ABSTRACT

REAL TIME ANALYSIS OF EYE MOVEMENTS USING COMPUTER AIDED SOFTWARE

by
Farhan Ahmad

Eye movement analysis is a slow and tasking job that require a trained vision analyst to sift through hundreds of eye movements. The difficulty of the task increases when a subject has undiagnosed eye related disorders. One such disorder is Convergence insufficiency (CI) which affects nearly 12 million people in the US. Convergence insufficiency occurs when the eyes are unable to coordinate simultaneous inward movement for near focal tracking. Individuals with CI are unable to coordinate eye movements this can result in many symptoms such as double vision, blurred vision, headaches, vision strain, and fatigue. A diagnosis for CI relies on the experience of the optometrist and their ability to detect a deviation in a patient's eyes as they track a near focal object, it also requires the patient to respond back to the optometrist which can be subjective. In conjunction with a robust eye tracking camera system, the software described herein, Real Time Analysis of Eye Movements using Computer Aided Software (R.E.T.I.N.A.S.), allows for large scale screening by processing eye movement data quickly and accurately as compared to the current standard that requires a human vision analyst. R.E.T.I.N.A.S. can remove the subjectivity that is currently a component in CI screening. This allows for inexperienced users who are not trained in optometry to screen for CI by reviewing the outputted metrics that R.E.T.I.N.A.S. provides. This type of accessibility allows for patients affected by CI to get necessary therapy quickly, which can greatly improve the patient’s quality of life.

REAL TIME ANALYSIS OF EYE MOVEMENTS USING COMPUTER AIDED SOFTWARE

by
Farhan Ahmad

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
In Partial Fulfillment of the Requirements for the Degree of
Master of Science in Biomedical Engineering

Department of Biomedical Engineering

May 2021

APPROVAL PAGE

REAL TIME ANALYSIS OF EYE MOVEMENTS USING COMPUTER AIDED SOFTWARE

Farhan Ahmad

Dr. Chang Yaramothu, Thesis Advisor — Date
Assistant Professor of Engineering Technology and Biomedical Engineering, NJIT

Dr. Tara L. Alvarez, Thesis Co-Advisor — Date
Professor of Biomedical Engineering, NJIT

Dr. Xiaobo Li, Committee Member — Date
Associate Professor of Biomedical Engineering, NJIT

John Vito d'Antonio-Bertagnolli, Committee Member — Date
Adjunct Professor of Biomedical Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: Farhan Ahmad

Degree: Master of Science

Date: May 2021

Undergraduate and Graduate Education

- Bachelor of Science in Biomedical Engineering,
  New Jersey Institute of Technology, Newark NJ, 2020

Major: Biomedical Engineering

ACKNOWLEDGMENT

To my advisor, Dr. Yaramothu – thank you for helping me throughout the development process, giving me feedback on ways to improve R.E.T.I.N.A.S and allowing me to better understand the academic side of the project. To my co-advisor, Dr. Alvarez – thank you for allowing me to work with you for my senior year capstone which lead to this project and sharing your immense knowledge of eye movements. To my committee member, Dr. Li – thank you for serving on my committee, and for your expertise on attention. To my committee member, John Vito d'Antonio-Bertagnolli – for setting a great example on how to tackle this project and sharing your experience having gone through something similar. To my friend, Sebastian – thank you for constant words of encouragement and your cynical commentary. To my Capstone member, Ayushi – for helping me code R.E.T.I.N.A.S and help me figure out different strategies when I was stuck. To my friends, Kyle and Jon – for their great sense of humor and their upbeat attitude towards life. To my parents, Shafiq and Khushbun – thank you for everything you have done. Without your hard work I would have never been able to accomplish what I have. Finally, to my uncle, Dr. Ali – for constantly pushing me and giving me words of encouragement to always apply myself.

TABLES OF CONTENTS

TABLES OF CONTENTS
(Continued)

LIST OF FIGURES

LIST OF FIGURES
(Continued)

LIST OF TABLES

LIST OF TERMS

R.E.T.I.N.A.S. – Real-time Eye Tracking Impartial Numerical Analysis Software.

Convergence – The inward rotation of the eyes.

Divergence – The outward rotation of the eyes.

Saccades – The rapid movement of the eyes in the same direction between points of visual fixation. They are a type of eye movement recorded in the data files but are considered artifacts when they occur during a vergence movement.

Vergence – Convergent and/or divergent eye movements.

Vergence Demand – The angular requirement from a visual stimulus to produce a single and clear image, measured in degrees.

Version – Saccadic eye movements, also known as the conjugate rotation (turning) of both eyes in the same direction.

Artifacts – Recorded blinks or saccades

Blink – Artifacts that cause saturation in the signal.

CSV – comma-separated values, a way to store data in a tabular form in a text file.

DC Voltage Offset – A constant which is added to the raw eye movement data before a gain value is applied.

Gain Value – A unique and calculated value from raw eye movement data that converts the native raw data file units of voltage to degrees.

Final Amplitude – The measured amplitude of an eye movement from 0 degrees to amplitude at settling.

Latency – The time at which a recorded eye movement can reach 5% of the expected vergence demand.

Peak Velocity – The highest recorded first derivative of position value recorded during the transient period.

Settling Time – The time at which a recorded eye movement can sustain the required vergence or version demand.

Transient – The interval between the onset of an eye movement and settling onto a visual target.

CHAPTER 1

INTRODUCTION

1.1 Objective

The main objective of this thesis is to present an alternative way to analyze eye movement data without the direct need of a trained vision analyst. This will be done using a program developed in the Python, an open-source coding language. The accuracy and the speed of the eye movements analyzed will be compared to that of a human analyst to determine the efficacy of R.E.T.I.N.A.S.

1.2 Visual Perception

Visual perception is the ability of the brain to coordinate with the eye to receive, interpret and adapt to a visual stimulus. This is not the same as visual acuity or how clearly one can see. Normally the light first enters the eye through the cornea and then it is focused on to the retina by the lens. The retina converts the light into neuronal signals. The conversion is facilitated by specialized photoreceptive cells called cones and rods. The neural signals then travel down the optic nerve to other intermediaries and finally to the visual cortex. From there the brain processes the information and adjusts the eyes to follow or focus on a stimulus. The eye uses the lens to focus on a stimulus, it is achieved by the changing the shape of the flexible lens by contracting or relaxing the ciliary muscles, this process is called accommodation. The rotation of the eye is regulated by 6 muscles: the lateral rectus, medial rectus, the inferior rectus, superior rectus, inferior oblique, and the superior oblique,

as seen in **Figure 1.1**. They attach at different positions on the eye and rotate it about a point at the center of the eye.

