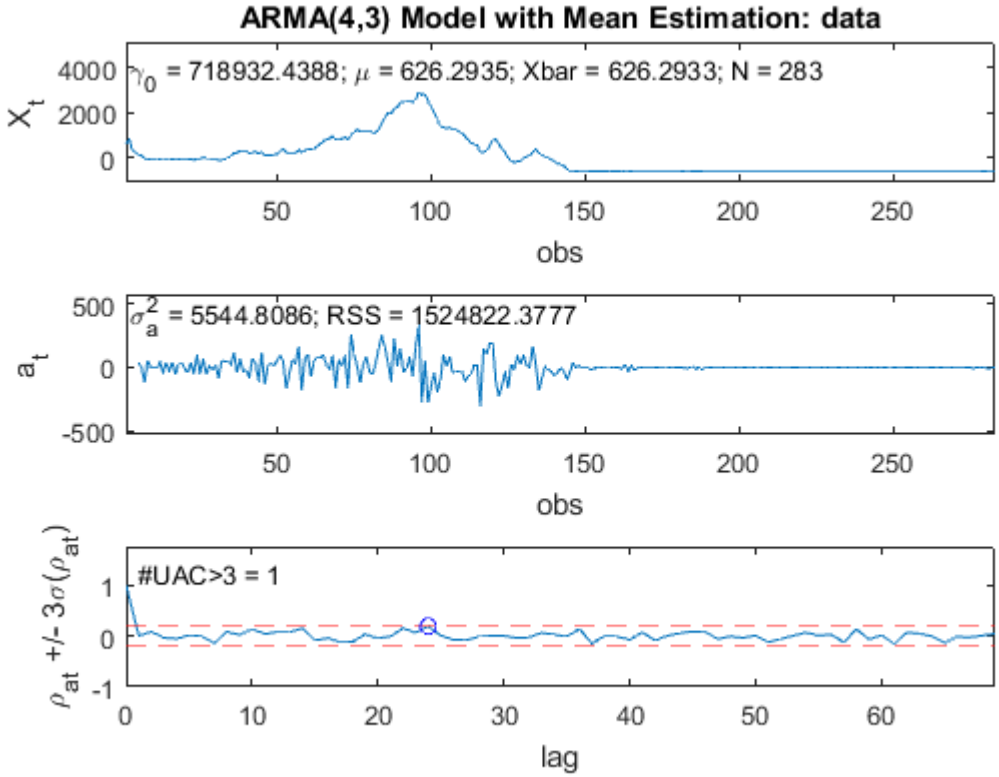
The model of interest is the ARMA(4,3) model. This was chosen due to the ARMA (6,5) model showing not much improvement in the reduction of errors proven with the F test -.072797<2.4194.

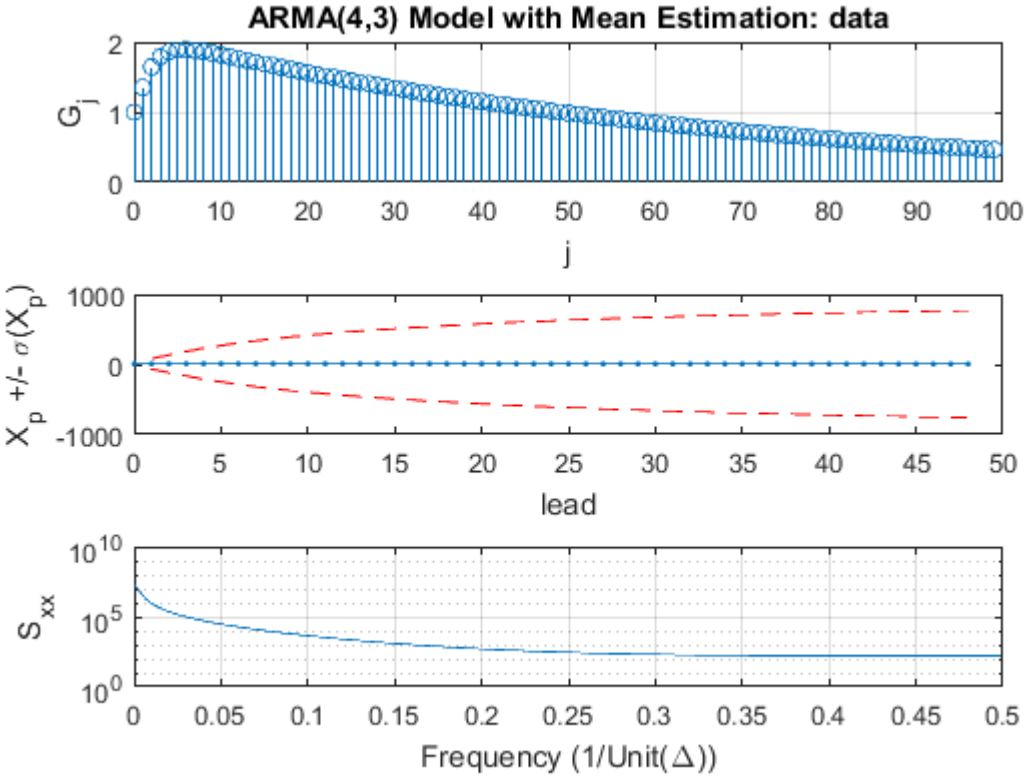
$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + \phi_3 x_{t-3} + \phi_4 x_{t-4} + a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \theta_3 a_{t-3}$$

ARMA(4,3)

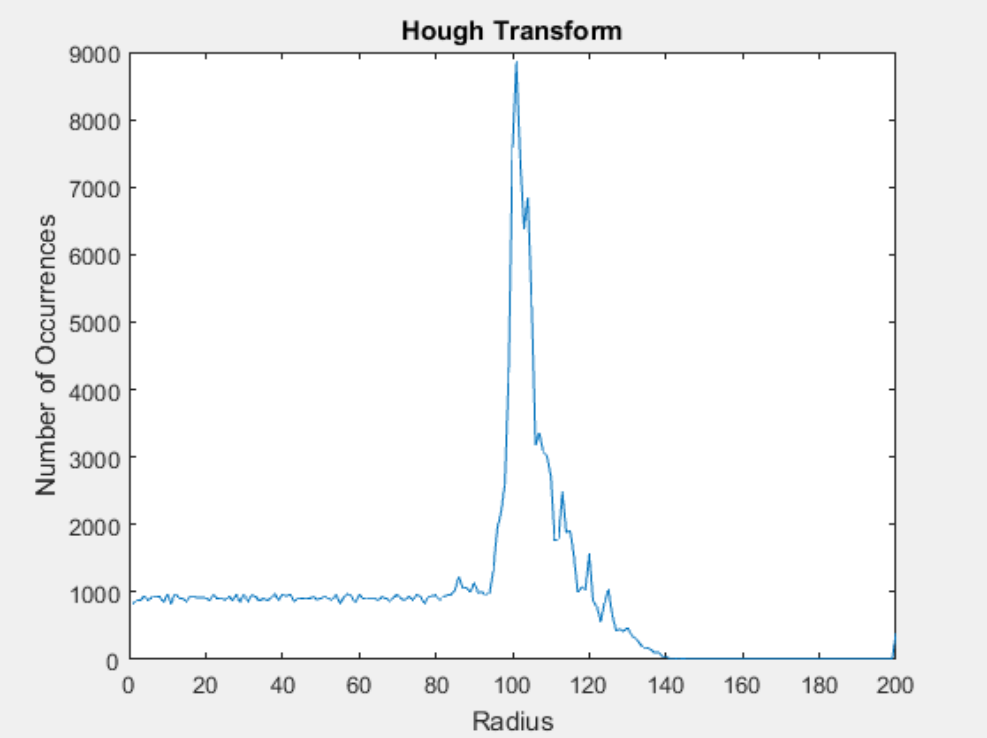
Plugging in the values from the program: $\phi_1 = .9378$ $\phi_2 = -.0512$ $\phi_3 = .0208$ $\phi_4 = -.0055$ $\theta_1 = -1.0223$ $\theta_2 = -.6001$ $\theta_3 = 2.6224 * 10^{-6}$

$$x_t = .9378\ x_{t-1} - .0512x_{t-2} + .0208\, x_{t-3} - .0055x_{t-4} + a_t + 1.0223a_{t-1} + .6001a_{t-2} - 2.6224 * 10^{-6}a_{t-3}$$


ARMA(4,3) Model with Mean Estimation: data
γ_0 = 718932.4388; μ = 626.2935; Xbar = 626.2933; N = 283
X_t
4000
2000
0
50
100
150
200
250
obs
σ_a^2 = 5544.8086; RSS = 1524822.3777
a_t
500
0
-500
50
100
150
200
250
obs
#UAC>3 = 1
ρ_{at} +/- 3σ(ρ_{at})
1
0
-1
0
10
20
30
40
50
60
lag


ARMA(4,3) Model with Mean Estimation: data
G_j
j
X_p +/- $\sigma(X_p)$
lead
S_{xx}
Frequency (1/Unit(Δ))

Square:



Order	RSS	#UAC>3	F0<>Fcrit
0,0	336688422.3	15	
1,0	29835388.44	2	2046.6887>3.8886
2,1	22677807.01	0	20.6205>2.6507
4,3	21786042.43	0	1.9648<2.4187
6,5	21951324.39	0	.77774<1.9879

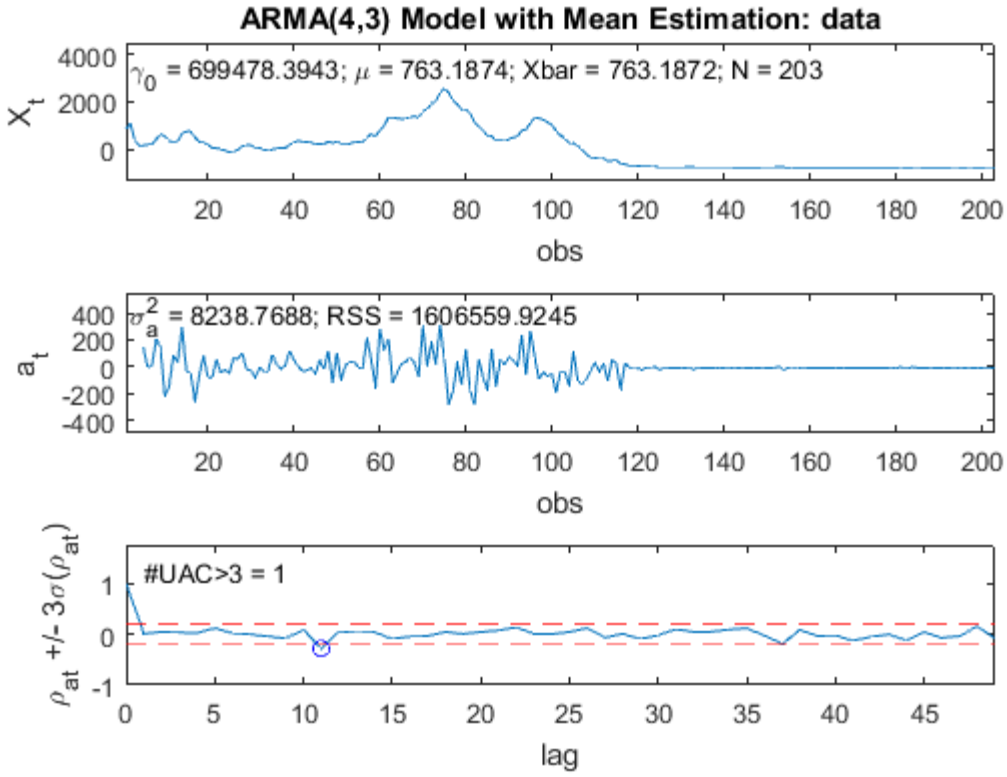
The model of interest is the ARMA(4,3) model. This was chosen due to the previous choosing of ARMA(4,3), although a ARMA(2,1) may have fit this model fine as shown with the F-test.

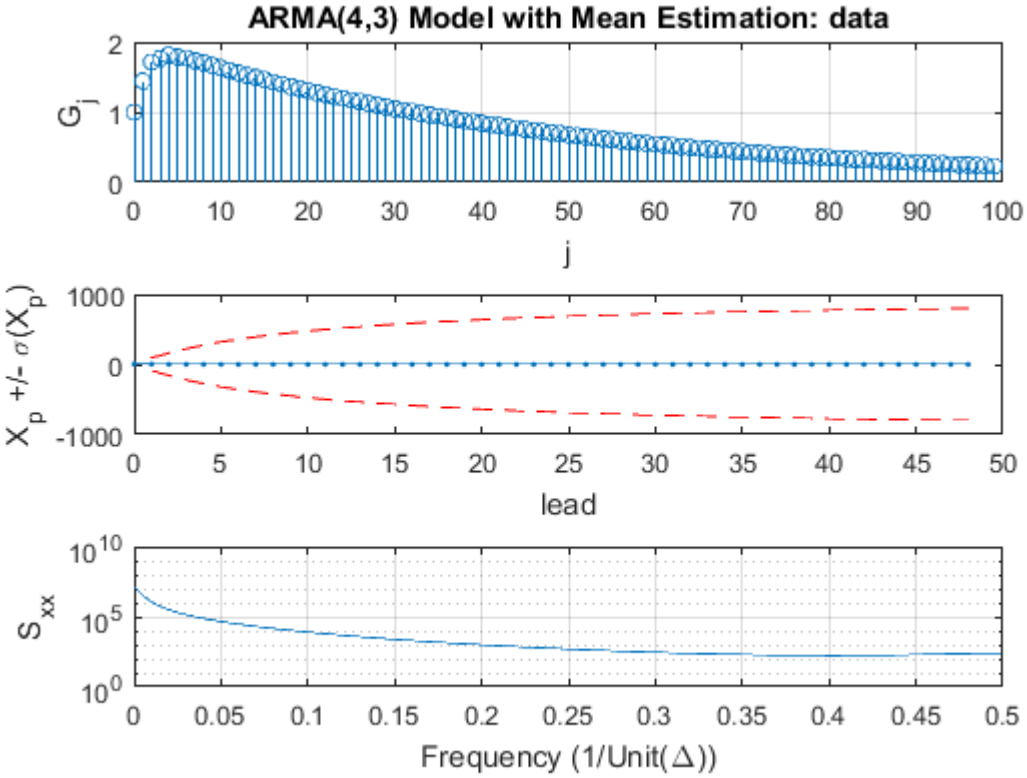
$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + \phi_3 x_{t-3} + \phi_4 x_{t-4} + a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \theta_3 a_{t-3}$$

ARMA(4,3)

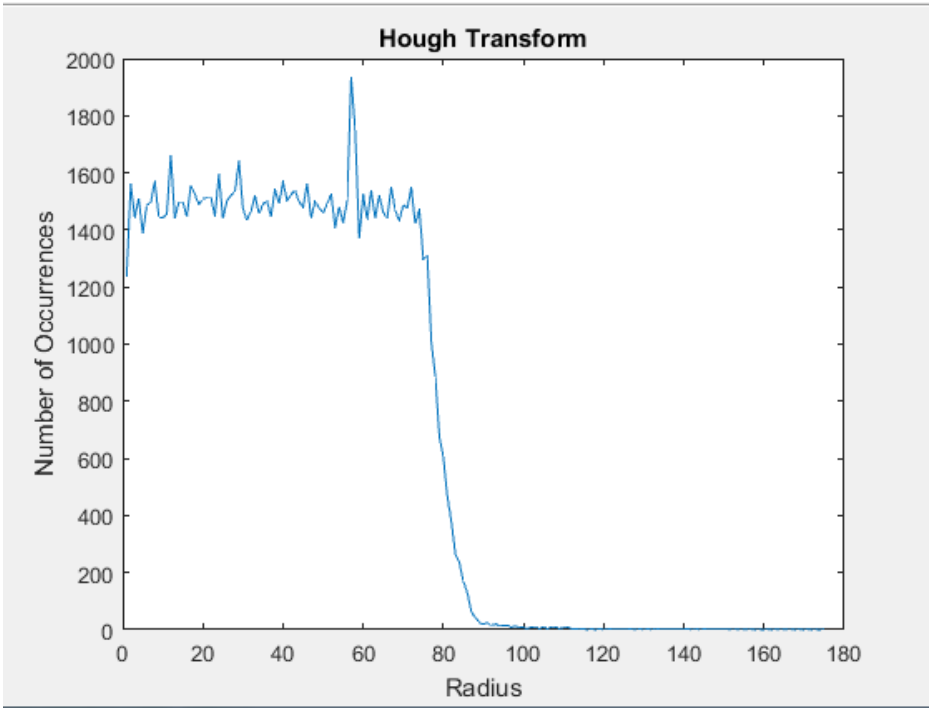
Plugging in the values from the program: $\phi_1 = 1.4977 \phi_2 = -.8875\ \phi_3 = .6115$ $\phi_4 = -.2867\ \theta_1 = .0063\ \theta_2 = -1.9988 * 10^{-5}\ \theta_3 = 8.6367 * 10^{-9}$

$$x_t = 1.4977\ x_{t-1} - .8875 x_{t-2} + .6115 x_{t-3} - .2867\ x_{t-4} + a_t - .0063\ a_{t-1} + 1.9988 * 10^{-5} a_{t-2} - 8.6367 * 10^{-9} a_{t-3}$$




ARMA(4,3) Model with Mean Estimation: data
G_j
j
X_p +/- $\sigma(X_p)$
lead
S_{xx}
Frequency (1/Unit(Δ))

Triangle:



Order	RSS	#UAC>3	F0<>Fcrit
0,0	92575038	43	
1,0	1180447	0	13471.7276>3.8955
2,1	1020838	0	8.912>2.6574
4,3	1013129	0	.31767<2.4258
6,5	979087.7	0	.86884<1.9956

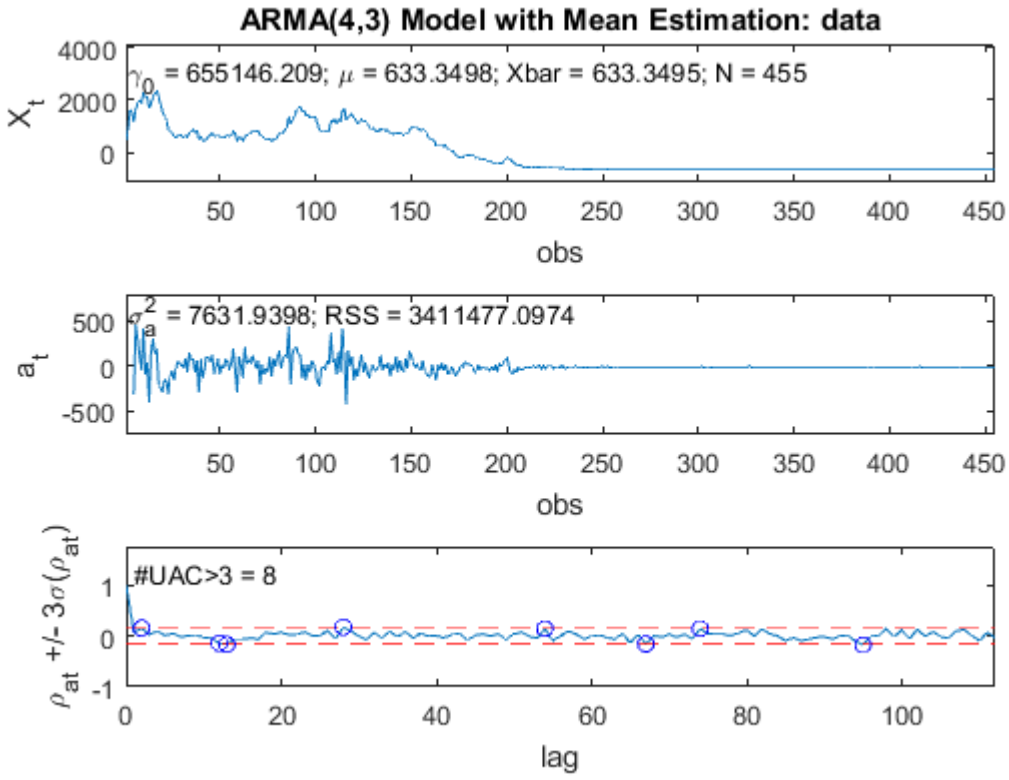
The model of interest is the ARMA(4,3) model. This was chosen due to the previous choosing of ARMA(4,3), although a ARMA(2,1) may have fit this model fine as shown with the F-test.