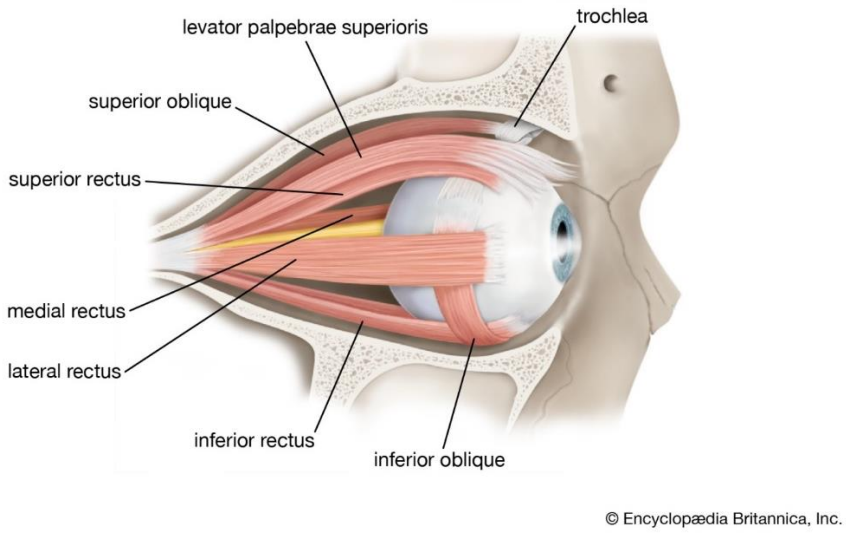

Figure 1.1 The 6 different muscles that control the movement of the eye.
Source: [1]

1.3 Types of Eye Movements

There are 5 main types of eye movements: saccades, vergence, smooth pursuit, optokinetic and vestibulo-ocular. In this paper the focus will be on the first two, saccades and vergence. Saccades are the rapid conjugate movement of the eye, that is usually used to quickly scan an environment. Conjugate refers the ability of the eyes to coordinate together to maintain binocular vision. They appear as high velocity movements that are used to quickly adjust to a moving stimulus. Vergence movements are relatively slow and deconjugate eye movements compared to saccades but are mainly used to fixate on objects. There are two types of vergence directions: convergence and divergence. In convergence the eyes rotate inwards and towards each other; this is mainly used to focus on near objects. Divergence is the outwards rotation of eye, necessary for focusing on far objects.

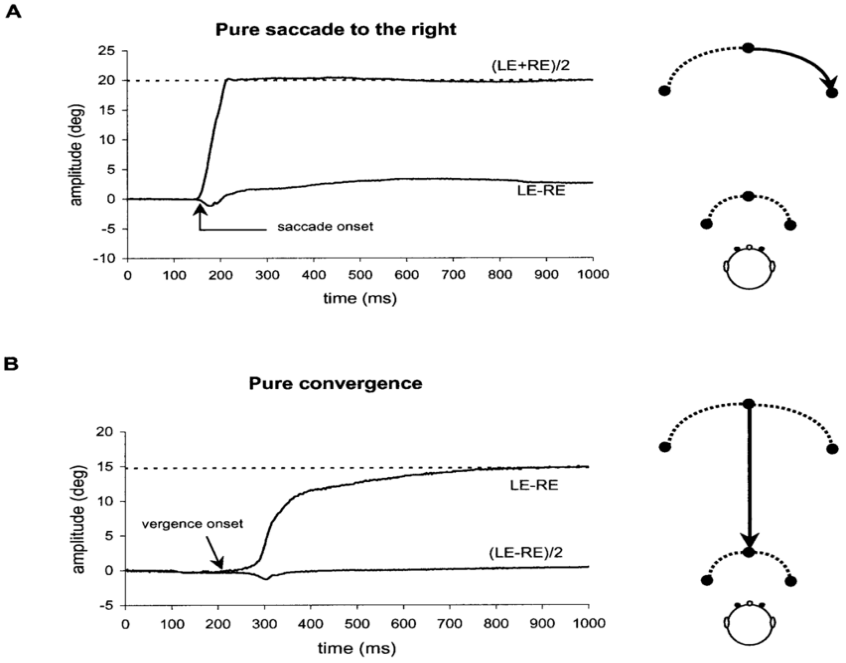

Figure 1.2 The figure above illustrates the difference between a saccadic eye movement and a vergence eye movement. Left shows the position traces and the right shows the target movement. Top is a saccadic eye movement and bottom is a vergence eye movement.
Source: [2]

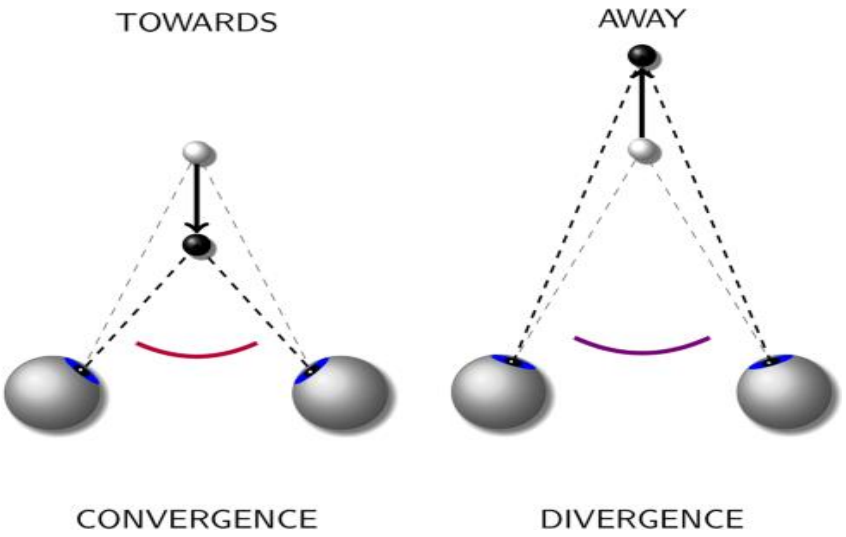

Figure 1.3 The figure above illustrates the difference between convergence and divergence eye movements. The left side depicts the eyes rotating inwards to fixate on a target that is coming towards the subject. The right side depicts the eyes rotating outwards to fixate on a target as it is moving away from the subject.

Source: [3]

1.4 Parts of an Eye Movements

Eye movements are separated into two main parts, transient portion, and the steady state portion. Transient is the dynamic portion of the eye movement where the subjects are trying to fixate on a target stimulus. This can be seen in **Figure 1.3** where the position rapidly increases between the time intervals of .25s and .75s. Steady state portion of the eye movement is when the subject has fixated on the target. Steady is the remaining portion of the eye movements after 1s.

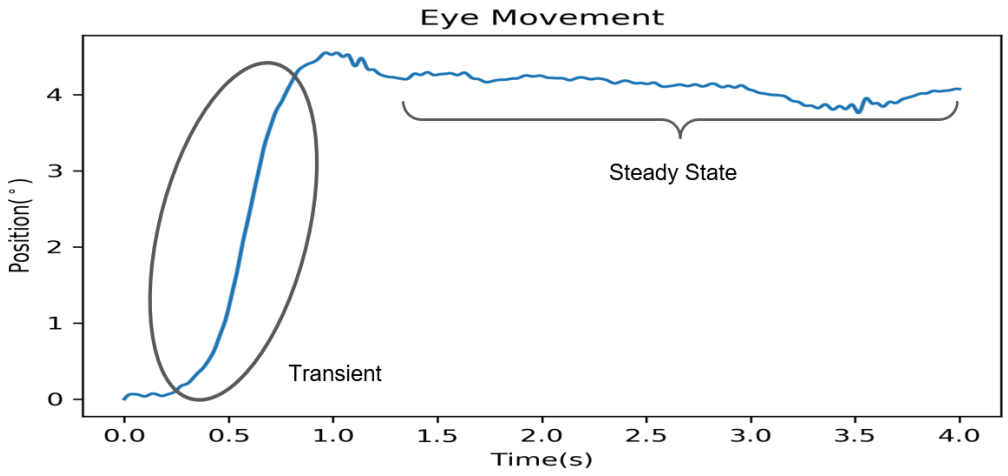

Figure 1.4 The two different parts of the eye movement; transient and steady state.

1.5 Classification

Classification refers to the quality of the movement, the quality of the average eye movement of a patient can provide a trained vision analyst the information necessary to determine if the patient has CI. This quality is determined by the data metrics in the eye movement and the number of blinks, and saccades. There are 5 main data metrics; Final Amplitude, Peak Velocity, Time to Peak Velocity, Latency and Settling Time. Final

amplitude is defined as the average of the last 1 second of the eye movement or position data, measured in degrees. Peak velocity is defined as the maximum of the derivative of the position data, measured in degrees/s. Time to peak velocity is measured by taking the time at which the maximum occurs. Finally, latency and settling time are measured by taking the time when the 5% and 95% respectively, of the final amplitude is reached. Another way the quality is determined is by the number of blinks and where they occur. Blinks are an artifact that are caused when the patient closes their eye lids during the recording of the data which can saturate the data. If the blink occurs during the transient portion of the eye movement, as seen in **Figure 1.6,** this can artificially increase the peak velocity reported by R.E.T.I.N.A.S.. Saccades are not only a type of eye movement, but also an artifact when they occur in a vergence trace of an eye movement.

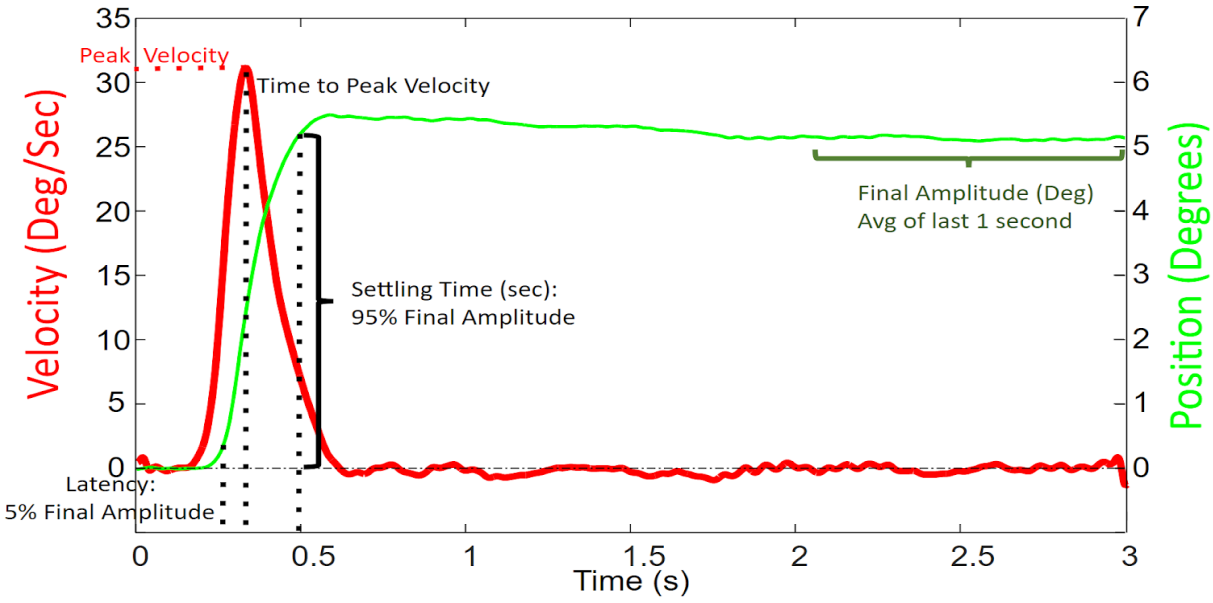

Figure 1.5 The figure above illustrates a position trace of an eye movement (green) and its respective velocity trace (red). The left y-axis is the amplitude values for the velocity trace in degrees/second and the right y-axis is the amplitude value of the position trace in degrees. The x-axis is the amplitude values of time in seconds. The 5 main data metrics necessary for classification are labeled as well.