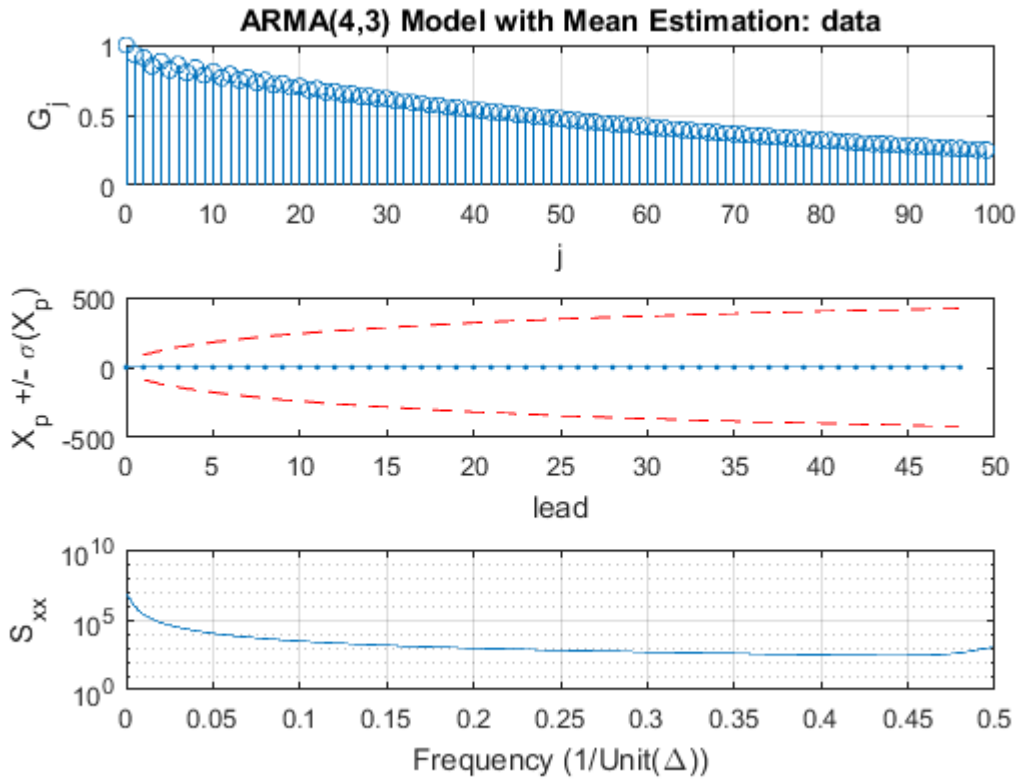
$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + \phi_3 x_{t-3} + \phi_4 x_{t-4} + a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \theta_3 a_{t-3}$$

ARMA(4,3)

Plugging in the values from the program: $\phi_1 = .2432 \phi_2 = .6255\ \phi_3 = .1589 \phi_4 = -.0380\ \theta_1 = -.5676\ \theta_2 = -.0815\ \theta_3 = -2.6194 * 10^{-4}$

$$x_t = .2432 x_{t-1} + .6255\ x_{t-2} + .1589 x_{t-3} - .0380\ x_{t-4} + a_t + .5676\ a_{t-1} + .0815\ a_{t-2} + 2.6194 * 10^{-4} a_{t-3}$$





Summarized Results:

Circle

$$x_t = .9378\ x_{t-1} - .0512x_{t-2} + .0208\,x_{t-3} - .0055x_{t-4} + a_t + 1.0223a_{t-1} + .6001a_{t-2} - 2.6224 * 10^{-6}a_{t-3}$$

Square

$$x_t = 1.4977\ x_{t-1} - .8875x_{t-2} + .6115x_{t-3} - .2867\,x_{t-4} + a_t - .0063\,a_{t-1} + 1.9988 * 10^{-5}a_{t-2} - 8.6367 * 10^{-9}a_{t-3}$$

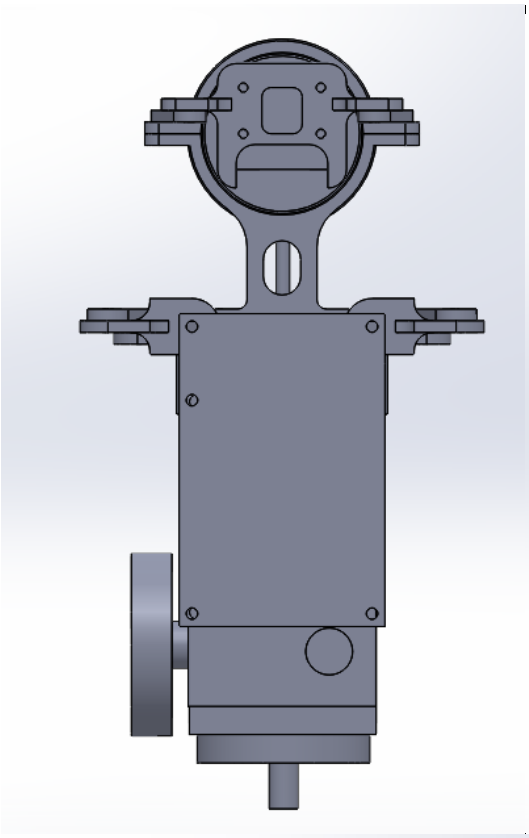
Triangle

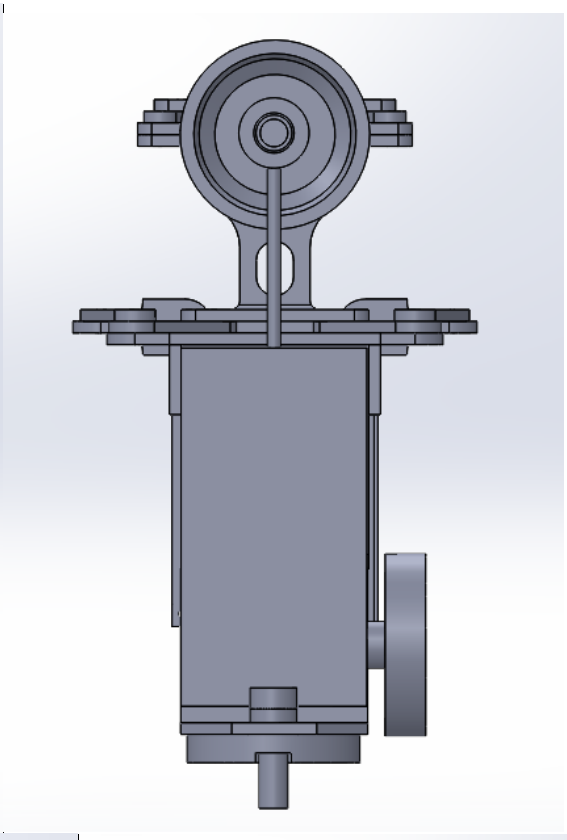
$$x_t = .2432x_{t-1} + .6255\ x_{t-2} + .1589x_{t-3} - .0380\ x_{t-4} + a_t + .5676\ a_{t-1} + .0815\ a_{t-2} + 2.6194 * 10^{-4}a_{t-3}$$

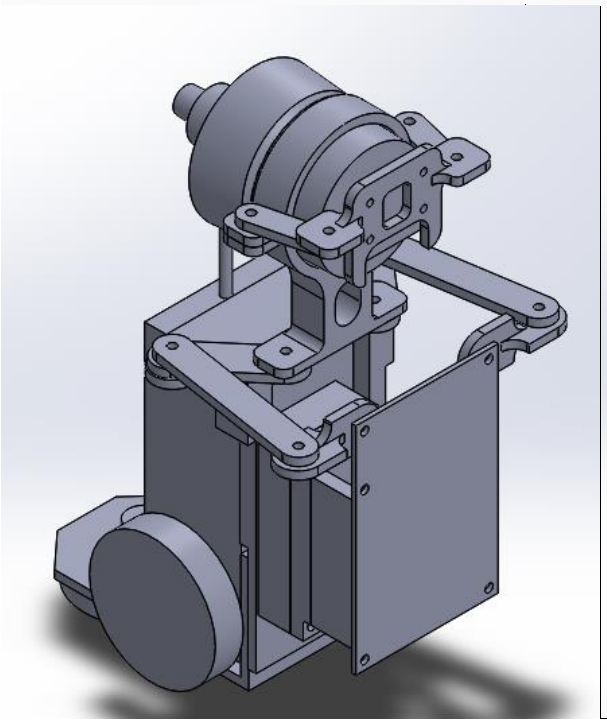
Appendix 2:

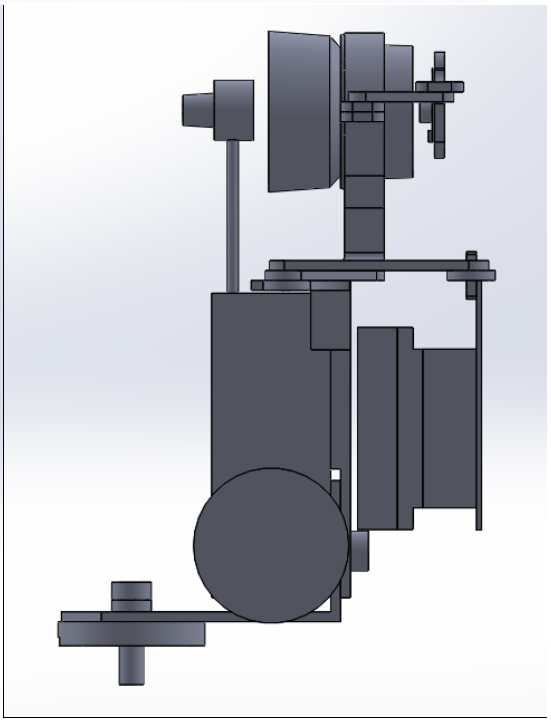
This appendix shows extra photos of the cad assembly.

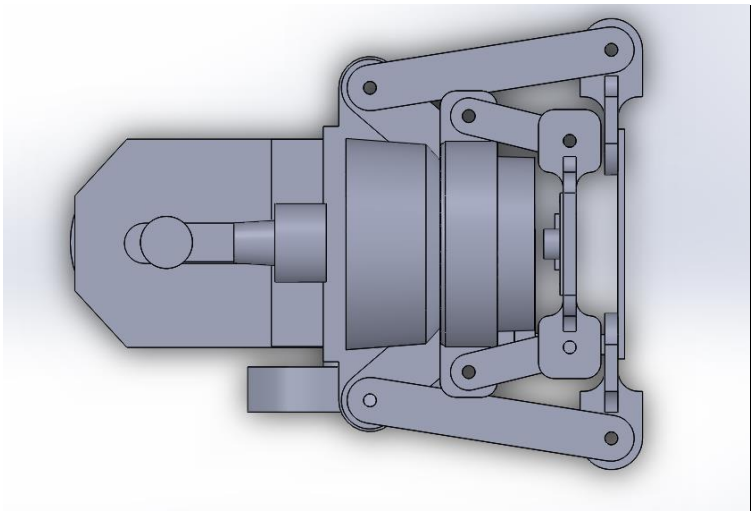
Iteration 1:



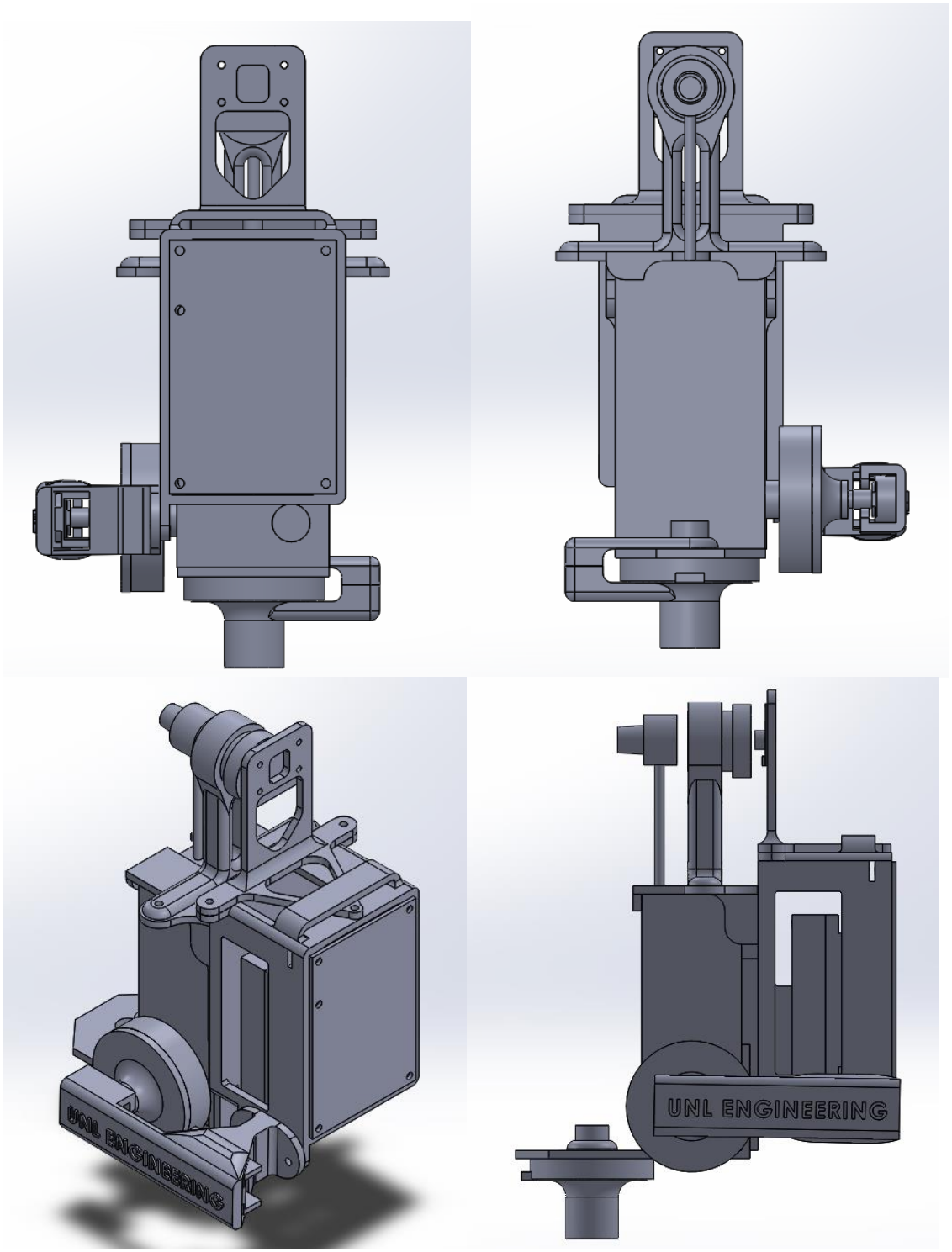


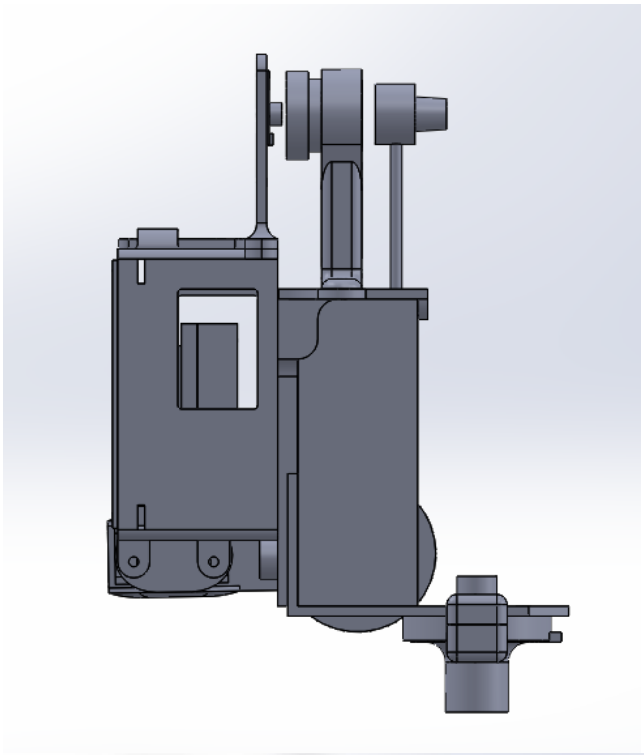


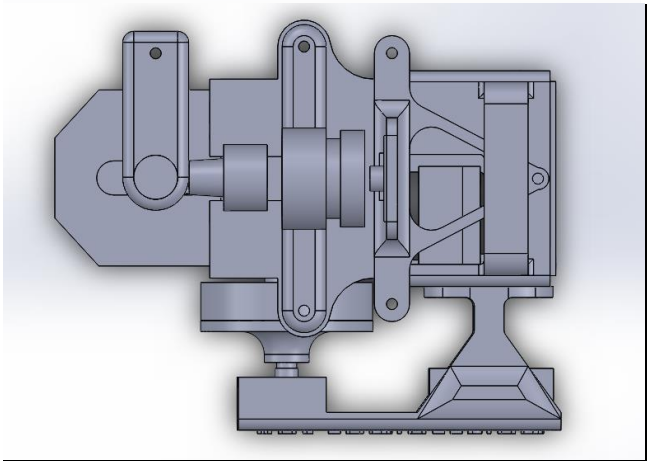




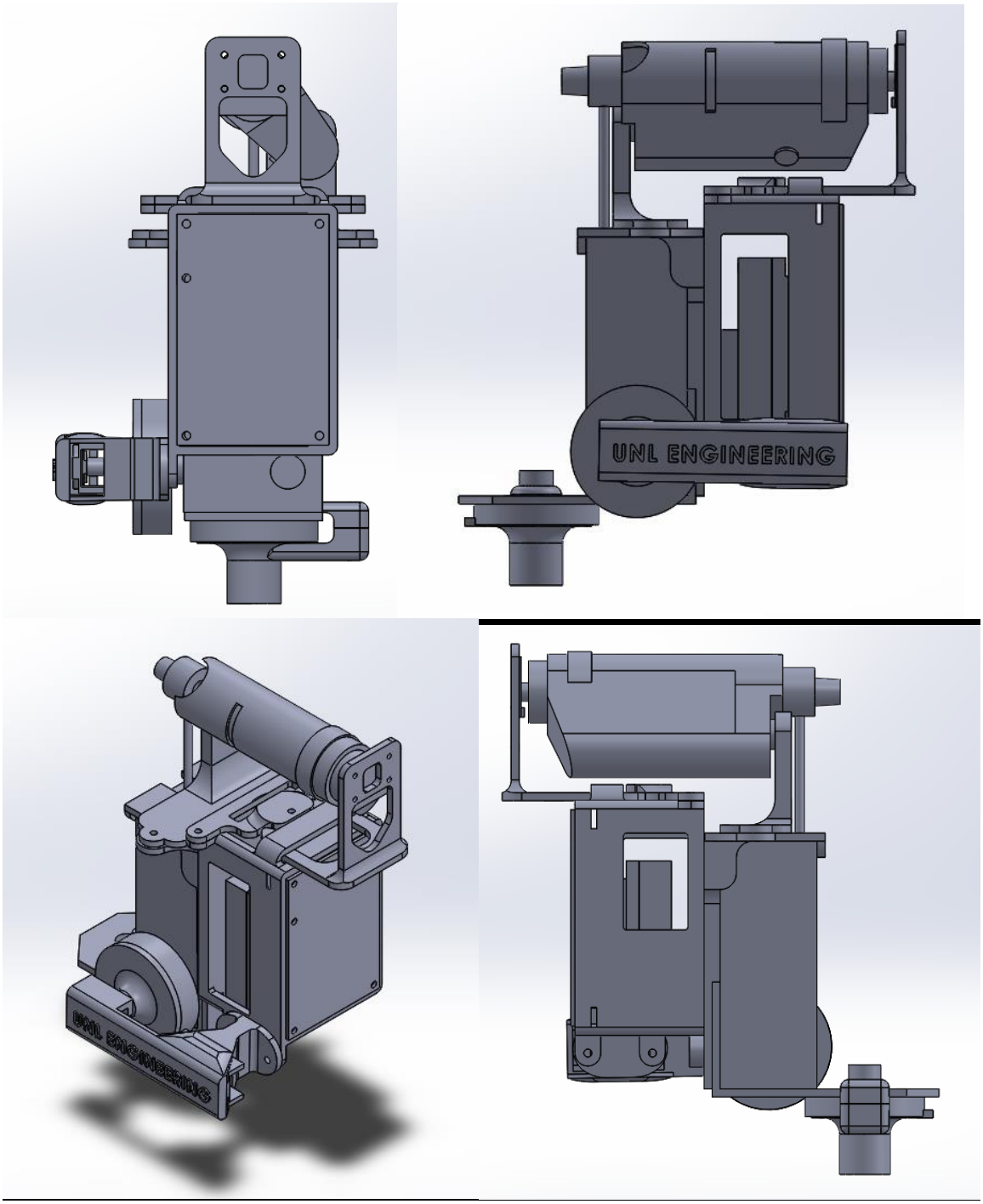
Iteration 2:

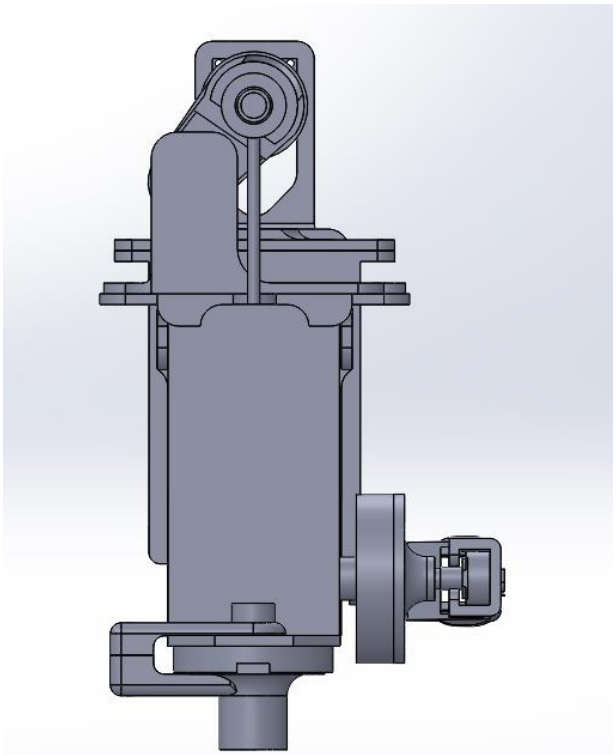


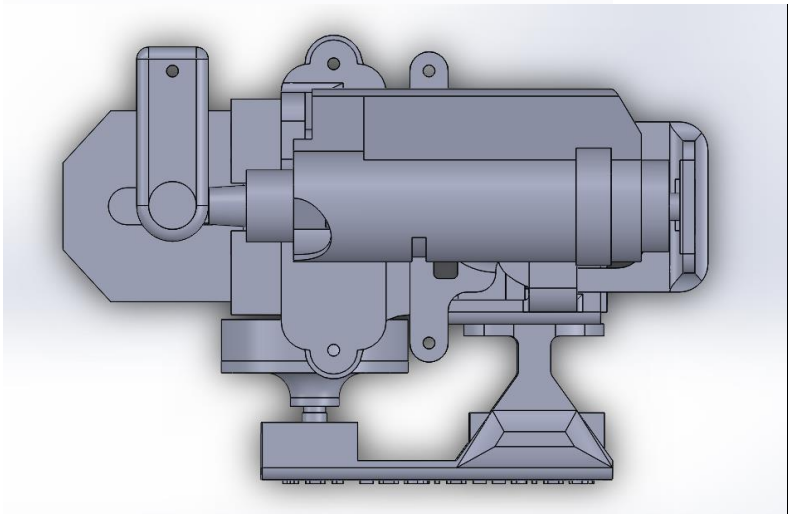




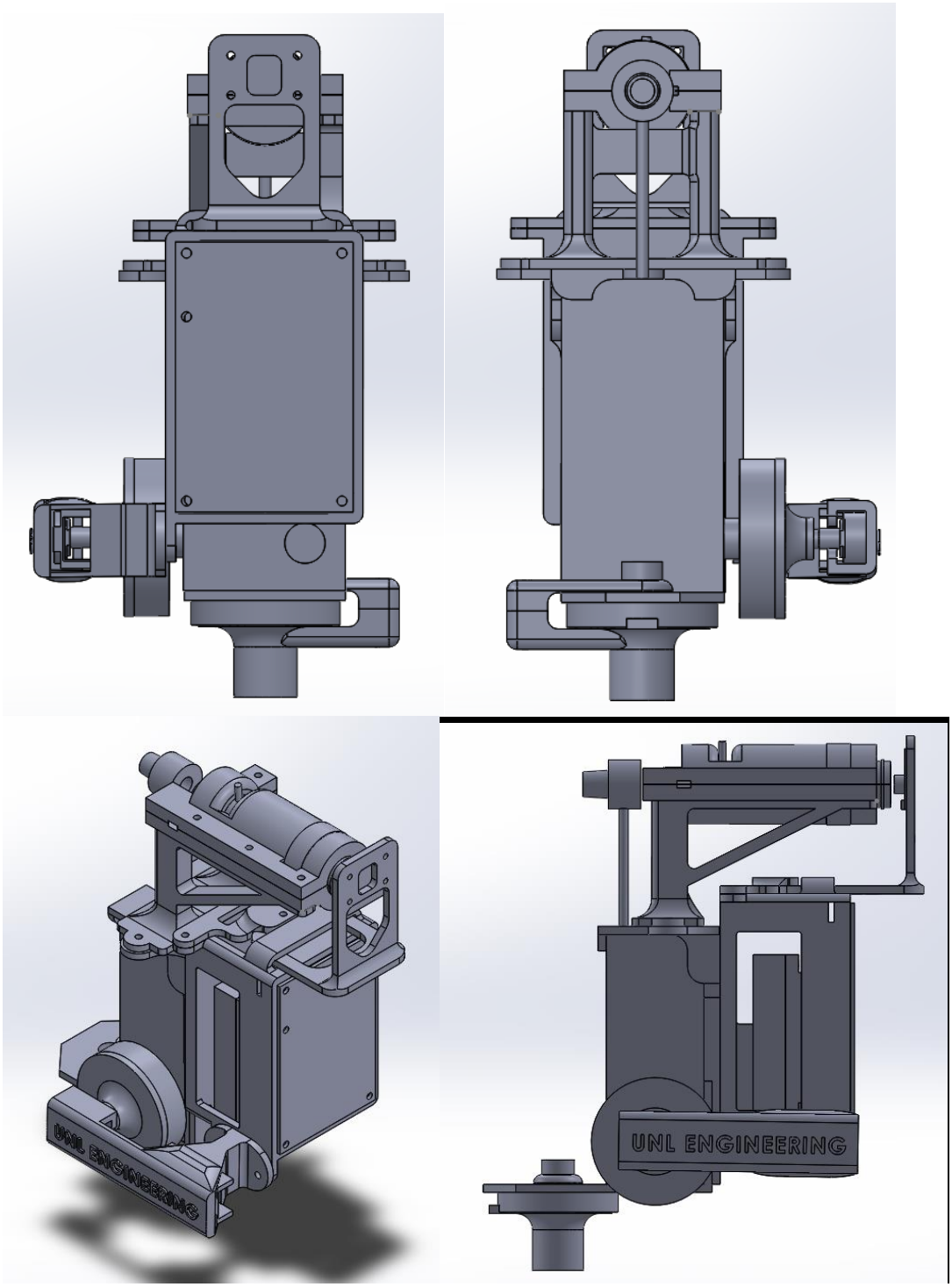
Iteration 3:

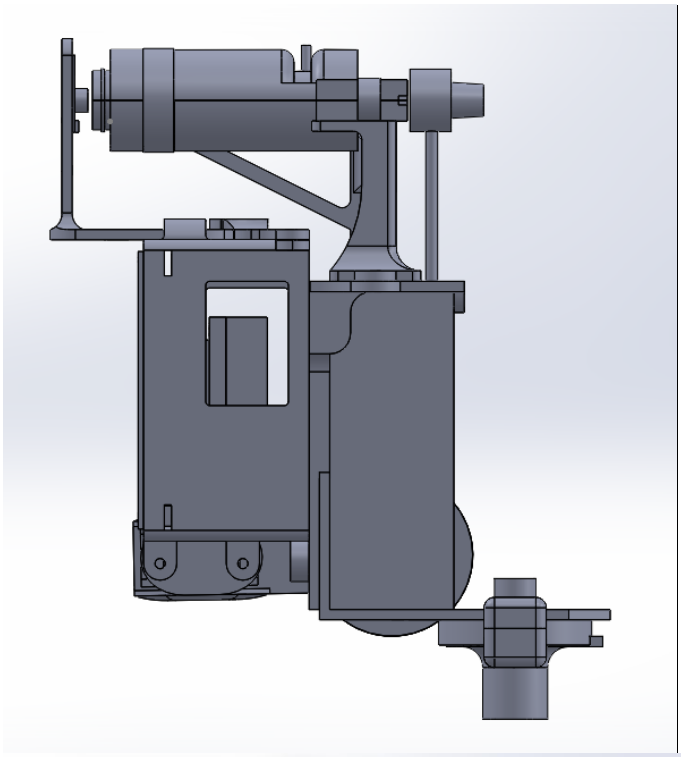


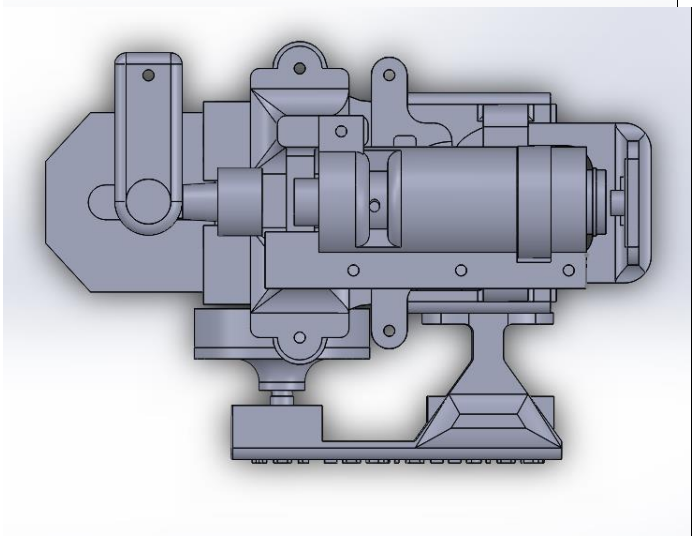




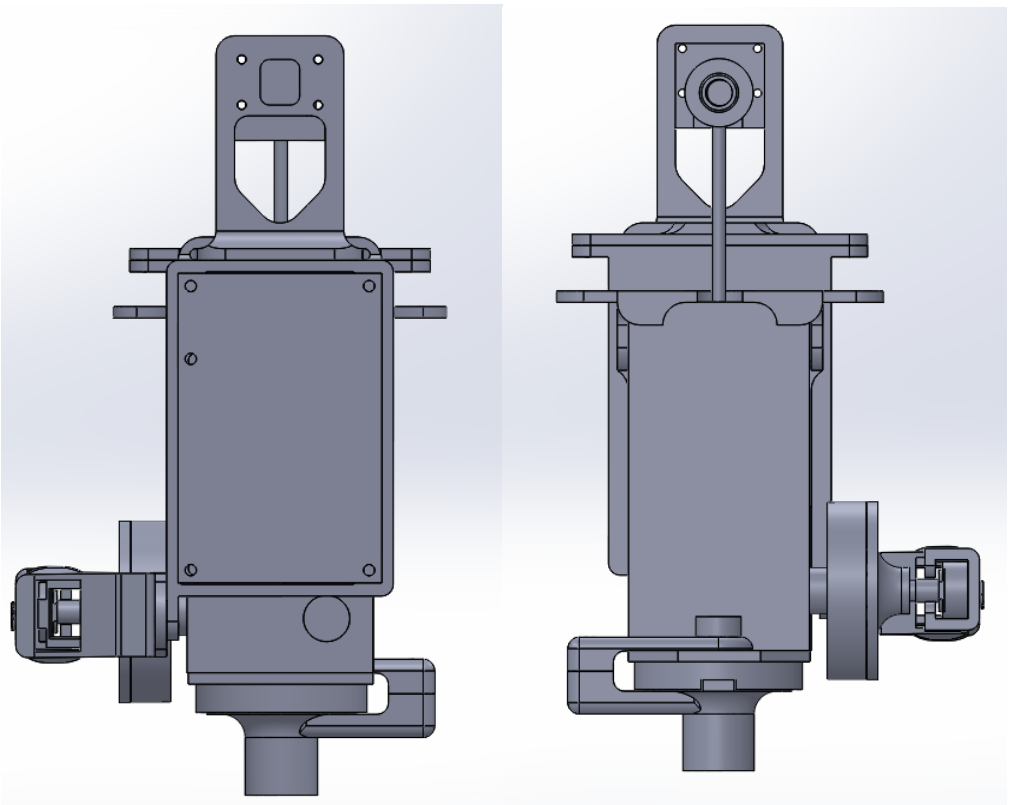
Iteration 4:

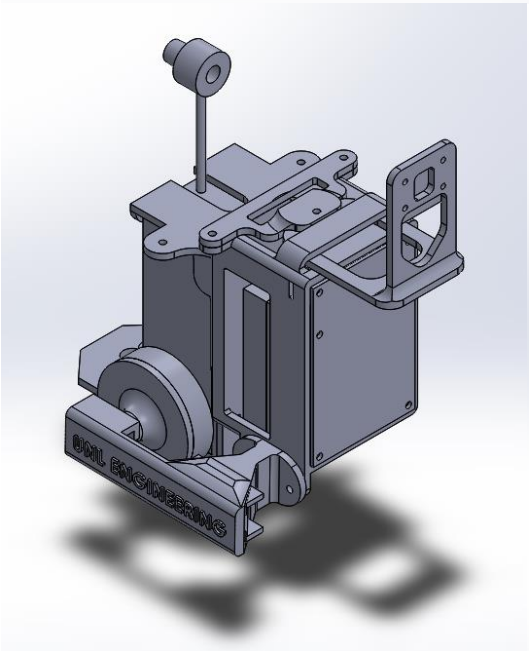


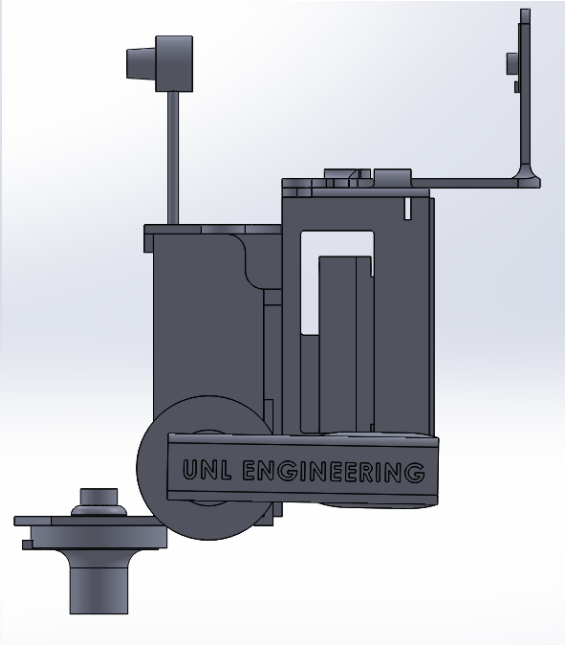


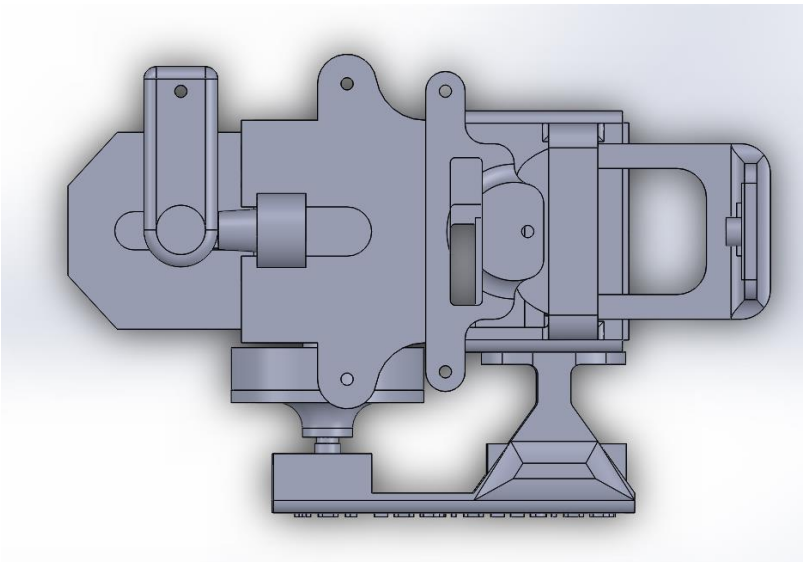


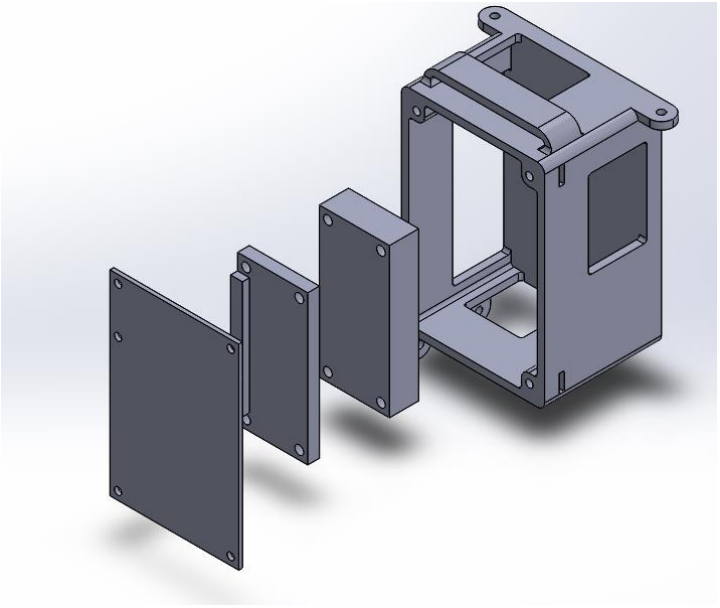
Iteration 5:

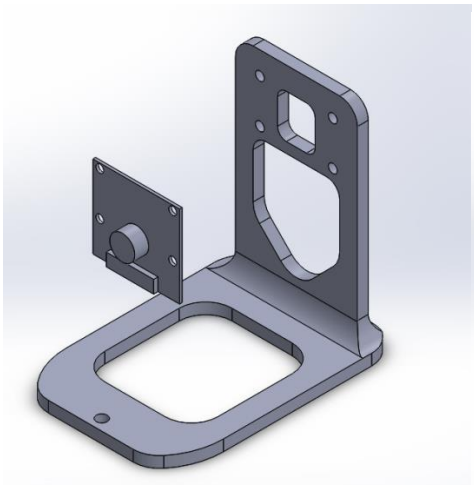


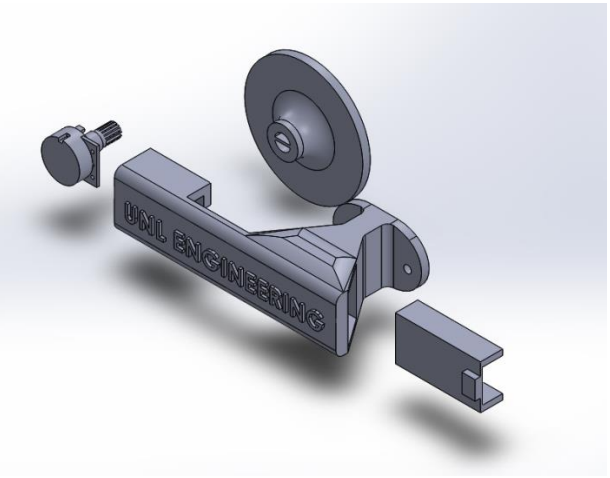

UNL ENGINEERING

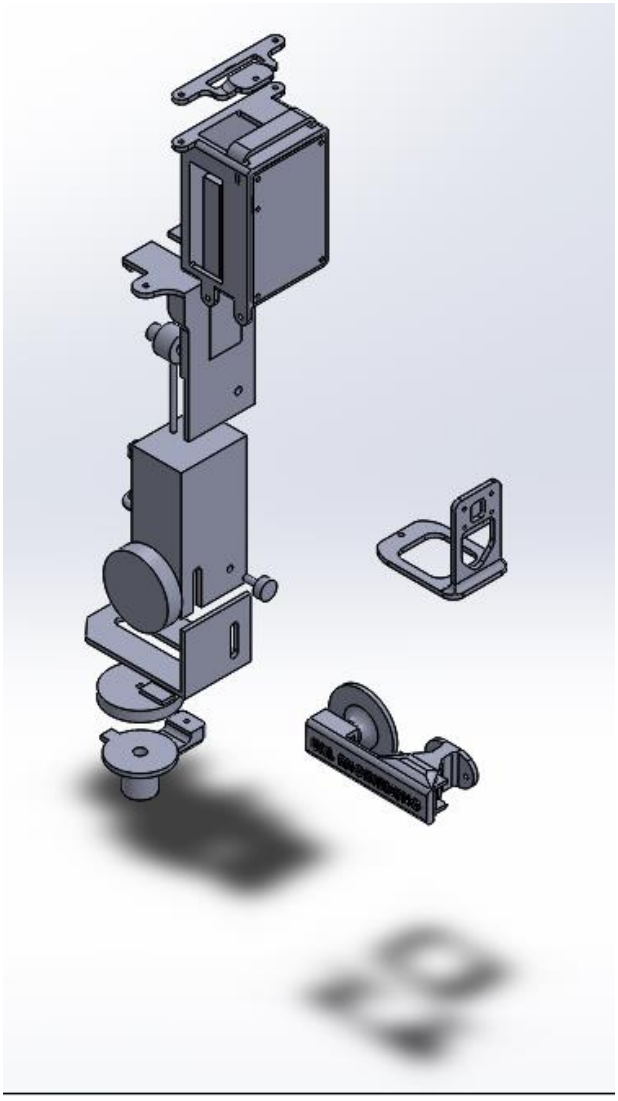

UNL ENGINEERING








UNL ENGINEERING



APPENDIX 3:

This appendix holds all of the code required for the current version of the Shape Detection program.

ShapeRecognGuiTest.m:

```
function varargout = ShapeRecognGuiTest(varargin)
% SHAPERECOGNGUITEST MATLAB code for ShapeRecognGuiTest.fig
%      SHAPERECOGNGUITEST, by itself, creates a new SHAPERECOGNGUITEST
or raises the existing
%      singleton*.
%
%      H = SHAPERECOGNGUITEST returns the handle to a new
SHAPERECOGNGUITEST or the handle to
%      the existing singleton*.
%
%      SHAPERECOGNGUITEST('CALLBACK',hObject,eventData,handles,...)
calls the local
%      function named CALLBACK in SHAPERECOGNGUITEST.M with the given
input arguments.
%
%      SHAPERECOGNGUITEST('Property','Value',...) creates a new
SHAPERECOGNGUITEST or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before ShapeRecognGuiTest_OpeningFcn gets
called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to ShapeRecognGuiTest_OpeningFcn
via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help ShapeRecognGuiTest

% Last Modified by GUIDE v2.5 02-Aug-2016 15:08:44

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @ShapeRecognGuiTest_OpeningFcn,
...
                   'gui_OutputFcn',  @ShapeRecognGuiTest_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
```

```
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before ShapeRecognGuiTest is made visible.
function ShapeRecognGuiTest_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to ShapeRecognGuiTest (see
VARARGIN)

% Choose default command line output for ShapeRecognGuiTest
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
%% Clear all the data
echo off; %Doesn't display code the user doesn't need
clc; %Clears the command window For the User
clear vars; %Cleans up any previous data
global testNum;
testNum=1;
%% Set up all the images
SetupOverlay=imread('CoMands\Overlay.PNG');
imshow(SetupOverlay, 'Parent', handles.ShapeOverlay);
SetupOverlay=imread('CoMands\Circle.PNG');
imshow(SetupOverlay, 'Parent', handles.Circle);
SetupOverlay=imread('CoMands\Square.PNG');
imshow(SetupOverlay, 'Parent', handles.Square);
SetupOverlay=imread('CoMands\Triangle.PNG');
imshow(SetupOverlay, 'Parent', handles.Triangle);
SetupOverlay=imread('CoMands\Blank.PNG');
imshow(SetupOverlay, 'Parent', handles.ShapeResult);
%% Information for the rest of the program
global inifilename;
inifilename='config\Settings.ini';
inputimage=SetupOverlay;
global imsize
imsize=size(inputimage);

% UIWAIT makes ShapeRecognGuiTest wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
```

```
function varargout = ShapeRecognGuiTest_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function TextResults_Callback(hObject, eventdata, handles)
% hObject    handle to TextResults (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of TextResults as text
%        str2double(get(hObject,'String')) returns contents of
TextResults as a double


% --- Executes during object creation, after setting all properties.
function TextResults_CreateFcn(hObject, eventdata, handles)
% hObject    handle to TextResults (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in Start.
function Start_Callback(hObject, eventdata, handles)
% hObject    handle to Start (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global inifilename;
set(handles.TextResults, 'String', 'Drawing....');
SetupOverlay=imread('CoMands\Blank.PNG');
imshow(SetupOverlay, 'Parent', handles.ShapeResult);
imshow(SetupOverlay, 'Parent', handles.ActualDrawn);
Calibration=cell2mat(inifile(inifilename,'read',{'GazePointAPI','Calibr
ation','Calibrate','d','ERR'})); %Set Calibration to 1 for test
TotalBlinktime=cell2mat(inifile(inifilename,'read',{'GazePointAPI','Cal
ibration','TotalBlinktime','d','ERR'}));%Reccomended setting is to be 8
[tx1,ty1]=GazePointApi(Calibration,TotalBlinktime);
```

```
global recordX;
recordX=tx1;
global recordY;
recordY=ty1;
set(handles.TextResults, 'String', 'Processing....');
[position]=FindtheCenter(tx1,ty1,2); %Multiple options for What center
technique is used
NumAvG=cell2mat(inifile(inifilename,'read',{'DataFilter','NumbertoAvg',
'Number','d','ERR'}));
[x1,y1]=DataFilter(tx1,ty1,position,NumAvG); %Standarddeviation filter
with average sum
[shape,Value]=ShapeRecognFnc(x1,y1,position);
set(handles.TextResults, 'String', strcat(shape,' with SPSE of
',num2str(Value)));
global SPSE;
SPSE=Value;
if strcmp(shape,'Circle')==1
    SetupOverlay=imread('CoMands\Circle.PNG');
    imshow(SetupOverlay, 'Parent', handles.ShapeResult);
end
if strcmp(shape,'Square')==1
    SetupOverlay=imread('CoMands\Square.PNG');
    imshow(SetupOverlay, 'Parent', handles.ShapeResult);
end
if strcmp(shape,'Triangle')==1
    SetupOverlay=imread('CoMands\Triangle.PNG');
    imshow(SetupOverlay, 'Parent', handles.ShapeResult);
end
FilledImg=DISPXY(x1,y1);
global TransferShape;
TransferShape=shape;

imshow(FilledImg, 'Parent', handles.ActualDrawn);



% --- Executes on button press in OpenGazeControl.
function OpenGazeControl_Callback(hObject, eventdata, handles)
% hObject    handle to OpenGazeControl (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ProgramLaunher();



% --- Executes on button press in Calibrate.
function Calibrate_Callback(hObject, eventdata, handles)
% hObject    handle to Calibrate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Calibration=1;
TotalBlinktime=1;
GazePointApi(Calibration,TotalBlinktime);
```

```
% --- Executes on button press in EXIT.
function EXIT_Callback(hObject, eventdata, handles)
% hObject    handle to EXIT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close all


% --- Executes during object creation, after setting all properties.
function ActualDrawn_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ActualDrawn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: place code in OpeningFcn to populate ActualDrawn


% % --- Executes on button press in robotsend.
% function robotsend_Callback(hObject, eventdata, handles)
% % hObject    handle to robotsend (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    structure with handles and user data (see GUIDATA)
% global TransferShape;
% Shape=TransferShape;
% port=str2num(get(handles.PortTag, 'String'));
% BaudRate=str2num(get(handles.BaudTag, 'String'));
% set(handles.TextResults, 'String', 'Sending....');
% SendRobot(Shape,port,BaudRate)
% set(handles.TextResults, 'String', 'Ready!');







% function PortTag_Callback(hObject, eventdata, handles)
% % hObject    handle to PortTag (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    structure with handles and user data (see GUIDATA)
%
% % Hints: get(hObject,'String') returns contents of PortTag as text
% %        str2double(get(hObject,'String')) returns contents of
PortTag as a double
%
%
% % --- Executes during object creation, after setting all properties.
% function PortTag_CreateFcn(hObject, eventdata, handles)
% % hObject    handle to PortTag (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
```

```
% % handles    empty - handles not created until after all CreateFcns
called
%
% % Hint: edit controls usually have a white background on Windows.
% %       See ISPC and COMPUTER.
% if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
%     set(hObject,'BackgroundColor','white');
% end
%
%
%
% function BaudTag_Callback(hObject, eventdata, handles)
% % hObject    handle to BaudTag (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    structure with handles and user data (see GUIDATA)
%
% % Hints: get(hObject,'String') returns contents of BaudTag as text
% %        str2double(get(hObject,'String')) returns contents of
BaudTag as a double
%
%
% % --- Executes during object creation, after setting all properties.
% function BaudTag_CreateFcn(hObject, eventdata, handles)
% % hObject    handle to BaudTag (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    empty - handles not created until after all CreateFcns
called
%
% % Hint: edit controls usually have a white background on Windows.
% %       See ISPC and COMPUTER.
% if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
%     set(hObject,'BackgroundColor','white');
% end



function UserID_Callback(hObject, eventdata, handles)
% hObject    handle to UserID (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of UserID as text
%        str2double(get(hObject,'String')) returns contents of UserID
as a double


% --- Executes during object creation, after setting all properties.
function UserID_CreateFcn(hObject, eventdata, handles)
% hObject    handle to UserID (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in SaveResults.
function SaveResults_Callback(hObject, eventdata, handles)
% hObject    handle to SaveResults (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
IDNumber=get(handles.UserID, 'String');
global recordX;
trecordX=recordX;
global recordY;
trecordY=recordY;
global SPSE;
tSPSE=SPSE;
global testNum;
SaveToFiles(IDNumber,trecordX,trecordY,tSPSE,testNum);
testNum=testNum+1;
```