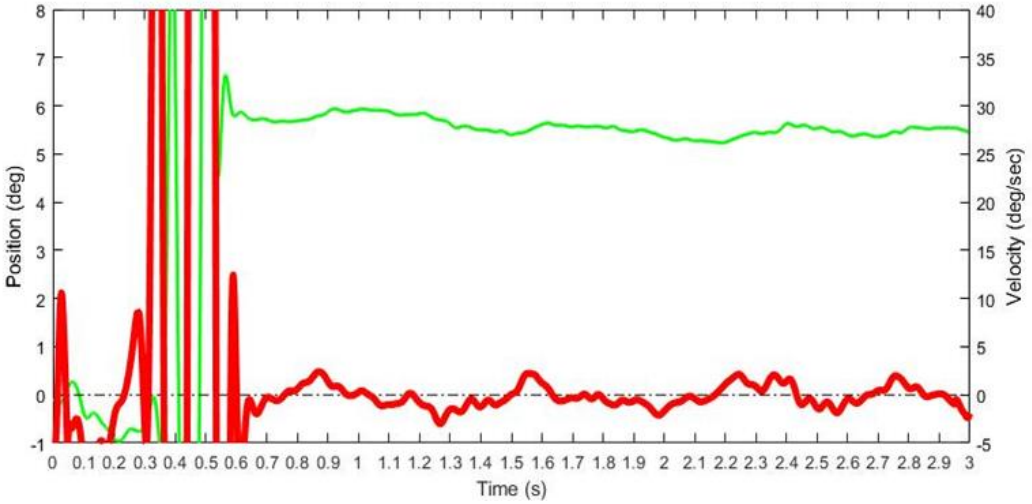

Figure 1.6 The figure above illustrates a blink artifact occurring at transient portion of the eye movement. The artifact occurs between .2 to .5 seconds and causes the data to fluctuate between a large negative and positive value. The position trace of the eye movement (green) and its respective velocity trace (red) are graphed. The right y-axis is the amplitude values for the velocity trace in degrees/second and the left y-axis is the amplitude value of the position trace in degrees. The x-axis is the amplitude values of time in seconds.

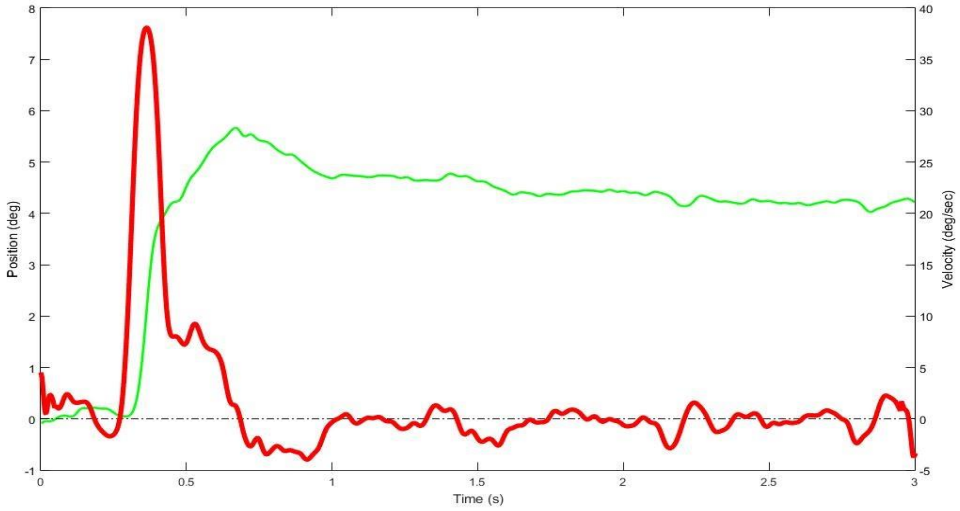

Figure 1.7 The figure above displays a small saccade at .5 seconds which is hard to notice in the position trace but is more visible in the velocity trace. The saccade is causing a decrease in the velocity at .5 seconds, but it is not affecting the peak velocity.

1.6 Convergence Insufficiency

The inability to have effective binocular coordination affects one's quality of life, an example is convergence insufficiency (CI). CI is an oculomotor dysfunction which affects nearly 16 million Americans or 4-13% of the population [4]. Symptoms of this dysfunction are blurry vision, onset of headaches and straining of the eyes from close range work which can inhibit one's quality of life just after 15-20 minutes of reading. The current methods to analyze binocular eye movements and determine if a patient may suffer from convergence insufficiency requires a trained optometrist or the processing of eye movement from data analysts in the research setting [5]. The utilization of human analysts to classify CI introduces biased outputs from the recorded eye movements, and this can vary between each analyst. As for optometrist many use the Convergence Insufficiency Symptom Survey, near point of convergence or exodeviation to diagnose CI [6]. All of these are time consuming and require a trained physician to perform or interpret the results.

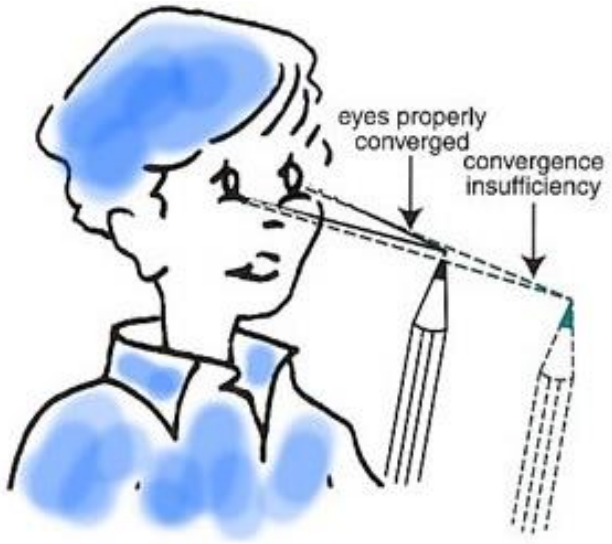

Figure 1.8 The image above shows how a patient with CI will have to move an object further than a healthy patient before they can properly see it.
Source: [7]

1.7 Automation of Data Analysis

One of the major issues that arises with classifying eye movements of patients with CI is that there is a large variation between eye movements. A vision analyst can take up to a week just classifying eye movements of one patient, this time can be increased if the patient has CI. The analyst not only has to go through the position trace, but they must also check if the velocity trace is largely impacted in the transient portion of the eye movement or just a small fluctuation has occurred, as seen in **Figure 1.7**. The analyst then must retrieve all the data metrics which are done manually. If an artifact occurs at any point in the eye movement, which are more prevalent in patients with CI, then either the eye movement must be classified as bad, or the analyst must manually change the parameter for calculating the 5 data metrics. The parameters selected can also vary depending on the vision analyst and their experience. R.E.T.I.N.A.S. tries to reduce the time for analyzing the eye movement and more importantly it tries to save the vision analyst the recalculation time for the 5 data metrics by automating the metric retrieval, while also removing the subjectivity introduced by the human analyst.