Settings.ini:

```
[ShapeRecogn]
{Image Resolution}
width=450
height=600

{Augmented Shape}
Marker=ON
Markersize=20
radius=45
LineWidth=5
Circle=ON
Square=ON
Triangle=ON

{Bound Size}
bounds=16

{Weighted Percentage}
Scalar=10

{Save File}
Save=OFF
```

{Disp Results}

Percentages=OFF

NUMOFCORNERS=OFF

SPSE=ON

{Create Results File}

ResultsFile=OFF

[Corner Detection]

{Harris}

MaxCorners=8

QualityLevel=.48

SensitivityFactor=.04

{SPSE WEIGHTS}

LOWCIRCLE=4

HIGHCIRCLE=6

LOWSQUARE=1

HIGHSQUARE=2

LOWTRIANGLE=1

HIGHTRIANGLE=2

LWEIGHT=1

HWEIGHT=2

[ShapeDetect]

{DataSet Options}

UseDataSet=ON

{Disp Results}

medval=OFF

diffarea=OFF

diffmeanvals=OFF

diffslopes=OFF

{MEDVAL}

BESTWEIGHT=3

OTHERWEIGHT=1

{Differnt Area}

Scale=4

{MeanValues}

howfar=1/3

Scale=4.5

{Slope}

avgpoints=5

howfar=3/5

scale=3

[FilterMaker]

{Correction Values}

Circle=0

Square=6

Triangle=6

[HoughAssist]

{Stationaryvalues}

topradius=2000

splits=10

[DataTableRead]

{ON OFF}

Data=OFF

{CurrentTest}

TESTNUM=1

{MASSTESER}

Tester=OFF

mintest=1

maxtest=60

[GazePointAPI]

{UseEyeGaze}

Data=ON

{Calibration}

Calibrate=2

TotalBlinktime=8

[DataFilter]

{NumbertoAvg}

Number=10

Inifile.m:

```
function varargout = inifile(varargin)
%INIFILE Creates, reads, or writes data from/to a standard ini (ascii)
%        file. Such a file is organized into sections
%       ([section name]), subsections(enclosed by {subsection name}),
%       and keys (key=value).  Empty lines and lines with the first
non-empty
%       character being ; (comment lines) are ignored.
%
%   Usage:
%       INIFILE(fileName,'new')
%           Rewrites an existing file - creates a new, empty file.
%
%       INIFILE(fileName,'write',keys,<style>)
%           Writes keys given as cell array of strings (see description
of
%           the keys below). Optional style variable: 'tabbed' writes
sections,
%           subsections and keys in a tabbed style to get more readable
%           file. The 'plain' style is the default style. This only
affects
%           the keys that will be written/rewritten.
%
%       INIFILE(fileName,'deletekeys',keys)
%           Deletes keys and their values - if they exist.
%
%       [readsett,result] = INIFILE(fileName,'read',keys)
%           Reads the values of the keys where readsett is a cell array
of
%           strings and/or numeric values of the keys. If any of the
keys
%           is not found, the default value is returned (if given in
the
%           5-th column of the keys parameter). result is a cell array
of
%           strings - one for each key read; empty if OK, error/warning
%           string if error; in both cases an empty string is returned
in
%           readsett{i} for the i-th key if error.
%
%       [keys,sections,subsections] = INIFILE(fName,'readall')
%           Reads entire file and returns all the sections, subsections
%           and keys found.
%
%
%   Notes on the keys cell array given as an input parameter:
%           Cell array of STRINGS; either 3, 4, or 5 columns.
%           Each row has the same number of columns. The columns are:
%           'section':      section name string (the root is considered
if
%                           empty)
%           'subsection':   subsection name string (the root is
considered
```

```
%                               if empty)
%           'key':              name of the field to write/read from (given
as
%                               a string).
%           value:              (optional) STRING or NUMERIC value (scalar
or
%                               matrix) to be written to the
%                               ini file in the case of 'write' operation
OR
%                               conversion CHAR for read operation:
%                               'i' for integer, 'd' for double, 's' or
%                               '' or not given for string (default).
%           defaultValue:       (optional) string or numeric value (scalar
or
%                               matrix) that is returned when the key is
not
%                               found or an empty value is found
%                               when reading ('read' operation).
%                               If the defaultValue is not given and the
key
%                               is not found, an empty value is returned.
%                               It MUST be in the format as given by the
%                               value, e.g. if the value = 'i' it must be
%                               given as an integer etc.
%
%
%   EXAMPLE:
%       Suppose we want a new ini file, test1.ini with 4 fields,
including a
%       5x5 matrix (see below). We can write the 5 fields into the ini
file
%       using:
%
%       x = rand(5);    % matrix data
%       inifile('test1.ini','new');
%       writeKeys = {'measurement','person','name','Primoz Cermelj';...
%                    'measurement','protocol','id',1;...
%                    'application','','description.m1','some...';...
%                    'application','','description.m2','some...';...
%                    'data','','x',x};
%       inifile('test1.ini','write',writeKeys,'plain');
%
%       Later, you can read them out. Additionally, if any of them
won't
%       exist, a default value will be returned (if the 5-th column is
given
%       for all the rows as below).
%
%       readKeys = {'measurement','person','name','','John Doe';...
%                    'measurement','protocol','id','i',0;...
%                    'application','','description.m1','','none';...
%                    'application','','description.m2','','none';...
%                    'data','','x','d',zeros(5)};
%       readSett = inifile('test1.ini','read',readKeys);
```

```
%
%       Or, we can just read all the keys out
%       [keys,sections,subsections] = inifile(test1.ini,'readall');
%
%
%   NOTES: If the operation is 'write' and the file is empty or does
not
%   exist, a new file is created. When writing and if any of the
section
%   or subsection or key does not exist, it creates (adds) a new one.
%   Everything but value is NOT case sensitive. Given keys and values
%   will be trimmed (leading and trailing spaces will be removed).
%   Any duplicates (section, subsection, and keys) are ignored. Empty
%   section and/or subsection can be given as an empty string, '',
%   but NOT as an empty matrix, [].
%
%   Numeric matrices can be represented as strings in one of the two
form:
%   '1 2 3;4 5 6' or '1,2,3;4,5,6' (an example).
%
%   Comment lines starts with ; as the first non-empty character but
%   comments can not exist as a tail to a standard, non-comment line as
;
%   is also used as a row delimiter for matrices.
%
%   This function was tested on the win32 platform only but it should
%   also work on Unix/Linux platforms. Since some short-circuit
operators
%   are used, at least Matlab 6.5 (R13) is required.
%
%
%   First release on 29.01.2003
%   (c) Primoz Cermelj, Slovenia
%   Contact: primoz.cermelj@gmail.com
%   Download location:
http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectI
d=2976&objectType=file
%
%   Version: 1.4.2
%   Last revision: 12.01.2007
%
%   Bug reports, questions, etc. can be sent to the e-mail given above.
%
%   ACKNOWLEDGEMENTS: Thanks to Diego De Rosa for a suggestion/fix how
to
%   read the value when the key is found but empty.
%-------------------------------------------------------------------------
----

%----------------
% INIFILE history
%----------------
%
% [v.1.4.2] 12.01.2007
```

```
% - FIX: When in read mode and a certain key is found but the value is
%        empty, the default value will be used instead.
%
% [v.1.4.1] 12.01.2006
% - FIX: Some minor refinements (speed,...)
%
% [v.1.4.0] 05.12.2006
% - NEW: New 'readall' option added which reads all the sections,
%        subsections and keys out
%
% [v.1.3.2 - v.1.3.5] 25.08.2004
% - NEW: Speed improvement for large files - using fread and fwrite
instead
%        of fscanf and fprintf, respectively
% - NEW: Some minor changes
% - NEW: Writing speed-up
% - NEW: New-line chars are properly set for pc, unix, and mac
%
% [v.1.3.1] 04.05.2004
% - NEW: Comment lines are detected and thus ignored; comment lines are
%        lines with first non-empty character being ;
% - NEW: Lines not belonging to any of the recognized types (key,
section,
%        comment,...) raise an error.
%
% [v.1.3.0] 21.04.2004
% - NEW: 2D Numeric matrices can be read/written
% - FIX: Bug related to read operation and default value has been
removed
%
% [v.1.2.0] 30.04.2004
% - NEW: Automatic conversion capability (integers, doubles, and
strings)
%        added for read and write operations
%
% [v.1.1.0] 04.02.2004
% - FIX: 'writetext' option removed (there was a bug previously)
%
% [v.1.01b] 19.12.2003
% - NEW: A new concept - multiple keys can now be read, written, or
deleted
%        ALL AT ONCE which makes this function much faster. For
example, to
%        write 1000 keys, using previous versions it took 157 seconds
on a
%        1.5 GHz machine, with this new version it took only 0.9
seconds.
%        In general, the speed improvement is greater when a larger
number of
%        read/written keys is considered (with respect to the older
version).
% - NEW: The format of the input parameters has changed. See above.
%
% [v.0.97] 19.11.2003
```

```
% - NEW: Additional m-function, strtrim, is no longer needed
%
% [v.0.96] 16.10.2003
% - FIX: Detects empty keys
%
% [v.0.95] 04.07.2003
% - NEW: 'deletekey' option/operation added
% - FIX: A major file refinement to obtain a more compact utility ->
%        additional operations can "easily" be implemented
%
% [v.0.91-0.94]
% - FIX: Some minor refinements
%
% [v.0.90] 29.01.2003
% - NEW: First release of this tool
%
%----------------

global NL_CHAR;

% Checks the input arguments
if nargin == 0
    disp('INIFILE v1.4.2');
    disp('Copyright (c) 2003-2007 Primoz Cermelj');
    disp('This is FREE SOFTWARE');
    disp('Type <help inifile> to get more help on its usage');
    return
elseif nargin < 2
    error('Not enough input arguments');
end

fileName = varargin{1};
operation = varargin{2};

if (strcmpi(operation,'read')) | (strcmpi(operation,'deletekeys'))
    if nargin < 3
        error('Not enough input arguments.');
    end
    if ~exist(fileName)
        error(['File ' fileName ' does not exist.']);
    end
    keys = varargin{3};
    [m,n] = size(keys);
    if n < 3
        error('Keys argument must have at least 3 columns for read
operation');
    end
    for ii=1:m
        if isempty(keys(ii,3)) | ~ischar(keys{ii,3})
            error('Empty or non-char keys are not allowed.');
        end
    end
elseif (strcmpi(operation,'write'))
    if nargin < 3
```

```
        error('Not enough input arguments');
    end
    keys = varargin{3};
    if nargin < 4 || isempty(varargin{4})
        style = 'plain';
    else
        style = varargin{4};
        if ~(strcmpi(style,'plain') | strcmpi(style,'tabbed')) |
~ischar(style)
            error('Unsupported style given or style not given as a
string');
        end
    end
    [m,n] = size(keys);
    if n < 4
        error('Keys argument requires 4 columns for write operation');
    end
    for ii=1:m
        if isempty(keys(ii,3)) | ~ischar(keys{ii,3})
            error('Empty or non-char keys are not allowed.');
        end
    end
elseif (strcmpi(operation,'readall'))
    %
elseif (~strcmpi(operation,'new'))
    error(['Unknown inifile operation: ''' operation '''']);
end
if nargin >= 3
    for ii=1:m
        for jj=1:3
            if ~ischar(keys{ii,jj})
                error('All cells from the first 3 columns must be given
as strings, even the empty ones.');
            end
        end
    end
end


% Sets the new-line character/string
if ispc
    NL_CHAR = '\r\n';
elseif isunix
    NL_CHAR = '\n';
else
    NL_CHAR = '\r';
end

readsett = [];
result = [];

%----------------------------
% CREATES a new, empty file (rewrites an existing one)
%----------------------------
```

```
if strcmpi(operation,'new')
    fh = fopen(fileName,'w');
    if fh == -1
        error(['File: ''' fileName ''' can not be (re)created']);
    end
    fclose(fh);
    return

%----------------------------
% READS the whole data (all keys)
%----------------------------
elseif (strcmpi(operation,'readall'))
    [keys,sections,subsections] = readallkeys(fileName);
    varargout(1) = {keys};
    varargout(2) = {sections};
    varargout(3) = {subsections};
    return

%----------------------------
% READS key-value pairs out
%----------------------------
elseif (strcmpi(operation,'read'))
    result = cell(m,1);
    if n >= 4
        conversionOp = keys(:,4);       % conversion operation: 'i',
'd', or 's' ('') - for each key to be read
    else
        conversionOp = cellstrings(m,1);
    end
    if n < 5
        defaultValues = cellstrings(m,1);
    else
        defaultValues = keys(:,5);
    end
    readsett = defaultValues;
    keysIn = keys(:,1:3);
    [secsExist,subsecsExist,keysExist,readValues,so,eo] =
findkeys(fileName,keysIn);
    ind = find(keysExist);
    % For those keys that exist but have empty values, replace them
with
    % the default values
    if ~isempty(ind)
        ind_empty = zeros(size(ind));
        for kk = 1:size(ind,1)
            ind_empty(kk) = isempty(readValues{ind(kk)});
        end
        ind(find(ind_empty)) = [];
        readsett(ind) = readValues(ind);
    end
    % Now, go through all the keys and do the conversion if the
conversion
    % char is given
    for ii=1:m
```

```
        if ~isempty(conversionOp{ii}) & ~strcmpi(conversionOp{ii},'s')
            if strcmpi(conversionOp{ii},'i') |
strcmpi(conversionOp{ii},'d')
                if ~isnumeric(readsett{ii})
                    readsett{ii} = str2num(readsett{ii});
                end
                if strcmpi(conversionOp{ii},'i')
                    readsett{ii} = round(readsett{ii});
                end
                if isempty(readsett{ii})
                    result{ii} = [num2str(ii) '-th key ' keysIn{ii,3}
'or given defaultValue could not be converted using '''
conversionOp{ii} ''' conversion'];
                end
            else
                error(['Invalid conversion char given: '
conversionOp{ii}]);
            end
        end
    end
    varargout(1) = {readsett};
    varargout(2) = {result};
    return

%----------------------------
% WRITES key-value pairs to an existing or non-existing
% file (file can even be empty)
%----------------------------
elseif (strcmpi(operation,'write'))
    if m < 1
        error('At least one key is needed when writing keys');
    end
    if ~exist(fileName)
        inifile(fileName,'new');
    end
    for ii=1:m  % go through ALL the keys and convert them to strings
        keys{ii,4} = n2s(keys{ii,4});
    end
    writekeys(fileName,keys,style);
    return

%----------------------------
% DELETES key-value pairs out
%----------------------------
elseif (strcmpi(operation,'deletekeys'))
    deletekeys(fileName,keys);



else
    error('Unknown operation for INIFILE.');
end
```

```
%--------------------------------------------------
%%%%%%%%%%%%% SUBFUNCTIONS SECTION %%%%%%%%%%%%%%%
%--------------------------------------------------


%-------------------------------------
function
[secsExist,subSecsExist,keysExist,values,startOffsets,endOffsets] =
findkeys(fileName,keysIn)
% This function parses ini file for keys as given by keysIn. keysIn is
a cell
% array of strings having 3 columns; section, subsection and key in
each row.
% section and/or subsection can be empty (root section or root
subsection)
% but the key can not be empty. The startOffsets and endOffsets are
start and
% end bytes that each key occuppies, respectively. If any of the keys
doesn't exist,
% startOffset and endOffset for this key are the same. A special case
is
% when the key that doesn't exist also corresponds to a non-existing
% section and non-existing subsection. In such a case, the startOffset
and
% endOffset have values of -1.

nKeys = size(keysIn,1);          % number of keys
nKeysLocated = 0;                % number of keys located
secsExist = zeros(nKeys,1);      % if section exists (and is non-empty)
subSecsExist = zeros(nKeys,1);   % if subsection...
keysExist = zeros(nKeys,1);      % if key that we are looking for exists
keysLocated = keysExist;         % if the key's position (existing or
non-existing) is LOCATED
values = cellstrings(nKeys,1);   % read values of keys (strings)
startOffsets = -ones(nKeys,1);   % start byte-position of the keys
endOffsets = -ones(nKeys,1);     % end byte-position of the keys

keyInd = find(strcmpi(keysIn(:,1),''));  % key indices having []
section (root section)

line = [];
lineN = 0;                       % line number
currSection = '';
currSubSection = '';

fh = fopen(fileName,'r');
if fh == -1
    error(['File: ''' fileName ''' does not exist or can not be
opened.']);
end
```

```
try
    %--- Searching for the keys - their values and start and end
locations in bytes
    while 1

        pos1 = ftell(fh);
        line = fgetl(fh);
        if line == -1               % end of file, exit
            line = [];
            break
        end
        lineN = lineN + 1;
        [status,readValue,readKey] = processiniline(line);
        if (status == 1)            % (new) section found
            % Keys that were found as belonging to any previous section
            % are now assumed as located (because another
            % section is found here which could even be a repeated one)
            keyInd = find( ~keysLocated &
strcmpi(keysIn(:,1),currSection) );
            if length(keyInd)
                keysLocated(keyInd) = 1;
                nKeysLocated = nKeysLocated + length(keyInd);
            end
            currSection = readValue;
            currSubSection = '';
            % Indices to non-located keys belonging to current section
            keyInd = find( ~keysLocated &
strcmpi(keysIn(:,1),currSection) );
            if ~isempty(keyInd)
                secsExist(keyInd) = 1;
            end
            pos2 = ftell(fh);
            startOffsets(keyInd) = pos2+1;
            endOffsets(keyInd) = pos2+1;
        elseif (status == 2)        % (new) subsection found
            % Keys that were found as belonging to any PREVIOUS section
            % and/or subsection are now assumed as located (because
another
            % subsection is found here which could even be a repeated
one)
            keyInd = find( ~keysLocated &
strcmpi(keysIn(:,1),currSection) & ~keysLocated &
strcmpi(keysIn(:,2),currSubSection));
            if length(keyInd)
                keysLocated(keyInd) = 1;
                nKeysLocated = nKeysLocated + length(keyInd);
            end
            currSubSection = readValue;
            % Indices to non-located keys belonging to current section
and subsection at the same time
            keyInd = find( ~keysLocated &
strcmpi(keysIn(:,1),currSection) & ~keysLocated &
strcmpi(keysIn(:,2),currSubSection));
```

```
            if ~isempty(keyInd)
                subSecsExist(keyInd) = 1;
            end
            pos2 = ftell(fh);
            startOffsets(keyInd) = pos2+1;
            endOffsets(keyInd) = pos2+1;
        elseif (status == 3)        % key found
            if isempty(keyInd)
                continue            % no keys from 'keys' - from
section-subsection par currently in
            end
            currKey = readValue;
            pos2 = ftell(fh);       % the last-byte position of the
read key  - the total sum of chars read so far
            for ii=1:length(keyInd)
               if strcmpi( keysIn(keyInd(ii),3),readKey ) &
~keysLocated(keyInd(ii))
                   keysExist(keyInd(ii)) = 1;
                   startOffsets(keyInd(ii)) = pos1+1;
                   endOffsets(keyInd(ii)) = pos2;
                   values{keyInd(ii)} = currKey;
                   keysLocated(keyInd(ii)) = 1;
                   nKeysLocated = nKeysLocated + 1;
               else
                   if ~keysLocated(keyInd(ii))
                       startOffsets(keyInd(ii)) = pos2+1;
                       endOffsets(keyInd(ii)) = pos2+1;
                   end
               end
            end
            if nKeysLocated >= nKeys  % if all the keys are located
stop the searching
                break
            end
        else                            % general text found (even empty
line(s))
            if (status == -1)
                error(['unknown string found at line '
num2str(lineN)]);
            end
        end
    %--- End of searching
    end
    fclose(fh);
catch
    fclose(fh);
    error(['Error parsing the file for keys: ' fileName ': ' lasterr]);
end
%-----------------------------------




%-----------------------------------
```

```
function writekeys(fileName,keys,style)
% Writes keys to the section and subsection pair
% If any of the keys doesn't exist, a new key is added to
% the end of the section-subsection pair otherwise the key is updated
(changed).
% Keys is a 4-column cell array of strings.

global NL_CHAR;

RETURN = sprintf('\r');
NEWLINE = sprintf('\n');

[m,n] = size(keys);
if n < 4
    error('Keys to be written are given in an invalid format.');
end

% Get keys position first using findkeys
keysIn = keys;
[secsExist,subSecsExist,keysExist,readValues,so,eo] =
findkeys(fileName,keys(:,1:3));

% Read the whole file's contents out
fh = fopen(fileName,'r');
if fh == -1
    error(['File: ''' fileName ''' does not exist or can not be
opened.']);
end
try
    dataout = fread(fh,'char=>char')';
catch
    fclose(fh);
    error(lasterr);
end
fclose(fh);

%--- Rewriting the file -> writing the refined contents
fh = fopen(fileName,'w');
if fh == -1
    error(['File: ''' fileName ''' does not exist or can not be
opened.']);
end
try
    tab1 = [];
    if strcmpi(style,'tabbed')
        tab1 = sprintf('\t');
    end
    % Proper sorting of keys is cruical at this point in order to avoid
    % inproper key-writing.

    % Find keys with -1 offsets - keys with non-existing section AND
    % subsection - keys that will be added to the end of the file
    fs = length(dataout);       % file size in bytes
```

```
    nAddedKeys = 0;
    ind = find(so==-1);
    if ~isempty(ind)
        so(ind) = (fs+10);      % make sure these keys will come to the
end when sorting
        eo(ind) = (fs+10);
        nAddedKeys = length(ind);
    end

    % Sort keys according to start- and end-offsets
    [dummy,ind] = sort(so,1);
    so = so(ind);
    eo = eo(ind);
    keysIn = keysIn(ind,:);
    keysExist = keysExist(ind);
    secsExist = secsExist(ind);
    subSecsExist = subSecsExist(ind);
    readValues = readValues(ind);
    values = keysIn(:,4);

    % Find keys with equal start offset (so) and additionally sort them
    % (locally). These are non-existing keys, including the ones whose
    % section and subsection will also be added.
    nKeys = size(so,1);
    fullInd = 1:nKeys;
    ii = 1;
    while ii < nKeys
        ind = find(so==so(ii));
        if ~isempty(ind) && length(ind) > 1
            n = length(ind);
            from = ind(1);
            to = ind(end);
            tmpKeys = keysIn( ind,: );
            [tmpKeys,ind2] = sortrows( lower(tmpKeys) );
            fullInd(from:to) = ind(ind2);
            ii = ii + n;
        else
            ii = ii + 1;
        end
    end

    % Final (re)sorting
    so = so(fullInd);
    eo = eo(fullInd);
    keysIn = keysIn(fullInd,:);
    keysExist = keysExist(fullInd);
    secsExist = secsExist(fullInd);
    subSecsExist = subSecsExist(fullInd);
    readValues = readValues(fullInd);
    values = keysIn(:,4);

    % Refined data - datain
    datain = [];
```

```
    for ii=1:nKeys      % go through all the keys, existing and non-
existing ones
        if ii==1
            from = 1;   % from byte-offset of original data (dataout)
        else
            from = eo(ii-1);
            if keysExist(ii-1)
                from = from + 1;
            end
        end
        to = min(so(ii)-1,fs);  % to byte-offset of original data
(dataout)

        if ~isempty(dataout)
            datain = [datain dataout(from:to)];    % the lines before
the key
        end

        if length(datain) & (~(datain(end)==RETURN |
datain(end)==NEWLINE))
            datain = [datain, sprintf(NL_CHAR)];
        end

        tab = [];
        if ~keysExist(ii)
            if ~secsExist(ii) && ~isempty(keysIn(ii,1))
                if ~isempty(keysIn{ii,1})
                    datain = [datain sprintf(['%s' NL_CHAR],['['
keysIn{ii,1} ']'])];
                end
                % Key-indices with the same section as this, ii-th key
(even empty sections are considered)
                ind = find( strcmpi( keysIn(:,1), keysIn(ii,1)) );
                % This section exists at all keys  corresponding to the
same section from know on (even the empty ones)
                secsExist(ind) = 1;
            end
            if ~subSecsExist(ii) && ~isempty(keysIn(ii,2))
                if ~isempty( keysIn{ii,2})
                    if secsExist(ii); tab = tab1;  end;
                    datain = [datain sprintf(['%s' NL_CHAR],[tab '{'
keysIn{ii,2} '}'])];
                end
                % Key-indices with the same section AND subsection as
this, ii-th key
                % (even empty sections and subsections are considered)
                ind = find( strcmpi( keysIn(:,1), keysIn(ii,1)) &
strcmpi( keysIn(:,2), keysIn(ii,2)) );
                % This subsection exists at all keys corresponding to
the
                % same section and subsection from know on (even the
empty ones)
                subSecsExist(ind) = 1;
```

```
            end
        end
        if secsExist(ii) & (~isempty(keysIn{ii,1})); tab = tab1;  end;
        if subSecsExist(ii) & (~isempty(keysIn{ii,2})); tab = [tab
tab1];  end;
        datain = [datain sprintf(['%s' NL_CHAR],[tab keysIn{ii,3} ' = '
values{ii}])];
    end
    from = eo(ii);
    if keysExist(ii)
        from = from + 1;
    end
    to = length(dataout);
    if from < to
        datain = [datain dataout(from:to)];
    end
    fwrite(fh,datain,'char');
catch
    fclose(fh);
    error(['Error writing keys to file: ''' fileName ''' : ' lasterr]);
end
fclose(fh);
%-------------------------------------



%-------------------------------------
function deletekeys(fileName,keys)
% Deletes keys and their values out; keys must have at least 3 columns:
% section, subsection, and the key

[m,n] = size(keys);
if n < 3
    error('Keys to be deleted are given in an invalid format.');
end

% Get keys position first
keysIn = keys;
[secsExist,subSecsExist,keysExist,readValues,so,eo] =
findkeys(fileName,keys(:,1:3));

% Read the whole file's contents out
fh = fopen(fileName,'r');
if fh == -1
    error(['File: ''' fileName ''' does not exist or can not be
opened.']);
end
try
    dataout = fread(fh,'char=>char')';
catch
    fclose(fh);
    rethrow(lasterror);
end
```

```
fclose(fh);

%--- Rewriting the file -> writing the refined content
fh = fopen(fileName,'w');
if fh == -1
    error(['File: ''' fileName ''' does not exist or can not be
opened.']);
end
try
    ind = find(keysExist);
    nExistingKeys = length(ind);
    datain = dataout;

    if nExistingKeys
        % Filtering - retain only the existing keys...
        fs = length(dataout);       % file size in bytes
        so = so(ind);
        eo = eo(ind);
        keysIn = keysIn(ind,:);
        % ...and sorting
        [so,ind] = sort(so);
        eo = eo(ind);
        keysIn = keysIn(ind,:);

        % Refined data - datain
        datain = [];

        for ii=1:nExistingKeys  % go through all the existing keys
            if ii==1
                from = 1;   % from byte-offset of original data
(dataout)
            else
                from = eo(ii-1)+1;
            end
            to = so(ii)-1;  % to byte-offset of original data (dataout)

            if ~isempty(dataout)
                datain = [datain dataout(from:to)];    % the lines
before the key
            end
        end
        from = eo(ii)+1;
        to = length(dataout);
        if from < to
            datain = [datain dataout(from:to)];
        end
    end

    fwrite(fh,datain,'char');
catch
    fclose(fh);
    error(['Error deleting keys from file: ''' fileName ''' : '
lasterr]);
```

```
end
fclose(fh);
%-------------------------------------




%-------------------------------------
function [keys,sections,subsections] = readallkeys(fileName)
% Reads all the keys out as well as the sections and subsections

keys = [];
sections = [];
subsections = [];
% Read the whole file's contents out
try
    dataout = textread(fileName,'%s','delimiter','\n');
catch
    error(['File: ''' fileName ''' does not exist or can not be
opened.']);
end
nLines = size(dataout,1);

% Go through all the lines and construct the keys variable
keys = cell(nLines,4);
sections = cell(nLines,1);
subsections = cell(nLines,2);
keyN = 0;
secN = 0;
subsecN = 0;
secStr = '';
subsecStr = '';
for ii=1:nLines
    [status,value,key] = processiniline(dataout{ii});
    if status == 1
        secN = secN + 1;
        secStr = value;
        sections(secN) = {secStr};
    elseif status == 2
        subsecN = subsecN + 1;
        subsecStr = value;
        subsections(subsecN,:) = {secStr,subsecStr};
    elseif status == 3
        keyN = keyN + 1;
        keys(keyN,:) = {secStr,subsecStr,key,value};
    end
end
keys(keyN+1:end,:) = [];
sections(secN+1:end,:) = [];
subsections(subsecN+1:end,:) = [];
%-------------------------------------
```

```
%-------------------------------------
function [status,value,key] = processiniline(line)
% Processes a line read from the ini file and
% returns the following values:
%   - status:  -1   => unknown string found
%               0   => empty line found
%               1   => section found
%               2   => subsection found
%               3   => key-value pair found
%               4   => comment line found (starting with ;)
%   - value:    value-string of a key, section, subsection, comment, or
unknown string
%   - key:      key as string

status = 0;
value = [];
key = [];
line = strim(line);                             % removes any leading and
trailing spaces
if isempty(line)                                % empty line
    return
end
if strcmpi(line(1),';')                         % comment found
    status = 4;
    value = line(2:end);
elseif (line(1) == '[') & (line(end) == ']') & (length(line) >= 3)  %
section found
    value = lower(line(2:end-1));
    status = 1;
elseif (line(1) == '{') &...                    % subsection found
       (line(end) == '}') & (length(line) >= 3)
    value = lower(line(2:end-1));
    status = 2;
else                                            % either key-value pair or
unknown string
    pos = findstr(line,'=');
    if ~isempty(pos)                            % key-value pair found
        status = 3;
        key = lower(line(1:pos-1));
        value = line(pos+1:end);
        key = strim(key);                       % removes any leading and
trailing spaces
        value = strim(value);                   % removes any leading and
trailing spaces
        if isempty(key)                         % empty keys are not
allowed
            status = 0;
            key = [];
            value = [];
        end
    else                                        % unknown string found
        status = -1;
        value = line;
```

```
    end
end


%-------------------------------------
function outstr = strim(str)
% Removes leading and trailing spaces (spaces, tabs, endlines,...)
% from the str string.
if isnumeric(str);
    outstr = str;
    return
end
ind = find( ~isspace(str) );        % indices of the non-space
characters in the str
if isempty(ind)
    outstr = [];
else
    outstr = str( ind(1):ind(end) );
end



%-------------------------------------
function cs = cellstrings(m,n)
% Creates a m x n cell array of empty strings - ''
cs = cell(m,n);
cs(:) = {''};


%-------------------------------------
function y = n2s(x)
% Converts numeric matrix to string representation.
% Example: x given as [1 2;3 4] returns y = '1,2;3;4'
if ischar(x) | isempty(x)
    y = x;
    return
end
[m,n] = size(x);
y = [num2str(x(1,:),'%15.6g')];
for ii=2:m
    y = [y ';' num2str(x(ii,:),'%15.6g')];
end
```

GazePointApi:

```
% Gazepoint Function
%Made by TRevor Craig
%Started 5/17/2016 at 1:08 PM
% This function handles the data from the gazepoint

function [TrueX,TrueY]=GazePointApi(Calibration,TotalBlinktime)

%% Setting up the socket
delay=15;
counter=1;
% Blinkcouter=1;%How long the blink counts for
% TotalBlinktime=8; %This should be adjust for blink length beyond
unintentail
% Calibration=2;
ip = '127.0.0.1';
portnum = 4242; %This may need to be adjusted based on what the current
setting is
InputBufferSize=4096;
obj.ip_address=ip;
obj.port_number = portnum;
obj.client_socket = tcpip(obj.ip_address, obj.port_number);
set(obj.client_socket, 'InputBufferSize', InputBufferSize);
obj.client_socket.Terminator = 'CR/LF';
gazepoint_info = strcat('Connected to:', obj.ip_address, ' on port:',
num2str(obj.port_number), '\n');

%% Open the socket
fopen(obj.client_socket);% This opens the camera connection
fprintf(gazepoint_info);

%% Calibaration
if Calibration==1
    fprintf(obj.client_socket, '<SET ID="CALIBRATE_SHOW" STATE="1"
/>');
    fprintf(obj.client_socket, '<SET ID="CALIBRATE_START" STATE="1"
/>');
    pause(delay);
    fprintf(obj.client_socket, '<SET ID="CALIBRATE_SHOW" STATE="0"
/>');
    fprintf(obj.client_socket, '<SET ID="CALIBRATE_START" STATE="0"
/>');
    fprintf(obj.client_socket, '<GET ID="CALIBRATE_RESULT_SUMMARY"
/>');
    fprintf(obj.client_socket, '<SET ID="ENABLE_SEND_DATA" STATE="0"
/>');
    while (get(obj.client_socket, 'BytesAvailable') > 0)
        results = fscanf(obj.client_socket);
        %Sample of returns   <CAL ID="CALIB_START_PT" PT="5"
CALX="0.1500" CALY="0.1500" />
```

```
        CALIXNum = strfind(results, 'CALX="');
        CALIYNum = strfind(results, 'CALY="');
        if ((~isempty(CALIXNum))&&(~isempty(CALIYNum)));

Calix(counter)=str2double(results((CALIXNum+6):(CALIXNum+5+6)));

Caliy(counter)=str2double(results((CALIYNum+6):(CALIYNum+5+6)));
            counter=counter+1;
        end
        pause(.01);
    end
    fprintf(obj.client_socket, '<SET ID="ENABLE_SEND_DATA" STATE="1"
/>');
    fclose(obj.client_socket); %This closes the port
end

%% POG Data Extraction
if(Calibration==2)
    POGV=1;
    fprintf(obj.client_socket, '<SET ID="ENABLE_SEND_DATA" STATE="0"
/>');
    fprintf(obj.client_socket, '<SET ID="ENABLE_SEND_DATA" STATE="1"
/>');
    counter=1;
    fprintf(obj.client_socket, '<SET ID="ENABLE_SEND_POG_FIX" STATE="1"
/>');
    pause(1);%This pause is needed to send all the commands

    while (get(obj.client_socket, 'BytesAvailable') > 0)
        data = fscanf(obj.client_socket);
        POGXNum = strfind(data, 'FPOGX="');
        POGYNum = strfind(data, 'FPOGY="');
        POGVNUM = strfind(data, 'FPOGV="');
        if ((~isempty(POGXNum))&&(~isempty(POGYNum)));
            POGx(counter)=str2double(data((POGXNum+7):(POGXNum+7+6)));
            POGy(counter)=str2double(data((POGYNum+7):(POGYNum+7+6)));
            counter=counter+1;
        end
        if(~isempty(POGVNUM))
             POGV=int32(str2double(data(POGVNUM+7)));
        end
        pause(0.01);
        if POGV==1
            Blinkcouter=1;
        end
        if POGV==0
            Blinkcouter=Blinkcouter+1;
        end
        if Blinkcouter>=TotalBlinktime
            %This line is what makes it last forever so put theis in
the whileloop to stop it
            fprintf(obj.client_socket, '<SET ID="ENABLE_SEND_DATA"
STATE="0"/>');%This line is what makes it last forever so put theis in
the whileloop to stop it
```

```
            break;
        end
    end
    fprintf(obj.client_socket, '<SET ID="ENABLE_SEND_DATA" STATE="1"
/>');
    disp('PROGRAM STOPPED');
    %% Closes the Connection and then free everything UP!
    fclose(obj.client_socket); %This closes the port

    %% Display the results
    % screensize=[1400,900];
    screensize=[700,450];
    % fillmatrix=zeros(screensize(2),screensize(1));
    counter=1;
    for i=1:numel(POGx)
        x=int32(POGx(i)*screensize(1));
        y=int32(POGy(i)*screensize(2));
        if((x>0)&&(y>0))
            if((x<screensize(1))&&(y<screensize(2)))
                TrueX(counter)=x;
                TrueY(counter)=y;
                counter=counter+1;
    %             fillmatrix(y,x)=255;
            end
        end
    end
    % fillmatrix=zeros(450,700);
    % for i=1:numel(TrueX);
    %     fillmatrix(TrueY(i),TrueX(i))=255;
    % end
    % imshow(fillmatrix);
end

end
```

FindtheCenter.m:

```
%% This is a function used to find the center of an enclosed object
%Made by TRevor Craig
%Started 4/27/2016 at 3:37 pm

function [position]=FindtheCenter(x1,y1,option)
if option==1
    %% Mean Technique
    % Mean Method
    centerx=mean(x1);
    centery=mean(y1);
    position=int32([centerx,centery]);
end
%% Filled Centroid Option
if option==2
    FilledImg=zeros((max(x1)-(min(x1))),(max(y1)-min(y1)));
    ConnectPoints=zeros(2*(numel(x1)+1),1);
    for k=1:numel(x1)
        FilledImg(int32(y1(k)),int32(x1(k)))=255;%This can probably be
removed
        ConnectPoints(((2*k)-1))=int32(x1(k));
        ConnectPoints(2*k)=int32(y1(k));
    end
    k=k+1;
    ConnectPoints(((2*k)-1))=int32(x1(1));
    ConnectPoints(2*k)=int32(y1(1));
    ConnectPoints=transpose(ConnectPoints);
    shape='Line';
    LineWidth=1;
    color='red';
    ConnectedImage=insertShape(FilledImg,shape,ConnectPoints,'color',
color,'LineWidth',LineWidth);
    ConnectedImage=im2double(im2bw(ConnectedImage,.1));
    FilledImg=ConnectedImage;
    FilledImg=imfill(FilledImg,'holes');
    % Centroid Method
    s  = regionprops(FilledImg, 'centroid');
    centroids = cat(1, s.Centroid);
    centerx=int32(centroids(1));
    centery=int32(centroids(2));
    position=int32([centerx,centery]);
end
```

DataFilter.m:

```
% My own Data filter program.
%Made by TRevor Craig
%Started 5/17/2016 at 2:04 PM
% This function filters out messy data
function [truefX,truefY]=DataFilter(x1,y1,position,PercentAvg)
stdx=std(double(x1));
stdy=std(double(y1));
NumtoAverage=int32((numel(x1)/PercentAvg));
count=1;
for i=1:numel(x1)
    if (((abs(x1(i)-position(1)))<=(1.5)*stdx)&&((abs(y1(i)-
position(2)))<=(1.5)*stdy))
        tfX(count)=x1(i);
        tfY(count)=y1(i);
        count=count+1;
    end
end

for i=1:numel(tfX)-NumtoAverage
    truefX(i)=int32((sum(tfX(i:i+NumtoAverage)))/NumtoAverage);
    truefY(i)=int32((sum(tfY(i:i+NumtoAverage)))/NumtoAverage);
end
end
```