CHAPTER 2

METHODS

2.1 Raw Data Collection

The data for this project was collected using a head mounted display with an ISCAN RK-826PCI infrared eye movement tracking system (Burlington, MA, USA) operating at a wavelength of 940 nm and sampling at a rate of 240 Hz. The left and right eye are collected independently, and the data digitized using a custom software written using LabVIEW™ 2013 SP1 Virtual Instrument (National Instrument, Austin, TX, USA), called NJIT VisulaEyes 2020 [8].

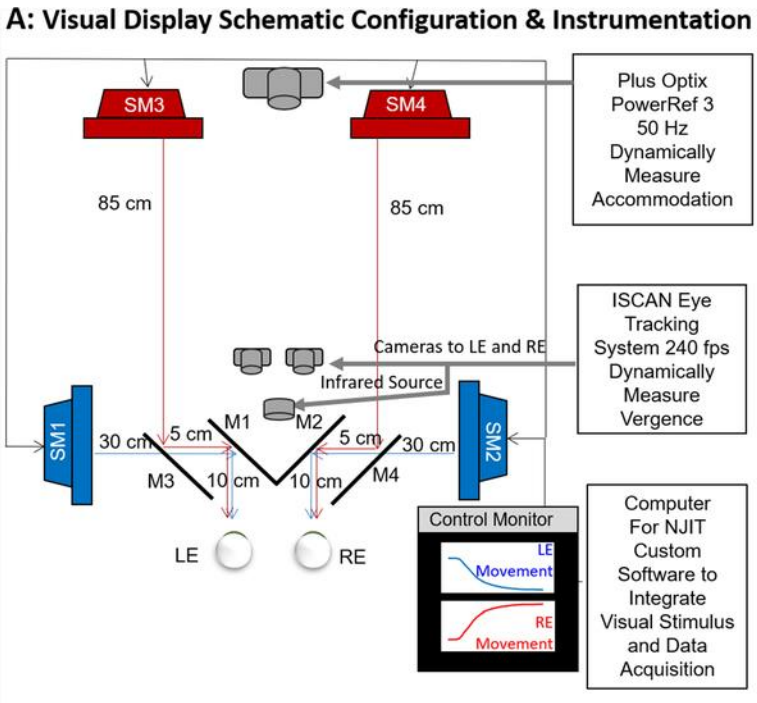

Figure 2.1 The ISCAN setup used to provide stimulus and collect data.
Source: [8]

2.2 Subject Selection

Five healthy subjects were selected for their high data retention when a human analyst analyzed the subjects eye movements. Data retention was calculated as the ratio of the number of eye movements classified as good over the number of total eye movements. The age and the gender of the participant were unknown. The data collection occurred at The University of New Mexico.

Table 2.1 Subject Data Retention

Subject Id	Data Retention
23389	65%
76947	39%
41857	51%
17530	47%
04370	40%

2.3 R.E.T.I.N.A.S. Software

The R.E.T.I.N.A.S. software consists of 4 main functions; Load Data function, Preprocess function, Analyze function and Plot function. The Load Data function is launched when the user presses the “Load Data” button, from there the user selects a subject file that contains the eye movements. The format of the file is a “.mat” which is a proprietary extension of MATLAB, the data is already parsed into the different cells and the function simply exports the data into Python lists.

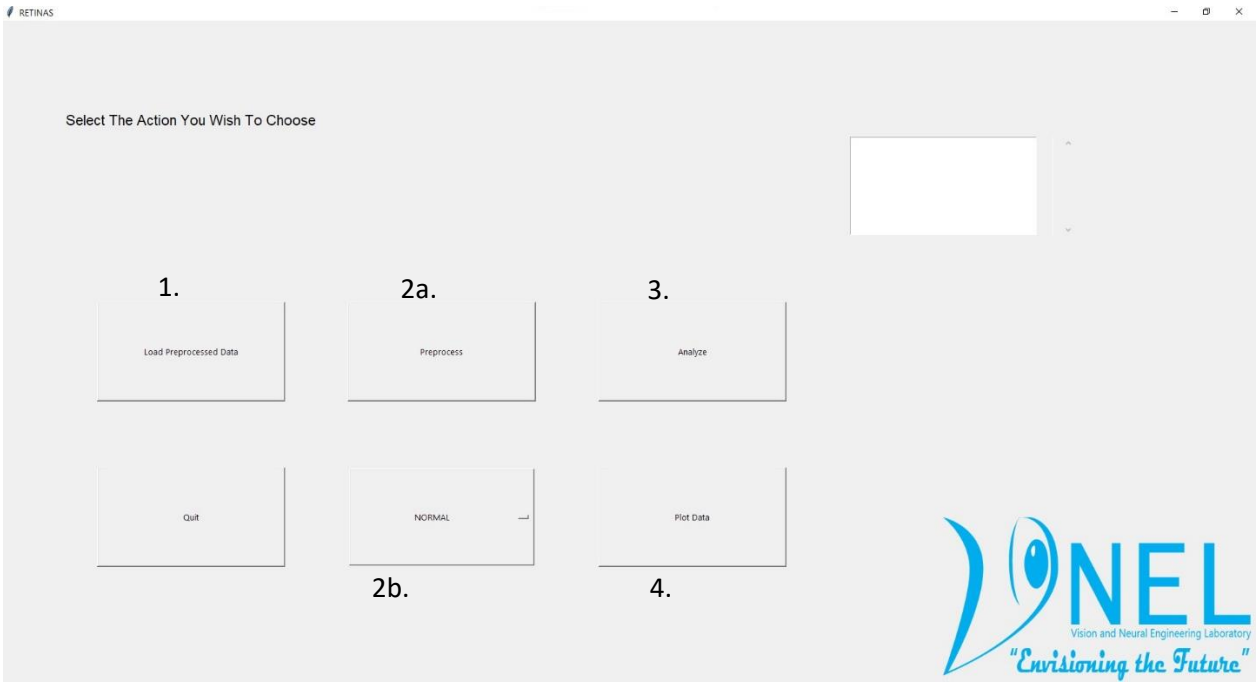

Figure 2.2 The Figure above shows the layout of R.E.T.I.N.A.S and the 4 main functions and a drop-down menu that lets the user decide which type of calibration they want to use. The first function is the "Load Data" labeled 1, the next function is the "Preprocess" function labeled 2a and it's corresponding drop-down menu labeled 2b. The drop-down menu has 3 different choices for calibration: "NORMAL", "SACCADES" and "AUTOMATIC". The third function is "Analyze" labeled 3 and last function is "Plot" labeled 4.