ShapeRecognFnc.m:

```
% Shape Recognizition Function
% This is based all off Shape Recogn
% This will open an image and see if a program can detect the shape
%Made by TRevor Craig
%Started 5/26/2016 at 11:15 am

function [shape,Value]=ShapeRecognFnc(x1,y1,position)

%% The opener
filename='Images\Image1.JPG';
inifilename='config\Settings.ini';
inputimage=imread(filename);
imsize=size(inputimage);

% %% Importing data if Reading from tables
% if
strcmp('ON',char(strrep(inifile(inifilename,'read',{'DataTableRead','ON
OFF','Data','','ERR'}),'''',''')))
%
TestNUM=cell2mat(inifile(inifilename,'read',{'DataTableRead','CurrentTe
st','TESTNUM','d','ERR'}));
%     if
strcmp('ON',char(strrep(inifile(inifilename,'read',{'DataTableRead','MA
SSTESER','Tester','','ERR'}),'''',''')))
%
TestNUM=cell2mat(inifile(inifilename,'read',{'DataTableRead','MASSTESER
','mintest','d','ERR'}));
%         mintest=TestNUM;
%
maxtest=cell2mat(inifile(inifilename,'read',{'DataTableRead','MASSTESER
','maxtest','d','ERR'}));
%         NSPSE=zeros(maxtest-mintest+1,3);
%         NPercentages=zeros(maxtest-mintest+1,3);
%         NCorners=zeros(maxtest-mintest+1,4);
%         Nmedval=zeros(maxtest-mintest+1,1);
%         NAreanormscale=zeros(maxtest-mintest+1,3);
%         NFUllNormMeanDiff=zeros(maxtest-mintest+1,3);
%         NTotalSlopes=zeros(maxtest-mintest+1,3);
%         WildCorners=zeros(maxtest-mintest+1,3);
%     end
% end
%% Giant FOR LOOP FOR MASS TESTING
% currentTest=1;

% %% Eye Based Test
% if
strcmp('ON',char(strrep(inifile(inifilename,'read',{'GazePointAPI','Use
EyeGaze','Data','','ERR'}),'''',''')))
```

```
%
Calibration=cell2mat(inifile(inifilename,'read',{'GazePointAPI','Calibr
ation','Calibrate','d','ERR'})); %Set Calibration to 1 for test
%
TotalBlinktime=cell2mat(inifile(inifilename,'read',{'GazePointAPI','Cal
ibration','TotalBlinktime','d','ERR'}));%Reccomended setting is to be 8
%     [tx1,ty1]=GazePointApi(Calibration,TotalBlinktime);
%     [position]=FindtheCenter(tx1,ty1,2); %Multiple options for What
center technique is used
%
NumAvG=cell2mat(inifile(inifilename,'read',{'DataFilter','NumbertoAvg',
'Number','d','ERR'}));
%     [x1,y1]=DataFilter(tx1,ty1,position,NumAvG); %Standarddeviation
filter with average sum
%
% end

% for TestNUM=mintest:maxtest

    FillMatrix=zeros(imsize(1),imsize(2));
    %% The gaze selected object with a click in this case
%     if
strcmp('OFF',char(strrep(inifile(inifilename,'read',{'DataTableRead','O
N OFF','Data','','ERR'}),'''',''')))
%         figure;
%         imshow(inputimage);
%         if
strcmp('OFF',char(strrep(inifile(inifilename,'read',{'GazePointAPI','Us
eEyeGaze','Data','','ERR'}),'''',''')))
%             [x,y,button] = ginput(1);
%             position=[x,y];
%         end
%         color='red';
%     end
%     if
strcmp('ON',char(strrep(inifile(inifilename,'read',{'DataTableRead','ON
OFF','Data','','ERR'}),'''',''')))
%         figure;
%         imshow(inputimage);
%
%         [position,x1,y1]=TrialTableRead(TestNUM);
%         x=position(1);
%         y=position(2);
%         color='red';
%     end

    sizes=cell2mat(inifile(inifilename,'read',{'ShapeRecogn','Augmented
Shape','Markersize','d','ERR'}));
    %% Marking the selected object
    if
strcmp('ON',char(strrep(inifile(inifilename,'read',{'ShapeRecogn','Augm
ented Shape','Marker','','ERR'}),'''',''')));
        color='red';
```

```
        MarkedImage = insertMarker(inputimage,position,'color', color,
'size',sizes);
%         imshow(MarkedImage);

LineWidth=cell2mat(inifile(inifilename,'read',{'ShapeRecogn','Augmented
Shape','LineWidth','d','ERR'}));
    else
        MarkedImage=inputimage;
    end

    %% Adding shapes to the image to see better(circle first)
    if
strcmp('ON',char(strrep(inifile(inifilename,'read',{'ShapeRecogn','Augm
ented Shape','Circle','','ERR'}),'''',''')));

LineWidth=cell2mat(inifile(inifilename,'read',{'ShapeRecogn','Augmented
Shape','LineWidth','d','ERR'}));
        shape='circle';

radius=cell2mat(inifile(inifilename,'read',{'ShapeRecogn','Augmented
Shape','radius','d','ERR'}));
        cposition=[position,radius];
        ShapedImage=insertShape(MarkedImage,shape,cposition,'color',
color,'LineWidth',LineWidth);
%         imshow(ShapedImage);
    else
        ShapedImage=MarkedImage;

radius=cell2mat(inifile(inifilename,'read',{'ShapeRecogn','Augmented
Shape','radius','d','ERR'}));
        %Note the radius is also the minimum value that you can draw
the shape
        %so be sure to make a radius that is appropriate for the image
    end


    %% Adding shapes to the image to see better(square second)
    if
strcmp('ON',char(strrep(inifile(inifilename,'read',{'ShapeRecogn','Augm
ented Shape','Square','','ERR'}),'''',''')));
        shape='Rectangle';

radius=cell2mat(inifile(inifilename,'read',{'ShapeRecogn','Augmented
Shape','radius','d','ERR'}));

LineWidth=cell2mat(inifile(inifilename,'read',{'ShapeRecogn','Augmented
Shape','LineWidth','d','ERR'}));
        width=2*radius;
        height=width;
        sposition=[(position(1)-width/2),(position(2)-
width/2),width,height];
        ShapedImage=insertShape(ShapedImage,shape,sposition,'color',
color,'LineWidth',LineWidth);
%         imshow(ShapedImage);
```

```
    end

    %% Adding shapes to the image to see better(triangle third)
    if
strcmp('ON',char(strrep(inifile(inifilename,'read',{'ShapeRecogn','Augm
ented Shape','Triangle','','ERR'}),'''',''')));
        shape='Line';

radius=cell2mat(inifile(inifilename,'read',{'ShapeRecogn','Augmented
Shape','radius','d','ERR'}));

LineWidth=cell2mat(inifile(inifilename,'read',{'ShapeRecogn','Augmented
Shape','LineWidth','d','ERR'}));
        Point1=[(position(1)),(position(2)-radius)];
        Point2=[(position(1)-
int32(radius*(cos(degtorad(30.0))))),(position(2)+int32(radius*sin(degt
orad(30))))];

Point3=[(position(1)+int32(radius*(cos(degtorad(30.0))))),(position(2)+
int32(radius*sin(degtorad(30))))];
        triangleposition=[Point1,Point2,Point3,Point1];

ShapedImage=insertShape(ShapedImage,shape,triangleposition,'color',
color,'LineWidth',LineWidth);
%         imshow(ShapedImage);
    end

    %% Tracking Mouse Around Screen
    % Directions.
    % Left click to start tracking
    % Right click to end tracking
%     if
strcmp('OFF',char(strrep(inifile(inifilename,'read',{'DataTableRead','O
N OFF','Data','','ERR'}),'''','''')))
%         if
strcmp('OFF',char(strrep(inifile(inifilename,'read',{'GazePointAPI','Us
eEyeGaze','Data','','ERR'}),'''','''')))
%             [hand,x1,y1]=freehanddraw();
%         end
%     end


    %% Draw the values on our picture.
    for i=1:numel(x1)
        FillMatrix(int32(y1(i)),int32(x1(i)))=255;
    end
%     figure;
%     WithMark=insertMarker(FillMatrix,position,'color', color,
'size',sizes);
%     imshow(WithMark); % Added this to see better if the shape was in
the correct spot
    %% finding the max and min values for the shape scanner to stop
    bounds=16;
```

```
    distances=[abs(position(1)-max(x1)),abs(position(1)-
min(x1)),abs(position(2)-max(y1)),abs(position(2)-min(y1))];
    stoppingpoint=int32((max(distances))/(bounds));


[CircleResutls,SquareResults,TriangleResults]=ShapeScanner(position,rad
ius,bounds,FillMatrix,stoppingpoint);

    %% Finding the percentages for each point
    [PercentCircle,Cradius]=max(CircleResutls(1:stoppingpoint,2));
    Cradius=CircleResutls(Cradius,1);
    [PercentSquare,Sradius]=max(SquareResults(1:stoppingpoint,2));
    Sradius=SquareResults(Sradius,1);
    [PercentTriangle,Tradius]=max(TriangleResults(1:stoppingpoint,2));
    Tradius=TriangleResults(Tradius,1);
    PercentCircle=PercentCircle/numel(x1);
    PercentSquare=PercentSquare/numel(x1);
    PercentTriangle=PercentTriangle/numel(x1);
    Percentages=[PercentCircle,PercentSquare,PercentTriangle];

    %Cool to see each percentage
    if
strcmp('ON',char(strrep(inifile(inifilename,'read',{'ShapeRecogn','Disp
Results','Percentages','','ERR'}),'''',''')));
        disp(Percentages);
    end
    %To Make a results file
    if
strcmp('ON',char(strrep(inifile(inifilename,'read',{'ShapeRecogn','Crea
te Results File','ResultsFile','','ERR'}),'''',''')));
        ResultsfileName='Results\Results.ini';
        inifile(ResultsfileName,'new');
        ResultsTitle='ShapeRecogn Results';

CwriteKeys={ResultsTitle,'Percentages','Circle',Percentages(1),'plain'}
;

SwriteKeys={ResultsTitle,'Percentages','Square',Percentages(2),'plain'}
;

TwriteKeys={ResultsTitle,'Percentages','Triangles',Percentages(3),'plai
n'};
        inifile(ResultsfileName,'write',CwriteKeys);
        inifile(ResultsfileName,'write',SwriteKeys);
        inifile(ResultsfileName,'write',TwriteKeys);
    end

    %% Corner Detection

[NumofCorners,SuccessCircleCorner,SuccessSquareCorner,SuccessTriangleCo
rner]=CornerDetection(position,bounds,x1,y1,FillMatrix,Cradius,Sradius,
Tradius);
```

```
    if
strcmp('ON',char(strrep(inifile(inifilename,'read',{'ShapeRecogn','Disp
Results','NUMOFCORNERS','','ERR'}),'''',''')));

disp([SuccessCircleCorner,SuccessSquareCorner,SuccessTriangleCorner]);
    end

    if
strcmp('ON',char(strrep(inifile(inifilename,'read',{'ShapeRecogn','Crea
te Results File','ResultsFile','','ERR'}),'''',''')));

CwriteKeys={ResultsTitle,'Corner','Circle',SuccessCircleCorner,'plain'}
;

SwriteKeys={ResultsTitle,'Corner','Square',SuccessSquareCorner,'plain'}
;

TwriteKeys={ResultsTitle,'Corner','Triangles',SuccessTriangleCorner,'pl
ain'};
        inifile(ResultsfileName,'write',CwriteKeys);
        inifile(ResultsfileName,'write',SwriteKeys);
        inifile(ResultsfileName,'write',TwriteKeys);
    end


    %% This is all the Shape detect code.


[Cscale,Sscale,Tscale,medval,Areanormscale,FUllNormMeanDiff,TotalSlopes
]=ShapeDetect(x1,y1,FillMatrix,Cradius,Sradius,Tradius,position);

    %% This code sumarizes all of the other programs and uses "fuzzy
logic" to decide which choice to do.
    %This mode is very accurate over all so want it to be
    %powerful. THis may need to be adjusted to account for the values
in the
    %other function.
    Scaler=cell2mat(inifile(inifilename,'read',{'ShapeRecogn','Weighted
Percentage','Scalar','d','ERR'}));

SPSE=[Scaler*Percentages(1),Scaler*Percentages(2),Scaler*Percentages(3)
];

    %This process is to place the corner results into weights.
    currentTest=1;

    if
SuccessCircleCorner>=cell2mat(inifile(inifilename,'read',{'Corner
Detection','SPSE WEIGHTS','LOWCIRCLE','d','ERR'}))
        if
SuccessCircleCorner>=cell2mat(inifile(inifilename,'read',{'Corner
Detection','SPSE WEIGHTS','HIGHCIRCLE','d','ERR'}))
```

```
SPSE(1)=SPSE(1)+cell2mat(inifile(inifilename,'read',{'Corner
Detection','SPSE WEIGHTS','HWEIGHT','d','ERR'}));

WildCorners(currentTest,1)=cell2mat(inifile(inifilename,'read',{'Corner
Detection','SPSE WEIGHTS','HWEIGHT','d','ERR'}));
        else

SPSE(1)=SPSE(1)+cell2mat(inifile(inifilename,'read',{'Corner
Detection','SPSE WEIGHTS','LWEIGHT','d','ERR'}));

WildCorners(currentTest,1)=cell2mat(inifile(inifilename,'read',{'Corner
Detection','SPSE WEIGHTS','LWEIGHT','d','ERR'}));
        end
    end

    if
SuccessSquareCorner>=cell2mat(inifile(inifilename,'read',{'Corner
Detection','SPSE WEIGHTS','LOWSQUARE','d','ERR'}))
        if
SuccessSquareCorner>=cell2mat(inifile(inifilename,'read',{'Corner
Detection','SPSE WEIGHTS','HIGHSQUARE','d','ERR'}))

SPSE(2)=SPSE(2)+cell2mat(inifile(inifilename,'read',{'Corner
Detection','SPSE WEIGHTS','HWEIGHT','d','ERR'}));

WildCorners(currentTest,2)=cell2mat(inifile(inifilename,'read',{'Corner
Detection','SPSE WEIGHTS','HWEIGHT','d','ERR'}));
        else

SPSE(2)=SPSE(2)+cell2mat(inifile(inifilename,'read',{'Corner
Detection','SPSE WEIGHTS','LWEIGHT','d','ERR'}));

WildCorners(currentTest,2)=cell2mat(inifile(inifilename,'read',{'Corner
Detection','SPSE WEIGHTS','LWEIGHT','d','ERR'}));
        end
    end

    if
SuccessTriangleCorner>=cell2mat(inifile(inifilename,'read',{'Corner
Detection','SPSE WEIGHTS','LOWTRIANGLE','d','ERR'}))
        if
SuccessTriangleCorner>=cell2mat(inifile(inifilename,'read',{'Corner
Detection','SPSE WEIGHTS','HIGHTRIANGLE','d','ERR'}))

SPSE(3)=SPSE(3)+cell2mat(inifile(inifilename,'read',{'Corner
Detection','SPSE WEIGHTS','HWEIGHT','d','ERR'}));

WildCorners(currentTest,3)=cell2mat(inifile(inifilename,'read',{'Corner
Detection','SPSE WEIGHTS','HWEIGHT','d','ERR'}));
        else

SPSE(3)=SPSE(3)+cell2mat(inifile(inifilename,'read',{'Corner
Detection','SPSE WEIGHTS','LWEIGHT','d','ERR'}));
```

```
WildCorners(currentTest,3)=cell2mat(inifile(inifilename,'read',{'Corner
Detection','SPSE WEIGHTS','LWEIGHT','d','ERR'}));
        end
    end

    %The shape algorith results in SPSE format
    SPSE(1)=SPSE(1)+Cscale;
    SPSE(2)=SPSE(2)+Sscale;
    SPSE(3)=SPSE(3)+Tscale;

    if
strcmp('ON',char(strrep(inifile(inifilename,'read',{'ShapeRecogn','Disp
Results','SPSE','','ERR'}),'''',''')));
        disp(SPSE);
    end

    if
strcmp('ON',char(strrep(inifile(inifilename,'read',{'ShapeRecogn','Crea
te Results File','ResultsFile','','ERR'}),'''',''')));
        writeKeys={ResultsTitle,'SPSE','ALL THE RESULTS',SPSE,'plain'};
        inifile(ResultsfileName,'write',writeKeys);
    end

    [Value,Index] = max(SPSE);
    if Value<9 %This might need to be adjusted
        disp('No shape found');
    end

    if Value>=9
        if Index==1
            shape='Circle';
        end
        if Index==2
            shape='Square';
        end
        if Index==3
            shape='Triangle';
        end

        disp(['Shape was found to be a ',shape, ' with
',num2str(Value),' SPSE']);
    end

end
```

ShapeScanner.m:

```
function
[CircleResutls,SquareResults,TriangleResults]=ShapeScanner(position,rad
ius,bounds,FillMatrix,stoppingpoint)
imsize=size(FillMatrix);
CircleResutls=zeros(stoppingpoint,2);
SquareResults=CircleResutls;
TriangleResults=CircleResutls;
    for i=1:stoppingpoint

[HighBounds,LowBounds,DrawnShapeMOD]=FilterMaker('circle',position,radi
us+(bounds*i-bounds),FillMatrix,bounds);

SuccesfulPoints1=SuccessFilter(imsize,HighBounds,LowBounds,DrawnShapeMO
D);
        CircleResutls(i,1:2)=[radius+(bounds*i-
bounds),SuccesfulPoints1];

[HighBounds2,LowBounds2,DrawnShapeMOD2]=FilterMaker('Rectangle',positio
n,radius+(bounds*i-bounds),FillMatrix,bounds);

SuccesfulPoints2=SuccessFilter(imsize,HighBounds2,LowBounds2,DrawnShape
MOD2);
        SquareResults(i,1:2)=[radius+(bounds*i-
bounds),SuccesfulPoints2];

[HighBounds3,LowBounds3,DrawnShapeMOD3]=FilterMaker('triangle',position
,radius+(bounds*i-bounds),FillMatrix,bounds);

SuccesfulPoints3=SuccessFilter(imsize,HighBounds3,LowBounds3,DrawnShape
MOD3);
        TriangleResults(i,1:2)=[radius+(bounds*i-
bounds),SuccesfulPoints3];
    end
end
```

CornerDetection.m:

```
function
[NumofCorners,SuccessCircleCorner,SuccessSquareCorner,SuccessTriangleCo
rner]=CornerDetection(position,bounds,x1,y1,FillMatrix,Cradius,Sradius,
Tradius)
inifilename='config\Settings.ini';
ConnectPoints=zeros(2*(numel(x1)+1),1);
for k=1:numel(x1)
    FillMatrix(int32(y1(k)),int32(x1(k)))=255;
    ConnectPoints(((2*k)-1))=int32(x1(k));
    ConnectPoints(2*k)=int32(y1(k));
end
k=k+1;
ConnectPoints(((2*k)-1))=int32(x1(1));
ConnectPoints(2*k)=int32(y1(1));

ConnectPoints=transpose(ConnectPoints);
position=double(position);

%% Connect all the points
shape='Line';
LineWidth=1;
color='red';
ConnectedImage=insertShape(FillMatrix,shape,ConnectPoints,'color',
color,'LineWidth',LineWidth);
ConnectedImage=im2double(im2bw(ConnectedImage,.1));
FillMatrix=ConnectedImage;

%% Corner results
CornerResults=corner(FillMatrix,'Harris',cell2mat(inifile(inifilename,'
read',{'Corner
Detection','Harris','MaxCorners','d','ERR'})),'QualityLevel',cell2mat(i
nifile(inifilename,'read',{'Corner
Detection','Harris','QualityLevel','d','ERR'})),'SensitivityFactor',cel
l2mat(inifile(inifilename,'read',{'Corner
Detection','Harris','SensitivityFactor','d','ERR'}))); %This should be
twice the amount of corners

%% Find all the corners
%Deconstruct Square Raidus to Find Corners
SCorners=[position(1)-Sradius,position(2)-Sradius;
          position(1)-Sradius,position(2)+Sradius;
          position(1)+Sradius,position(2)-Sradius;
          position(1)+Sradius,position(2)+Sradius];

%Deconstruct Triangle Raidus to Find Corners
TCorners=[(position(1)),(position(2)-Tradius);
         (position(1)-
int32(Tradius*(cos(degtorad(30.0))))),(position(2)+int32(Tradius*sin(de
gtorad(30))));
```

```
(position(1)+int32(Tradius*(cos(degtorad(30.0))))),(position(2)+int32(T
radius*sin(degtorad(30))))];

SuccessCircleCorner=0;
SuccessSquareCorner=0;
SuccessTriangleCorner=0;
NumofCorners=numel(CornerResults(:,1));
for g=1:NumofCorners
    %The circle Part
    temp0=sqrt(((abs(position(1)-
CornerResults(g,1)))^2)+(abs(position(2)-CornerResults(g,2))^2));
    temp0=abs(Cradius-temp0);
    if temp0<bounds
        SuccessCircleCorner=SuccessCircleCorner+1;
    end
    %The Square part
    for z=1:4
        temp1=abs(CornerResults(g,1)-SCorners(z,1));
        temp2=abs(CornerResults(g,2)-SCorners(z,2));
        if((temp1<(bounds))&& (temp2<(bounds)))
                SuccessSquareCorner=SuccessSquareCorner+1;
        end
    end
    %The Triangle Part
    for y=1:3
        temp1=abs(CornerResults(g,1)-TCorners(y,1));
        temp2=abs(CornerResults(g,2)-TCorners(y,2));
        if((temp1<(bounds))&& (temp2<(bounds)))
                SuccessTriangleCorner=SuccessTriangleCorner+1;
        end
    end
end
end
```

ShapeDetect.m:

```
function
[CTransfer,STransfer,TTransfer,medval,Areanormscale,FUllNormMeanDiff,To
talSlopes]=ShapeDetect(x1,y1,FillMatrix,Cradius,Sradius,Tradius,positio
n)
SPSE=[0,0,0];
inifilename='config\Settings.ini';
UseDataSet=char(strrep(inifile(inifilename,'read',{'ShapeDetect','DataS
et Options','UseDataSet','','ERR'}),'''',''));

%% OLD TEST STUFF BUT HELPFUL to make it more stand alone
ConnectPoints=zeros(2*(numel(x1)+1),1);

for k=1:numel(x1)
    FillMatrix(int32(y1(k)),int32(x1(k)))=255;
    ConnectPoints(((2*k)-1))=int32(x1(k));
    ConnectPoints(2*k)=int32(y1(k));
end
k=k+1;
ConnectPoints(((2*k)-1))=int32(x1(1));
ConnectPoints(2*k)=int32(y1(1));

ConnectPoints=transpose(ConnectPoints);

%Connect all the points
shape='Line';
LineWidth=1;
color='red';
ConnectedImage=insertShape(FillMatrix,shape,ConnectPoints,'color',
color,'LineWidth',LineWidth);
ConnectedImage=im2double(im2bw(ConnectedImage,.1));
FillMatrix=ConnectedImage;

%% previous test
ImageShape=imfill(FillMatrix,'holes');

BW = im2bw(ImageShape, .1);
[H,~,~] = hough(BW,'RhoResolution',0.5,'ThetaResolution',0.5);

data=zeros(max(max(H)),1);
for cnt = 1:max(max(H))
    data(cnt) = sum(sum(H == cnt));
end

medval = median(data);

if (medval>=0)&&(medval<170)
    shape='TRIANGLE';
```

```
SPSE(3)=cell2mat(inifile(inifilename,'read',{'ShapeDetect','MEDVAL','BE
STWEIGHT','d','ERR'}));

SPSE(2)=cell2mat(inifile(inifilename,'read',{'ShapeDetect','MEDVAL','OT
HERWEIGHT','d','ERR'}));

SPSE(1)=cell2mat(inifile(inifilename,'read',{'ShapeDetect','MEDVAL','OT
HERWEIGHT','d','ERR'}));
end

if (medval>=170)&&(medval<680)
    shape='CIRCLE';

SPSE(3)=cell2mat(inifile(inifilename,'read',{'ShapeDetect','MEDVAL','OT
HERWEIGHT','d','ERR'}));

SPSE(2)=cell2mat(inifile(inifilename,'read',{'ShapeDetect','MEDVAL','OT
HERWEIGHT','d','ERR'}));

SPSE(1)=cell2mat(inifile(inifilename,'read',{'ShapeDetect','MEDVAL','BE
STWEIGHT','d','ERR'}));
end

if (medval>=680)
    shape='SQUARE';

SPSE(3)=cell2mat(inifile(inifilename,'read',{'ShapeDetect','MEDVAL','OT
HERWEIGHT','d','ERR'}));

SPSE(2)=cell2mat(inifile(inifilename,'read',{'ShapeDetect','MEDVAL','BE
STWEIGHT','d','ERR'}));

SPSE(1)=cell2mat(inifile(inifilename,'read',{'ShapeDetect','MEDVAL','OT
HERWEIGHT','d','ERR'}));
end
if
strcmp('ON',char(strrep(inifile(inifilename,'read',{'ShapeDetect','Disp
Results','medval','','ERR'}),'''','')))
    disp(['Shape was found to be a ',shape, ' with ',num2str(medval),'
MedValue!']);
end

if
strcmp('ON',char(strrep(inifile(inifilename,'read',{'ShapeRecogn','Crea
te Results File','ResultsFile','','ERR'}),'''','')));
    ResultsTitle='ShapeRecogn Results';
    ResultsfileName='Results\Results.ini';
    WriteKeys={ResultsTitle,'MEDVAL','Number',medval,'plain'};
    inifile(ResultsfileName,'write',WriteKeys);
end
```

```
%% Try plotting a perfect result and force fitting the result to the
other one
%this is in the hopes of making perfect data to compare the two plots
%without guessing.
if strcmp(UseDataSet,'ON')
    [Cdata,Sdata,Tdata]=HoughAssist(Cradius,Sradius,Tradius);
end

if strcmp(UseDataSet,'OFF')
    imsize=size(FillMatrix);
    Cplot=zeros(imsize(1),imsize(2),1);
    Splot=zeros(imsize(1),imsize(2),1);
    Tplot=zeros(imsize(1),imsize(2),1);

    %Creating Circle First
    shape='circle';
    cposition=[position,Cradius];
    LineWidth=1;
    Cplot=insertShape(Cplot,shape,cposition,'color',
color,'LineWidth',LineWidth);
    Cplot = im2bw(Cplot, .1);
    Cplot=imfill(Cplot,'holes');

    % Adding shapes to the image (square second)
    shape='Rectangle';
    width=2*Sradius;
    height=width;
    sposition=[(position(1)-width/2),(position(2)-
width/2),width,height];
    LineWidth=1;
    Splot=insertShape(Splot,shape,sposition,'color',
color,'LineWidth',LineWidth);
    Splot = im2bw(Splot, .1);
    Splot=imfill(Splot,'holes');

    % Adding shapes to the image to see better(triangle third)
    shape='Line';
    Point1=[(position(1)),(position(2)-Tradius)];
    Point2=[(position(1)-
int32(Tradius*(cos(degtorad(30.0))))),(position(2)+int32(Tradius*sin(de
gtorad(30))))];

Point3=[(position(1)+int32(Tradius*(cos(degtorad(30.0))))),(position(2)
+int32(Tradius*sin(degtorad(30))))];
    triangleposition=[Point1,Point2,Point3,Point1];
    LineWidth=1;
    Tplot=insertShape(Tplot,shape,triangleposition,'color',
color,'LineWidth',LineWidth);
    Tplot = im2bw(Tplot, .1);
    Tplot=imfill(Tplot,'holes');

    %Do all the hough calculations
    [CH,~,~] = hough(Cplot,'RhoResolution',0.5,'ThetaResolution',0.5);
```

```
    [SH,~,~] = hough(Splot,'RhoResolution',0.5,'ThetaResolution',0.5);
    [TH,~,~] = hough(Tplot,'RhoResolution',0.5,'ThetaResolution',0.5);

    for cnt = 1:max(max(CH))
        Cdata(cnt) = sum(sum(CH == cnt));
    end
    for cnt = 1:max(max(SH))
        Sdata(cnt) = sum(sum(SH == cnt));
    end
    for cnt = 1:max(max(TH))
        Tdata(cnt) = sum(sum(TH == cnt));
    end
end
%% Plotting tools
%     plot(data,'.');
%     hold on;
%     plot(Tdata,'-');
%     pbaspect([1 .55 1]);
%     xlabel('Hough Matrix Intensity'), ylabel('Counts');
%     title('Idealized Triangle Shape vs Actual Input');
%     legend('User Input','Idealized Shape');
%     line([Tradius,Tradius],ylim);
%     ootherguess=int32(Tradius/tan(deg2rad(60)));
%     line([ootherguess,ootherguess],ylim);
%     halfguess=ootherguess+((Tradius-ootherguess)/3);
%     line([halfguess,halfguess],ylim);

%This is done to find the area under the curve and see which one is the
%closest fit for the program.
SampCUM = cumtrapz(data);
SampSUM=SampCUM(numel(data));

CCUM = cumtrapz(Cdata);
CSUM=CCUM(numel(Cdata));

SCUM = cumtrapz(Sdata);
SSUM=SCUM(numel(Sdata));

TCUM =cumtrapz(Tdata);
TSUM=TCUM(numel(Tdata));

Cdiff=abs(SampSUM-CSUM);
Sdiff=abs(SampSUM-SSUM);
Tdiff=abs(SampSUM-TSUM);
Differnces=[Cdiff,Sdiff,Tdiff];

normDifferences=Differnces/norm(Differnces);
AreaScale=cell2mat(inifile(inifilename,'read',{'ShapeDetect','Differnt
Area','Scale','d','ERR'}));
Areanormscale=[AreaScale*(1-normDifferences(1)),AreaScale*(1-
normDifferences(2)),AreaScale*(1-normDifferences(3))];
SPSE(1)=SPSE(1)+AreaScale*(1-normDifferences(1));
SPSE(2)=SPSE(2)+AreaScale*(1-normDifferences(2));
```

```
SPSE(3)=SPSE(3)+AreaScale*(1-normDiffernces(3));

if
strcmp('ON',char(strrep(inifile(inifilename,'read',{'ShapeDetect','Disp
Results','diffarea','','ERR'}),'''',''')))
    disp(1-normDiffernces);
end

if
strcmp('ON',char(strrep(inifile(inifilename,'read',{'ShapeRecogn','Crea
te Results File','ResultsFile','','ERR'}),'''',''')));
    CwriteKeys={ResultsTitle,'Area Differnces','Circle',AreaScale*(1-
normDiffernces(1)),'plain'};
    SwriteKeys={ResultsTitle,'Area Differnces','Square',AreaScale*(1-
normDiffernces(2)),'plain'};
    TwriteKeys={ResultsTitle,'Area
Differnces','Triangles',AreaScale*(1-normDiffernces(3)),'plain'};
    inifile(ResultsfileName,'write',CwriteKeys);
    inifile(ResultsfileName,'write',SwriteKeys);
    inifile(ResultsfileName,'write',TwriteKeys);
end

%Now to check for the average value and variance from expected for the
first part to see
%if information there can help.
%how much of the graph to look at?
howfar=cell2mat(inifile(inifilename,'read',{'ShapeDetect','MeanValues',
'howfar','d','ERR'}));

SampMEAN=mean(data(1:int32((howfar)*numel(data))));
CMEAN=mean(Cdata(1:int32((howfar)*numel(Cdata))));
SMEAN=mean(Sdata(1:int32((howfar)*numel(Sdata))));
TMEAN=mean(Tdata(1:int32((howfar)*numel(Tdata))));

MeanDiffernces=[abs(SampMEAN-CMEAN),abs(SampMEAN-SMEAN),abs(SampMEAN-
TMEAN)];

normMeanDiffernces=MeanDiffernces/norm(MeanDiffernces);
Scale=cell2mat(inifile(inifilename,'read',{'ShapeDetect','MeanValues','
Scale','d','ERR'}));
FUllNormMeanDiff=[Scale*(1-normMeanDiffernces(1)),Scale*(1-
normMeanDiffernces(2)),Scale*(1-normMeanDiffernces(3))];
SPSE(1)=SPSE(1)+Scale*(1-normMeanDiffernces(1));
SPSE(2)=SPSE(2)+Scale*(1-normMeanDiffernces(2));
SPSE(3)=SPSE(3)+Scale*(1-normMeanDiffernces(3));

if
strcmp('ON',char(strrep(inifile(inifilename,'read',{'ShapeDetect','Disp
Results','diffmeanvals','','ERR'}),'''',''')))
    disp(1-normMeanDiffernces);
end
```

```
if
strcmp('ON',char(strrep(inifile(inifilename,'read',{'ShapeRecogn','Crea
te Results File','ResultsFile','','ERR'}),'''',''')));
    CwriteKeys={ResultsTitle,'Mean Differnces','Circle',Scale*(1-
normMeanDiffernces(1)),'plain'};
    SwriteKeys={ResultsTitle,'Mean Differnces','Square',Scale*(1-
normMeanDiffernces(2)),'plain'};
    TwriteKeys={ResultsTitle,'Mean Differnces','Triangles',Scale*(1-
normMeanDiffernces(3)),'plain'};
    inifile(ResultsfileName,'write',CwriteKeys);
    inifile(ResultsfileName,'write',SwriteKeys);
    inifile(ResultsfileName,'write',TwriteKeys);
end


%Checking the first few points and the last few points near the slope
line
% Idea is to check for a consistent curve. Circle slopes upwards,
square
%is constant. and triangle near constant but higher value.This will
provide
%a sloped value that can be compared to see if values increase this
will
%not be effective for triangle or square. Also expect that circles may
have
%lower values as fewer points are as high
% MIGHT want to check if values go up or down for the slope

avgpoints=cell2mat(inifile(inifilename,'read',{'ShapeDetect','Slope','a
vgpoints','d','ERR'})); %how many point to average

CircleSlopePoint=Cradius;
SquareSlopePoint=Sradius;
TriangleSlopePoint=int32((Tradius/tan(deg2rad(60)))+((Tradius-
(Tradius/tan(deg2rad(60))))/3));

dataP1=mean(data(1:avgpoints));
CdataP2=mean(data(CircleSlopePoint-avgpoints:CircleSlopePoint));
SdataP2=mean(data(SquareSlopePoint-avgpoints:SquareSlopePoint));
TdataP2=mean(data(TriangleSlopePoint-avgpoints:TriangleSlopePoint));
slopeC=abs(dataP1-CdataP2);
slopeS=abs(dataP1-SdataP2);
slopeT=abs(dataP1-TdataP2);

CdataP1=mean(Cdata(1:avgpoints));
Cslope=abs(CdataP1-CdataP2);

SdataP1=mean(Sdata(1:avgpoints));
Sslope=abs(SdataP1-SdataP2);

TdataP1=mean(Tdata(1:avgpoints));
Tslope=abs(TdataP1-TdataP2);
```

```
slopes=[abs(slopeC-Cslope),abs(slopeS-Sslope),abs(slopeT-Tslope)];

normslopes=slopes/norm(slopes);
scale=cell2mat(inifile(inifilename,'read',{'ShapeDetect','Slope','scale
','d','ERR'}));
TotalSlopes=[scale*(1-normslopes(1)),scale*(1-normslopes(2)),scale*(1-
normslopes(3))];
SPSE(1)=SPSE(1)+scale*(1-normslopes(1));
SPSE(2)=SPSE(2)+scale*(1-normslopes(2));
SPSE(3)=SPSE(3)+scale*(1-normslopes(3));
if
strcmp('ON',char(strrep(inifile(inifilename,'read',{'ShapeDetect','Disp
Results','diffslopes','','ERR'}),'''',''')))
    disp(1-normslopes);
end

%% ResultsFile
if
strcmp('ON',char(strrep(inifile(inifilename,'read',{'ShapeRecogn','Crea
te Results File','ResultsFile','','ERR'}),'''',''')));
    CwriteKeys={ResultsTitle,'Slope','Circle',scale*(1-
normslopes(1)),'plain'};
    SwriteKeys={ResultsTitle,'Slope','Square',scale*(1-
normslopes(2)),'plain'};
    TwriteKeys={ResultsTitle,'Slope','Triangles',scale*(1-
normslopes(3)),'plain'};
    inifile(ResultsfileName,'write',CwriteKeys);
    inifile(ResultsfileName,'write',SwriteKeys);
    inifile(ResultsfileName,'write',TwriteKeys);
end

%% Transfer the data out of the program/function :D
CTransfer=SPSE(1);
STransfer=SPSE(2);
TTransfer=SPSE(3);
end
```

HoughAssist.m:

```
function [Cdata,Sdata,Tdata]=HoughAssist(Cradius,Sradius,Tradius)
%This Helps get the values from the tables.
%Hugh Table assist. To go in shape detect
Title='Hough Data';
inifilename='config\Settings.ini';
topradius=cell2mat(inifile(inifilename,'read',{'HoughAssist','Stationar
yvalues','topradius','d','ERR'}));
%This is the largest number aviable to the user Don't change
splits=cell2mat(inifile(inifilename,'read',{'HoughAssist','Stationaryva
lues','splits','d','ERR'}));
numofsplits=1;

basefileName='SmallHoughData\HoughData_';
fileName=strcat(basefileName,num2str(1));
fileName=strcat(fileName,'_');
fileName=strcat(fileName,num2str(splits*numofsplits));
fileName=strcat(fileName,'.ini');

if max([Cradius,Sradius,Tradius])<=topradius
    for i=1:max([Cradius,Sradius,Tradius])
        if i==Cradius
            readKeys = {Title,'Circle',num2str(Cradius),'d','ERR'};
            Cdata = cell2mat(inifile(fileName,'read',readKeys));
        end
        if i==Sradius
            readKeys = {Title,'Square',num2str(Sradius),'d','ERR'};
            Sdata = cell2mat(inifile(fileName,'read',readKeys));
        end
        if i==Tradius
            readKeys = {Title,'Triangle',num2str(Tradius),'d','ERR'};
            Tdata = cell2mat(inifile(fileName,'read',readKeys));
        end

        if numofsplits*splits==i
            if i~= topradius
                numofsplits=numofsplits+1;
                fileName=strcat(basefileName,num2str(i+1));
                fileName=strcat(fileName,'_');
                fileName=strcat(fileName,num2str(splits*numofsplits));
                fileName=strcat(fileName,'.ini');
            end
        end
    end

end
if max([Cradius,Sradius,Tradius])>topradius
    disp('Data set out of bounds');
end
end
```

DISPLAYXY.m:

```
% Display image from x and y
%Made by TRevor Craig
%Started 5/31/2016 at 10:57 PM
% This functions displays image given X and Y input

function FilledImg=DISPXY(truefX,truefY)

FilledImg=zeros((max(truefX)-(min(truefX))),(max(truefY)-min(truefY)));
ConnectPoints=zeros(2*(numel(truefX)+1),1);
for k=1:numel(truefX)
    FilledImg(int32(truefY(k)),int32(truefX(k)))=255;%This can probably
be removed
    ConnectPoints(((2*k)-1))=int32(truefX(k));
    ConnectPoints(2*k)=int32(truefY(k));
end
k=k+1;
ConnectPoints(((2*k)-1))=int32(truefX(1));
ConnectPoints(2*k)=int32(truefY(1));
ConnectPoints=transpose(ConnectPoints);
shape='Line';
LineWidth=1;
color='red';
ConnectedImage=insertShape(FilledImg,shape,ConnectPoints,'color',
color,'LineWidth',LineWidth);
ConnectedImage=im2double(im2bw(ConnectedImage,.1));
FilledImg=ConnectedImage;
FilledImg=imfill(FilledImg,'holes');
end
```

ProgramLauncer.m:

```
% Launch The external Sofware Gazepoint Control
%Made by TRevor Craig
%Started 5/17/2016 at 2:04 PM
% This function launches the program needed to collect eye data.
function ProgramLaunher()
%This needs to be specfied for eaxh case program path
% fileexe_path = which ('Gazepoint.exe')
% system_command_string = [fileexe_path, ' &'];
% status = system (system_command_string)

system('C:\Program Files (x86)\Gazepoint\Gazepoint\bin64\Gazepoint.exe
&');
dos('taskkill /IM cmd.exe');
clc;
end
```

SaveToFiles.m:

```
% Save Results to Fine
%Made by TRevor Craig
%Started 8/2/2016 at 2:33 PM
% This functions gathers and then displays the results.
function SaveToFiles(IDNumber,recordX,recordY,SPSE,testNum)
FileDirectory='UserResults\';
Excelfile=strcat(FileDirectory,IDNumber,'\',num2str(SPSE),'_',num2str(t
estNum),'.csv');
FileName=strcat(FileDirectory,IDNumber);
A = exist(FileName,'file');% Return value should Be 7
if A==0
    mkdir(FileDirectory,IDNumber)
end
littletable=table(transpose(recordX),transpose(recordY));
writetable(littletable,Excelfile,'WriteVariableNames',false);
end
```

RandomName.m:

```
%Random Characters String
%Made by TRevor Craig
%Started 10/6/2016 at 1:13 AM

clear all; %Clear the screen for the user
close all; %Closses all the windows
echo off; %Doesn't display code the user doesn't need
clc; %Clears the command window For the User
clear vars; %Cleans up any previous data

%% General Set up
Letter='A':'Z';
Number=0:1:9;
NameLength=10;
Name='';
for i=1:NameLength
    Option = randi([1 2],1,1);

    if Option==1
       RandomChar=int2str(Number(randi([1 10],1,1)));
    end
    if Option==2
        RandomChar=char(Letter(randi([1 26],1,1)));
    end

    Name=strcat(Name,RandomChar);
end
disp('Your Name IS:')
disp(Name);
```

RandomTestOrder.m:

```
%Random Test Order
%Made by TRevor Craig
%Started 10/6/2016 at 1:30 AM

clear all; %Clear the screen for the user
close all; %Closses all the windows
echo off; %Doesn't display code the user doesn't need
clc; %Clears the command window For the User
clear vars; %Cleans up any previous data

%% Finding Random orders
TestLength=30;
Letter=['C','S','T'];
CircleCount=0;
SquareCount=0;
TriangleCount=0;
TestOrder='';
i=0;
while(TestLength~=numel(TestOrder))
    RandomChar=char(Letter(randi([1 3],1,1)));
    if RandomChar=='C';
        if CircleCount~=10
            TestOrder=strcat(TestOrder,RandomChar);
            CircleCount=CircleCount+1;
        end
    end
    if RandomChar=='S';
        if SquareCount~=10
            TestOrder=strcat(TestOrder,RandomChar);
            SquareCount=SquareCount+1;
        end

    end
    if RandomChar=='T';
        if TriangleCount~=10
            TestOrder=strcat(TestOrder,RandomChar);
            TriangleCount=TriangleCount+1;
        end
    end
    i=i+1;
    disp(['Number of Iterations:',int2str(i)]);
end
clc;
disp(['Number of Iterations:',int2str(i)]);
disp('The order of the test:');
disp(TestOrder);
```

APPENDIX 4:

This section has all the code needed for the Raspberry Pi video capture.