The next function is the Preprocess function, and it is launched automatically when the user presses the "Preprocess" button. This function calibrates the data and filters it using a 10^{th} order low pass Butterworth filter at 40 Hz. The user selects how the data is calibrated using a drop-down menu, there are three modes for calibration: "NORMAL", "SACCADES" and "AUTOMATIC". The NORMAL mode uses 1°, 3° and 5° monocular vergence stimuli to calibrate the data. The data is then averaged, and a liner interpolation is taken across the three data points. The slope of the line best fit is used as the gain value. The SACCADES mode uses -10°, 0 and 10° monocular stimuli and the same process as the NORMAL mode is used to calculate the gain value. The algorithm used to calculate

these gain value uses a python library called SciPy and a function called *linregress*. The function has two inputs the x input and the y input. The x input was the list of the amplitude values of the stimuli for the respective modes. For NORMAL mode, the x input is 1, 3 and 5 as for SACCADES the x input is -10, 0 and 10. The y input is the average of the calibration data at each of the stimuli.

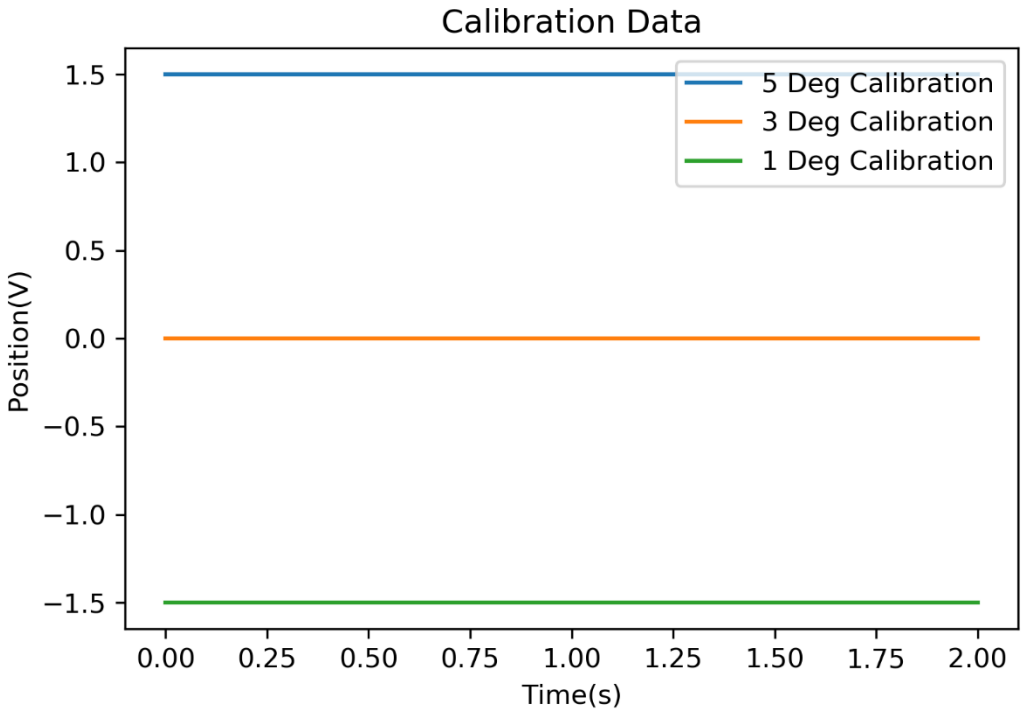

Figure 2.3 The plot of a hypothetical calibration data for 1°, 3° and 5° used for calculating the gain value for NORMAL mode. The y-axis is the position data in voltage and the x-axis is the time data in seconds.

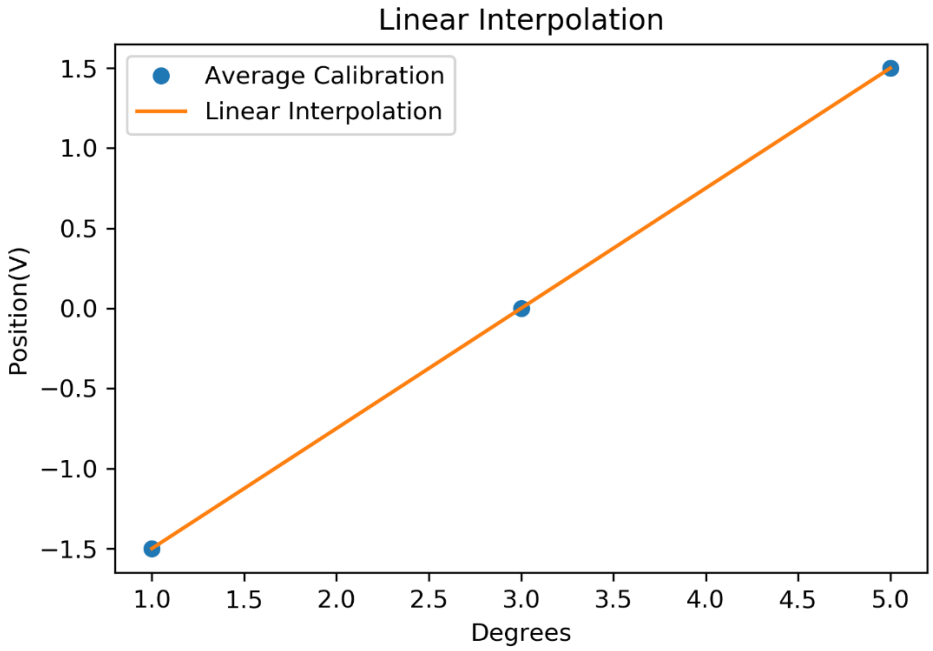

Figure 2.4 The figure above illustrates a hypothetical value for the linear interpolation used to calculate the gain value for NORMAL and SACCADE mode. The y-axis is the average of the position data in voltage and the x-axis is the target stimulus as in 1, 3 and 5 in degrees.