Camera Control Program:

```
# Point-and-shoot camera for Raspberry Pi w/camera and Adafruit PiTFT.
# This must run as root (sudo python cam.py) due to framebuffer, etc.
#

# This can also work with the Model A board and/or the Pi NoIR camera.
# Made by Trevor Craig
# Adapted from Phil Burgess / Paint Your Dragon for Adafruit Industries.

import atexit
import cPickle as pickle
import errno
import fnmatch
import io
import os
import os.path
import picamera
import pygame
import stat
import threading
import time
import yuv2rgb
```

```
import RPi.GPIO as GPIO
from pygame.locals import *
from subprocess import call
import subprocess
import pyfirmata

#Setting up the arduino to do its work
board=pyfirmata.Arduino('/dev/ttyACM0')
print "Setting up the connection"

APins=1 #Pin that we are reading

it=pyfirmata.util.Iterator(board)
it.start()
# Start reporting of pin 1
board.analog[1].enable_reporting()

#Setting up the GPIO Buttons

ExitButton=27
PICBUTTON=17
LowButton=23
HighButton=22
```

```
GPIO.setmode(GPIO.BCM)

GPIO.setup(ExitButton,GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(PICBUTTON,GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(LowButton,GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(HighButton,GPIO.IN, pull_up_down=GPIO.PUD_UP)

# UI classes ------------------------------------------------------------

# Small resistive touchscreen is best suited to simple tap interactions.
# Importing a big widget library seemed a bit overkill.  Instead, a couple
# of rudimentary classes are sufficient for the UI elements:

# Icon is a very simple bitmap class, just associates a name and a pygame
# image (PNG loaded from icons directory) for each.
# There isn't a globally-declared fixed list of Icons.  Instead, the list
# is populated at runtime from the contents of the 'icons' directory.

class Icon:

        def __init__(self, name):
          self.name = name
          try:
            self.bitmap = pygame.image.load(iconPath + '/' + name + '.png')
```

```
            except:
              pass


# Button is a simple tappable screen region.  Each has:
#  - bounding rect ((X,Y,W,H) in pixels)
#  - optional background color and/or Icon (or None), always centered
#  - optional foreground Icon, always centered
#  - optional single callback function
#  - optional single value passed to callback
# Occasionally Buttons are used as a convenience for positioning Icons
# but the taps are ignored.  Stacking order is important; when Buttons
# overlap, lowest/first Button in list takes precedence when processing
# input, and highest/last Button is drawn atop prior Button(s).  This is
# used, for example, to center an Icon by creating a passive Button the
# width of the full screen, but with other buttons left or right that
# may take input precedence (e.g. the Effect labels & buttons).
# After Icons are loaded at runtime, a pass is made through the global
# buttons[] list to assign the Icon objects (from names) to each Button.


class Button:


            def __init__(self, rect, **kwargs):
              self.rect     = rect # Bounds
              self.color    = None # Background fill color, if any
```

```
  self.iconBg   = None # Background Icon (atop color fill)
  self.iconFg   = None # Foreground Icon (atop background)
  self.bg       = None # Background Icon name
  self.fg       = None # Foreground Icon name
  self.callback = None # Callback function
  self.value    = None # Value passed to callback
  for key, value in kwargs.iteritems():
    if   key == 'color': self.color    = value
    elif key == 'bg'   : self.bg       = value
    elif key == 'fg'   : self.fg       = value
    elif key == 'cb'   : self.callback = value
    elif key == 'value': self.value    = value


def selected(self, pos):
  x1 = self.rect[0]
  y1 = self.rect[1]
  x2 = x1 + self.rect[2] - 1
  y2 = y1 + self.rect[3] - 1
  if ((pos[0] >= x1) and (pos[0] <= x2) and
      (pos[1] >= y1) and (pos[1] <= y2)):
    if self.callback:
      if self.value is None: self.callback()
      else:                  self.callback(self.value)
    return True
```

```
 return False


def draw(self, screen):
 if self.color:
  screen.fill(self.color, self.rect)
 if self.iconBg:
  screen.blit(self.iconBg.bitmap,
   (self.rect[0]+(self.rect[2]-self.iconBg.bitmap.get_width())/2,
    self.rect[1]+(self.rect[3]-self.iconBg.bitmap.get_height())/2))
 if self.iconFg:
  screen.blit(self.iconFg.bitmap,
   (self.rect[0]+(self.rect[2]-self.iconFg.bitmap.get_width())/2,
    self.rect[1]+(self.rect[3]-self.iconFg.bitmap.get_height())/2))


def setBg(self, name):
 if name is None:
  self.iconBg = None
 else:
  for i in icons:
   if name == i.name:
    self.iconBg = i
    break
```

```
# UI callbacks ------------------------------------------------------------
# These are defined before globals because they're referenced by items in
# the global buttons[] list.

def isoCallback(n): # Pass 1 (next ISO) or -1 (prev ISO)
    global isoMode
    setIsoMode((isoMode + n) % len(isoData))

def settingCallback(n): # Pass 1 (next setting) or -1 (prev setting)
    global screenMode
    screenMode += n
    if screenMode < 4:          screenMode = len(buttons) - 1
    elif screenMode >= len(buttons): screenMode = 4

def fxCallback(n): # Pass 1 (next effect) or -1 (prev effect)
    global fxMode
    setFxMode((fxMode + n) % len(fxData))

def quitCallback(): # Quit confirmation button
    saveSettings()
    raise SystemExit

def viewCallback(n): # Viewfinder buttons
    global loadIdx, scaled, screenMode, screenModePrior, settingMode, storeMode
```

```
            if n is 0:   # Gear icon (settings)
              screenMode = settingMode # Switch to last settings mode
            elif n is 1: # Play icon (image playback)
              if scaled: # Last photo is already memory-resident
                loadIdx        = saveIdx
                screenMode     =  0 # Image playback
                screenModePrior = -1 # Force screen refresh
              else:      # Load image
                r = imgRange(pathData[storeMode])
                if r: showImage(r[1]) # Show last image in directory
                else: screenMode = 2  # No images
            else: # Rest of screen = shutter
              takePicture()

def doneCallback(): # Exit settings
            global screenMode, settingMode
            if screenMode > 3:
              settingMode = screenMode
              saveSettings()
            screenMode = 3 # Switch back to viewfinder mode

def imageCallback(n): # Pass 1 (next image), -1 (prev image) or 0 (delete)
            global screenMode
```

```
        if n is 0:
          screenMode = 1 # Delete confirmation
        else:
          showNextImage(n)

def deleteCallback(n): # Delete confirmation
        global loadIdx, scaled, screenMode, storeMode
        screenMode      =  0
        screenModePrior = -1
        if n is True:
          os.remove(pathData[storeMode] + '/IMG_' + '%04d' % loadIdx + '.JPG')
          if(imgRange(pathData[storeMode])):
            screen.fill(0)
            pygame.display.update()
            showNextImage(-1)
          else: # Last image deleteted; go to 'no images' mode
            screenMode = 2
            scaled     = None
            loadIdx    = -1

def storeModeCallback(n): # Radio buttons on storage settings screen
        global storeMode
        buttons[4][storeMode + 3].setBg('radio3-0')
        storeMode = n
```

```
        buttons[4][storeMode + 3].setBg('radio3-1')


def sizeModeCallback(n): # Radio buttons on size settings screen
        global sizeMode
        buttons[5][sizeMode + 3].setBg('radio3-0')
        sizeMode = n
        buttons[5][sizeMode + 3].setBg('radio3-1')
        camera.resolution = sizeData[sizeMode][1]


def SetLowPot(): #This calibrates the pot to be at the zero pressure mark
    global LowPot
    LowPot=board.analog[1].read()


def SetHighPot(): #This calibrates the pot at the high end of pressure 30
    global HighPot
    HighPot=board.analog[1].read()


def ScalePotValues(): #Takes in potentiometer values and outputs pressures
    global PressureReading
    #LowPot=0
    #HighPot=1
    LowPressure=0
    HighPressure=30
    myreading=board.analog[1].read()
```

```
    PressureReading=(myreading-LowPot)*(HighPressure-LowPressure)/(HighPot-
LowPot)+LowPressure


def VideoNameSetter():
    global VideoName
    global VideoNumber
    WorkingDir='/media/EyeUSB/EyeVideos/'
    exists=1
    while (exists==1):
        exists=0
        if os.path.exists('/media/EyeUSB/EyeVideos/Myvideo'+str(VideoNumber)+'.h264'):
            exists=1
            print 'Finding File'
            VideoNumber=VideoNumber+1

    VideoName='/media/EyeUSB/EyeVideos/Myvideo'+str(VideoNumber)+'.h264'


def ConvertVideo():
    camera.annotate_text=str('SAVING')
    NewVideoName='/media/EyeUSB/EyeVideos/Myvideo'+str(VideoNumber)+'.mp4'
    ProcessName=str('MP4Box -add '+str(VideoName)+' '+str(NewVideoName))
    process=subprocess.Popen(ProcessName, shell=True, stdout=subprocess.PIPE)
    process.wait()


# Global stuff ------------------------------------------------------------
```

```
screenMode     =  3      # Current screen mode; default = viewfinder
screenModePrior = -1     # Prior screen mode (for detecting changes)
settingMode    =  4      # Last-used settings mode (default = storage)
storeMode      =  0      # Storage mode; default = Photos folder
storeModePrior  = -1     # Prior storage mode (for detecting changes)
sizeMode       =  0      # Image size; default = Large
fxMode         =  0      # Image effect; default = Normal
isoMode        =  0      # ISO settingl default = Auto
iconPath       = 'icons' # Subdirectory containing UI bitmaps (PNG format)
saveIdx        = -1      # Image index for saving (-1 = none set yet)
loadIdx        = -1      # Image index for loading
scaled         = None    # pygame Surface w/last-loaded image
PressureReading = 0      # This is the converted pressure reading from 0 and 1
LowPot         = 0       # This is the lower potetiometer value
HighPot        = 1       # This is the high potetiometer value
VideoNumber    = 1       #Starting VIdeo Number

sizeData = [ # Camera parameters for different size settings
 # Full res      Viewfinder  Crop window
 [(2592, 1944), (320, 240), (0.0   , 0.0   , 1.0   , 1.0   )], # Large
 [(1920, 1080), (320, 180), (0.1296, 0.2222, 0.7408, 0.5556)], # Med
 [(1440, 1080), (320, 240), (0.2222, 0.2222, 0.5556, 0.5556)]] # Small
```

```
isoData = [ # Values for ISO settings [ISO value, indicator X position]
 [  0,  27], [100,  64], [200,  97], [320, 137],
 [400, 164], [500, 197], [640, 244], [800, 297]]

# A fixed list of image effects is used (rather than polling
# camera.IMAGE_EFFECTS) because the latter contains a few elements
# that aren't valid (at least in video_port mode) -- e.g. blackboard,
# whiteboard, posterize (but posterise, British spelling, is OK).
# Others have no visible effect (or might require setting add'l
# camera parameters for which there's no GUI yet) -- e.g. saturation,
# colorbalance, colorpoint.
fxData = [
  'none', 'sketch', 'gpen', 'pastel', 'watercolor', 'oilpaint', 'hatch',
  'negative', 'colorswap', 'posterise', 'denoise', 'blur', 'film',
  'washedout', 'emboss', 'cartoon', 'solarize' ]

pathData = [
  #'/home/pi/Photos',     # Path for storeMode = 0 (Photos folder)
  '/media/EyeUSB/EyePics',   #Place to store the files to USB
  '/boot/DCIM/CANON999'] # Path for storeMode = 1 (Boot partition)

icons = [] # This list gets populated at startup

# buttons[] is a list of lists; each top-level list element corresponds
```

```
# to one screen mode (e.g. viewfinder, image playback, storage settings),
# and each element within those lists corresponds to one UI button.
# There's a little bit of repetition (e.g. prev/next buttons are
# declared for each settings screen, rather than a single reusable
# set); trying to reuse those few elements just made for an ugly
# tangle of code elsewhere.

buttons = [
 # Screen mode 0 is photo playback
 [Button((  0,188,320, 52), bg='done' , cb=doneCallback),
  Button((  0,  0, 80, 52), bg='prev' , cb=imageCallback, value=-1),
  Button((240,  0, 80, 52), bg='next' , cb=imageCallback, value= 1),
  Button(( 88, 70,157,102)), # 'Working' label (when enabled)
  Button((148,129, 22, 22)), # Spinner (when enabled)
  Button((121,  0, 78, 52), bg='trash', cb=imageCallback, value= 0)],

 # Screen mode 1 is delete confirmation
 [Button((  0,35,320, 33), bg='delete'),
  Button(( 32,86,120,100), bg='yn', fg='yes',
   cb=deleteCallback, value=True),
  Button((168,86,120,100), bg='yn', fg='no',
   cb=deleteCallback, value=False)],

 # Screen mode 2 is 'No Images'
```

```
[Button((0,  0,320,240), cb=doneCallback), # Full screen = button
 Button((0,188,320, 52), bg='done'),       # Fake 'Done' button
 Button((0, 53,320, 80), bg='empty')],     # 'Empty' message

# Screen mode 3 is viewfinder / snapshot
#[Button((  0,188,156, 52), bg='gear', cb=viewCallback, value=0),
[Button((  0,240-128,128, 128), bg='Gear2', cb=viewCallback, value=0),
 Button((164,188,156, 52), bg='play', cb=viewCallback, value=1),
 Button((  0,  0,320,240)          , cb=viewCallback, value=2),
 Button(( 88, 51,157,102)),  # 'Working' label (when enabled)
 Button((148, 110,22, 22))], # Spinner (when enabled)

# Remaining screens are settings modes

# Screen mode 4 is storage settings
[Button((  0,188,320, 52), bg='done', cb=doneCallback),
 Button((  0,  0, 80, 52), bg='prev', cb=settingCallback, value=-1),
 Button((240,  0, 80, 52), bg='next', cb=settingCallback, value= 1),
 Button((  2, 60,100,120), bg='radio3-1', fg='store-folder',
  cb=storeModeCallback, value=0),
 Button((110, 60,100,120), bg='radio3-0', fg='store-boot',
  cb=storeModeCallback, value=1),
 Button((218, 60,100,120), bg='radio3-0', fg='store-dropbox',
  cb=storeModeCallback, value=2),
```

```
 Button((  0, 10,320, 35), bg='storage')],

# Screen mode 5 is size settings
[Button((  0,188,320, 52), bg='done', cb=doneCallback),
 Button((  0,  0, 80, 52), bg='prev', cb=settingCallback, value=-1),
 Button((240,  0, 80, 52), bg='next', cb=settingCallback, value= 1),
 Button((  2, 60,100,120), bg='radio3-1', fg='size-l',
  cb=sizeModeCallback, value=0),
 Button((110, 60,100,120), bg='radio3-0', fg='size-m',
  cb=sizeModeCallback, value=1),
 Button((218, 60,100,120), bg='radio3-0', fg='size-s',
  cb=sizeModeCallback, value=2),
 Button((  0, 10,320, 29), bg='size')],

# Screen mode 6 is graphic effect
[Button((  0,188,320, 52), bg='done', cb=doneCallback),
 Button((  0,  0, 80, 52), bg='prev', cb=settingCallback, value=-1),
 Button((240,  0, 80, 52), bg='next', cb=settingCallback, value= 1),
 Button((  0, 70, 80, 52), bg='prev', cb=fxCallback     , value=-1),
 Button((240, 70, 80, 52), bg='next', cb=fxCallback     , value= 1),
 Button((  0, 67,320, 91), bg='fx-none'),
 Button((  0, 11,320, 29), bg='fx')],

# Screen mode 7 is ISO
```

```
  [Button((  0,188,320, 52), bg='done', cb=doneCallback),
   Button((  0,  0, 80, 52), bg='prev', cb=settingCallback, value=-1),
   Button((240,  0, 80, 52), bg='next', cb=settingCallback, value= 1),
   Button((  0, 70, 80, 52), bg='prev', cb=isoCallback    , value=-1),
   Button((240, 70, 80, 52), bg='next', cb=isoCallback    , value= 1),
   Button((  0, 79,320, 33), bg='iso-0'),
   Button((  9,134,302, 26), bg='iso-bar'),
   Button(( 17,157, 21, 19), bg='iso-arrow'),
   Button((  0, 10,320, 29), bg='iso')],

  # Screen mode 8 is quit confirmation
  [Button((  0,188,320, 52), bg='done'   , cb=doneCallback),
   Button((  0,  0, 80, 52), bg='prev'   , cb=settingCallback, value=-1),
   Button((240,  0, 80, 52), bg='next'   , cb=settingCallback, value= 1),
   Button((110, 60,100,120), bg='quit-ok', cb=quitCallback),
   Button((  0, 10,320, 35), bg='quit')]
]

# Assorted utility functions ----------------------------------------------

def setFxMode(n):
        global fxMode
        fxMode = n
        camera.image_effect = fxData[fxMode]
```

```
        buttons[6][5].setBg('fx-' + fxData[fxMode])


def setIsoMode(n):
        global isoMode
        isoMode    = n
        camera.ISO = isoData[isoMode][0]
        buttons[7][5].setBg('iso-' + str(isoData[isoMode][0]))
        buttons[7][7].rect = ((isoData[isoMode][1] - 10,) +
         buttons[7][7].rect[1:])


def saveSettings():
        try:
         outfile = open('cam.pkl', 'wb')
         # Use a dictionary (rather than pickling 'raw' values) so
         # the number & order of things can change without breaking.
         d = { 'fx'    : fxMode,
              'iso'   : isoMode,
              'size'  : sizeMode,
              'store' : storeMode }
         pickle.dump(d, outfile)
         outfile.close()
        except:
         pass
```

```
def loadSettings():
        try:
          infile = open('cam.pkl', 'rb')
          d      = pickle.load(infile)
          infile.close()
          if 'fx'    in d: setFxMode(   d['fx'])
          if 'iso'   in d: setIsoMode(  d['iso'])
          if 'size'  in d: sizeModeCallback( d['size'])
          if 'store' in d: storeModeCallback(d['store'])
        except:
          pass

# Scan files in a directory, locating JPEGs with names matching the
# software's convention (IMG_XXXX.JPG), returning a tuple with the
# lowest and highest indices (or None if no matching files).
def imgRange(path):
        min = 9999
        max = 0
        try:
          for file in os.listdir(path):
            if fnmatch.fnmatch(file, 'IMG_[0-9][0-9][0-9][0-9].JPG'):
              i = int(file[4:8])
              if(i < min): min = i
              if(i > max): max = i
```

```
        finally:
          return None if min > max else (min, max)


# Busy indicator.  To use, run in separate thread, set global 'busy'
# to False when done.
def spinner():
        global busy, screenMode, screenModePrior
        buttons[screenMode][3].setBg('working')
        buttons[screenMode][3].draw(screen)
        pygame.display.update()

        busy = True
        n    = 0
        while busy is True:
          buttons[screenMode][4].setBg('work-' + str(n))
          buttons[screenMode][4].draw(screen)
          pygame.display.update()
          n = (n + 1) % 5
          time.sleep(0.15)

        buttons[screenMode][3].setBg(None)
        buttons[screenMode][4].setBg(None)
        screenModePrior = -1 # Force refresh
```

```
def takePicture():
        global busy, gid, loadIdx, saveIdx, scaled, sizeMode, storeMode, storeModePrior, uid


        if not os.path.isdir(pathData[storeMode]):
         try:
           os.makedirs(pathData[storeMode])
           # Set new directory ownership to pi user, mode to 755
           os.chown(pathData[storeMode], uid, gid)
           os.chmod(pathData[storeMode],
            stat.S_IRUSR | stat.S_IWUSR | stat.S_IXUSR |
            stat.S_IRGRP | stat.S_IXGRP |
            stat.S_IROTH | stat.S_IXOTH)
         except OSError as e:
           # errno = 2 if can't create folder
           print errno.errorcode[e.errno]
           return


        # If this is the first time accessing this directory,
        # scan for the max image index, start at next pos.
        if storeMode != storeModePrior:
         r = imgRange(pathData[storeMode])
         if r is None:
           saveIdx = 1
         else:
```

```
    saveIdx = r[1] + 1
    if saveIdx > 9999: saveIdx = 0
  storeModePrior = storeMode

# Scan for next available image slot
while True:
  filename = pathData[storeMode] + '/IMG_' + '%04d' % saveIdx + '.JPG'
  if not os.path.isfile(filename): break
  saveIdx += 1
  if saveIdx > 9999: saveIdx = 0

t = threading.Thread(target=spinner)
t.start()

scaled = None
camera.resolution = sizeData[sizeMode][0]
camera.crop       = sizeData[sizeMode][2]
try:
  camera.capture(filename, use_video_port=False, format='jpeg',
    thumbnail=None)
  # Set image file ownership to pi user, mode to 644
  # os.chown(filename, uid, gid) # Not working, why?
  os.chmod(filename,
    stat.S_IRUSR | stat.S_IWUSR | stat.S_IRGRP | stat.S_IROTH)
```

```
            img    = pygame.image.load(filename)
            scaled = pygame.transform.scale(img, sizeData[sizeMode][1])

        finally:
            # Add error handling/indicator (disk full, etc.)
            camera.resolution = sizeData[sizeMode][1]
            camera.crop       = (0.0, 0.0, 1.0, 1.0)

        busy = False
        t.join()

        if scaled:
            if scaled.get_height() < 240: # Letterbox
                screen.fill(0)
            screen.blit(scaled,
                ((320 - scaled.get_width() ) / 2,
                 (240 - scaled.get_height()) / 2))
            pygame.display.update()
            time.sleep(2.5)
            loadIdx = saveIdx

def showNextImage(direction):
    global busy, loadIdx
```

```
        t = threading.Thread(target=spinner)
        t.start()

        n = loadIdx
        while True:
          n += direction
          if(n > 9999): n = 0
          elif(n < 0):  n = 9999
          if os.path.exists(pathData[storeMode]+'/IMG_'+'%04d'%n+'.JPG'):
            showImage(n)
            break

        busy = False
        t.join()

def showImage(n):
        global busy, loadIdx, scaled, screenMode, screenModePrior, sizeMode, storeMode
        t = threading.Thread(target=spinner)
        t.start()

        img     = pygame.image.load(
                pathData[storeMode] + '/IMG_' + '%04d' % n + '.JPG')
        scaled   = pygame.transform.scale(img, sizeData[sizeMode][1])
        loadIdx  = n
```

```
            busy = False
            t.join()

            screenMode      =  0 # Photo playback
            screenModePrior = -1 # Force screen refresh

# Initialization -------------------------------------------------------

# Init framebuffer/touchscreen environment variables
os.putenv('SDL_VIDEODRIVER', 'fbcon')
os.putenv('SDL_FBDEV'     , '/dev/fb1')
os.putenv('SDL_MOUSEDRV'   , 'TSLIB')
os.putenv('SDL_MOUSEDEV'   , '/dev/input/touchscreen')

# Get user & group IDs for file & folder creation
# (Want these to be 'pi' or other user, not root)
s = os.getenv("SUDO_UID")
uid = int(s) if s else os.getuid()
s = os.getenv("SUDO_GID")
gid = int(s) if s else os.getgid()

# Buffers for viewfinder data
rgb = bytearray(320 * 240 * 3)
```

```
yuv = bytearray(320 * 240 * 3 / 2)


# Init pygame and screen
pygame.init()
pygame.mouse.set_visible(False)
screen = pygame.display.set_mode((0,0), pygame.FULLSCREEN)


# Init camera and set up default values
camera           = picamera.PiCamera()
atexit.register(camera.close)
camera.resolution = sizeData[sizeMode][1]
#camera.crop      = sizeData[sizeMode][2]
#camera.crop      = (0.0, 0.0, 1.0, 1.0) #Orginall
zooma=.4
shiftY=.08
esides=.04
#Look up value for ROI default (0.0, 0.0, 1.0, 1.0) (x, y, w, h)
camera.crop      = (zooma+esides/2, zooma-shiftY, 1-zooma*2-esides, 1-zooma*2-esides)


# Leave raw format at default YUV, don't touch, don't set to RGB!
# Load all icons at startup.
for file in os.listdir(iconPath):
  if fnmatch.fnmatch(file, '*.png'):
    icons.append(Icon(file.split('.')[0]))
```

```
# Assign Icons to Buttons, now that they're loaded
for s in buttons:       # For each screenful of buttons...
  for b in s:           #  For each button on screen...
    for i in icons:     #   For each icon...
      if b.bg == i.name: #    Compare names; match?
        b.iconBg = i    #     Assign Icon to Button
        b.bg     = None  #     Name no longer used; allow garbage collection
      if b.fg == i.name:
        b.iconFg = i
        b.fg     = None

loadSettings() # Must come last; fiddles with Button/Icon states

# Main loop ----------------------------------------------------------------
videostarted=0
VideoNameSetter()

while(True):
  exit_state=GPIO.input(ExitButton)
  Picture_state=GPIO.input(PICBUTTON)
  LowPot_state=GPIO.input(LowButton)
  HighPot_state=GPIO.input(HighButton)
```

```
 ScalePotValues()
 camera.annotate_text=str("%.2f" % PressureReading)

 #No TOuch SCreen controls
 if Picture_state==False:
  print("Taking your picture")
  print("Starting Video")
  if videostarted==0:
    camera.start_recording(VideoName,splitter_port=3)
    camera.annotate_text=str('STARTED')
    videostarted=1
##   pos =[100,100]
##   for b in buttons[screenMode]:
##       if b.selected(pos): break

## # Process touchscreen input
## while True:
##   for event in pygame.event.get():
##     if(event.type is MOUSEBUTTONDOWN):
##       pos = pygame.mouse.get_pos()
##       print(pos)
##       for b in buttons[screenMode]:
##         if b.selected(pos): break
```

```
##    # If in viewfinder or settings modes, stop processing touchscreen
##    # and refresh the display to show the live preview.  In other modes
##    # (image playback, etc.), stop and refresh the screen only when
##    # screenMode changes.
##    if screenMode >= 3 or screenMode != screenModePrior: break

 # Refresh display
 if screenMode >= 3: # Viewfinder or settings modes
  stream = io.BytesIO() # Capture into in-memory stream
  camera.capture(stream, use_video_port=True, format='raw')
  stream.seek(0)
  stream.readinto(yuv)  # stream -> YUV buffer
  stream.close()
  yuv2rgb.convert(yuv, rgb, sizeData[sizeMode][1][0],
    sizeData[sizeMode][1][1])
  img = pygame.image.frombuffer(rgb[0:
    (sizeData[sizeMode][1][0] * sizeData[sizeMode][1][1] * 3)],
    sizeData[sizeMode][1], 'RGB')
 elif screenMode < 2: # Playback mode or delete confirmation
  img = scaled      # Show last-loaded image
 else:              # 'No Photos' mode
  img = None        # You get nothing, good day sir

 if img is None or img.get_height() < 240: # Letterbox, clear background
```

```
    screen.fill(0)
  if img:
    screen.blit(img,
      ((320 - img.get_width() ) / 2,
       (240 - img.get_height()) / 2))

# Exitthe progrgram
  if exit_state==False:
    print("Ending Program: Goodbye")
    if videostarted==1:
        camera.stop_recording(splitter_port=3)
        ConvertVideo()

    board.exit() #otherwise will cause bad errors
    break

# Calibrating the low state
  if LowPot_state==False:
    SetLowPot()
    print("Calibrated Low State")

# Calibrating the high state
  if HighPot_state==False:
    SetHighPot()
```

```
   print("Calibrated High State")


  # Overlay buttons on display and update <- Create a button that updates the pot state on
picture

##  for i,b in enumerate(buttons[screenMode]):

##    b.draw(screen)


  pygame.display.update()


  screenModePrior = screenMode
```