The AUTOMATIC mode uses the saccade left eye 5° and saccade right eye 5° movements to calculate a gain value. It does this by increasing the gain value of both eyes symmetrically until the final amplitude reaches 5°, as seen in **Figure 2.5.** This is done for all the eye movements and at the end an average is taken to determine the gain value used to calibrate the data. The gain value is a constant multiplier that convert the data from millivolts to degrees.

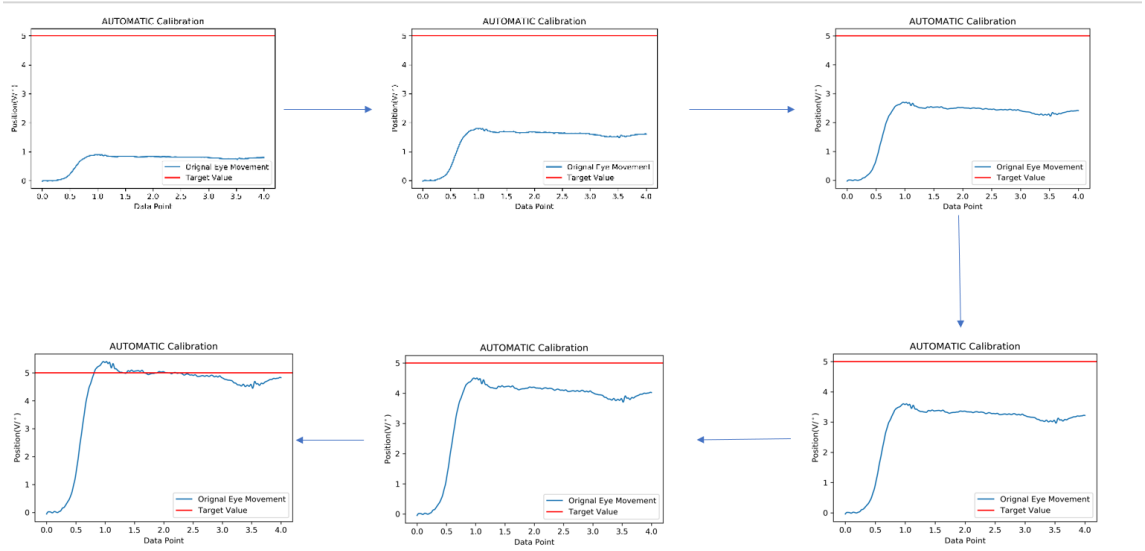

Figure 2.5 The figure above illustrates the process of how AUTOMATIC mode calculates the gain calibration by incrementally increasing the gain value until a target value is reached. The y axis is the position data in degrees and the x axis is the time elapsed in seconds.

The third and the main function is the Analyze function, and it is launched when the user presses the "Analyze" button. The purpose of this function is to find the 5 data statistics and classify the eye movement as bad or good. Before all of this happens the data first goes through artifact removal where blinks and saccades are linearly extrapolated to best fit the removed data, see **Figure 2.7**. In the "Analyze" function two python libraries are used SciPy and NumPy. In SciPy the *savgol_filter* function is used to calculate the velocity trace or the derivative of the position trace. It is also used to calculate the acceleration trace, or the second derivative of the position trace as seen in **Figure 2.6**. The acceleration trace is then used to find largest acceleration values. If the acceleration reaches 5 degrees/second^2 then a saccade or a blink has occurred and then the data around the peak

acceleration is then deleted using the *delete* function in the NumPy library. After deleting the data, a linear interpolation is made using the *interp1d* function in the SciPy library.

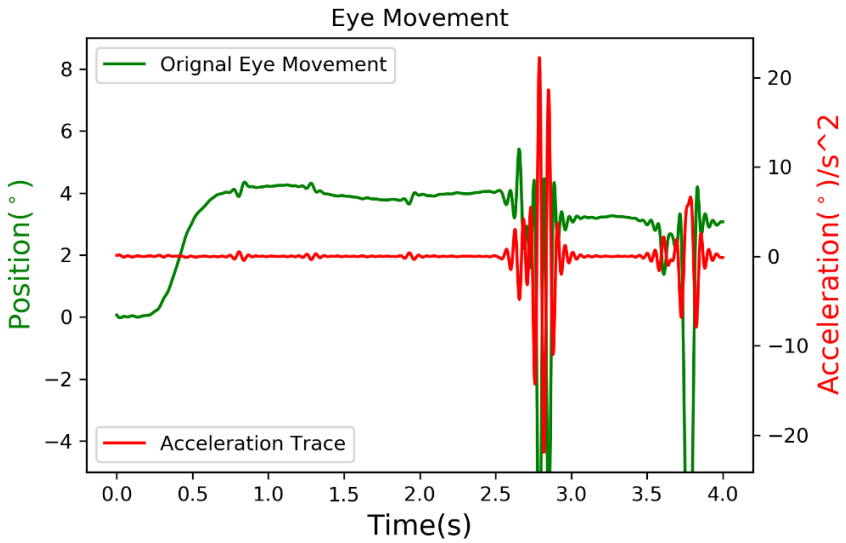

Figure 2.6 The acceleration trace of an eye movement calculated using the savgol_filter function. The position trace of the eye movement (green) and its respective acceleration trace (red) are graphed. The right y-axis is the amplitude values for the acceleration trace in degrees/second2 and the left y-axis is the amplitude value of the position trace in degrees. The x-axis is the amplitude values of time in seconds.

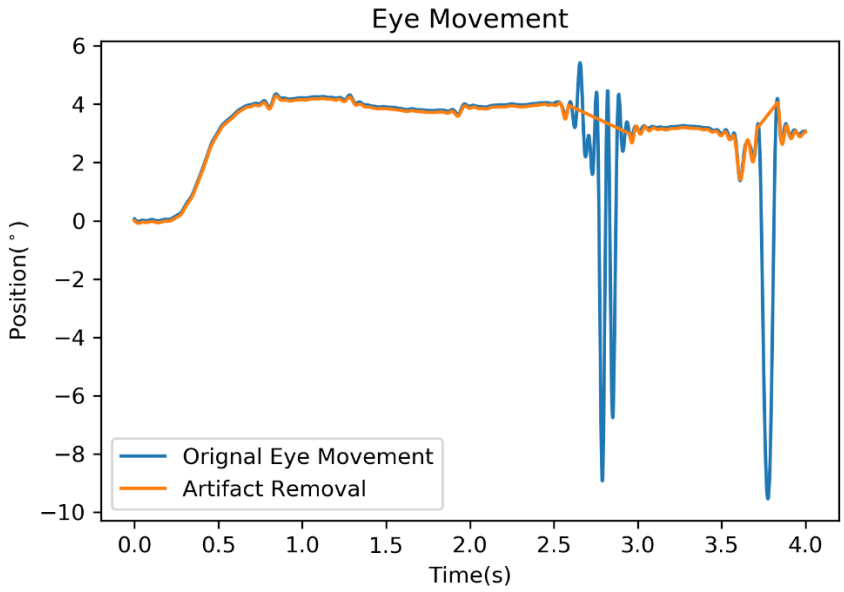

Figure 2.7 The figure above shows how two blink artifacts are removed using linear interpolation between a blink.

After artifact removal, the data metrics are retrieved from the eye movement, when the values of the metrics are too large or too small for a healthy participant to achieve then the eye movement is classified as "bad". The boundaries for final amplitude are as follows, it is considered too large if it is 1.5 x target amplitude, for example for a 4-degree eye movement a final amplitude above 6 degrees is considered "bad". The final amplitude value is considered too small if it is .25 x target amplitude or less than 1 degree for a 4-degree eye movement. For latency, a boundary of .150s to .500s is used to determine if it can be classified as "good". The boundary for peak velocity ranged from 7-25 degrees/seconds^2 in accordance to a previous paper [9]. Finally for time to peak velocity the boundary ranged from .200s to 1s. When a value is outside of these ranges the R.E.T.I.N.A.S. output is changed to "9999" to show that the values gotten were insufficient. The data metrics are then exported to a CSV.

The last function of R.E.T.I.N.A.S. plots the data for visualization purposes, but also allows the user to manually change the classification. The last functionality has not been fully implemented.

CHAPTER 3

RESULTS

3.1 Analysis of Eye Movements

The data consisted of convergent and divergent 4° eye movements. It was then analyzed using a custom MATLAB software by the human analyst. The human analyst did not need settling time to classify the data and hence it is unavailable for comparison. Table 3.1, 3.2, 3.3 and 3.4 shows the comparison of the average data metrics analyzed by the human analyst and R.E.T.I.N.A.S., in total 65 eye movements that were able to be analyzed by both R.E.T.I.N.A.S. and the human analyst were used.

Table 3.1 Latency Comparison

Latency		
	Human Analyst	R.E.T.I.N.A.S.
Average	0.275	0.328
STD	0.163	0.297

Table 3.2 Final Amplitude Comparison

Final Amplitude		
	Human Analyst	R.E.T.I.N.A.S.
Average	3.674	1.873
STD	0.936	1.289

Table 3.3 Peak Velocity Comparison

Peak Velocity		
	Human Analyst	R.E.T.I.N.A.S.
Average	11.371	14.035
STD	3.858	6.929

Table 3.4 Time to Peak Velocity Comparison

Time to Peak Velocity		
	Human Analyst	R.E.T.I.N.A.S.
Average	0.426	0.638
STD	0.125	0.439

The total average and the standard of deviation was taken across the multiple subjects to determine the behavioral data. A paired t-test was used to measure the differences between the averages. A Pearson correlation test was also done to measure the statistical relationship between the metrics received from human analyst and R.E.T.I.N.A.S., with a N = 65. Figures 3.1-3.8 illustrate the main differences between the human analyst and the R.E.T.I.N.A.S output.

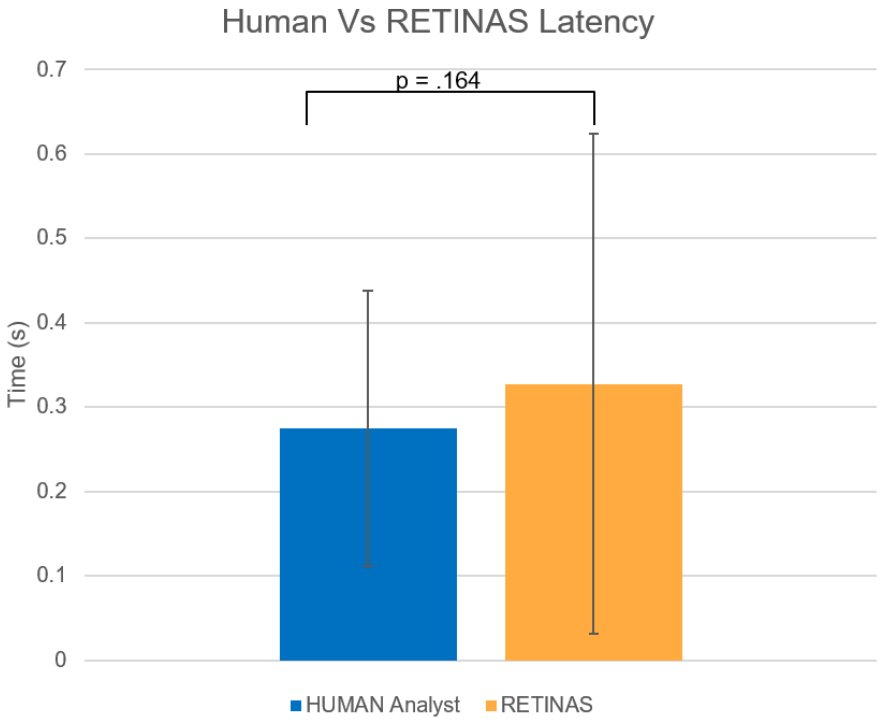

Figure 3.1 Average latency comparison between the data analyzed by human analyst Vs R.E.T.I.N.A.S.. The human analyst achieved an average latency of 0.275s with STD of .163, whereas R.E.T.I.N.A.S. reported a value of .328s with a STD of .297.

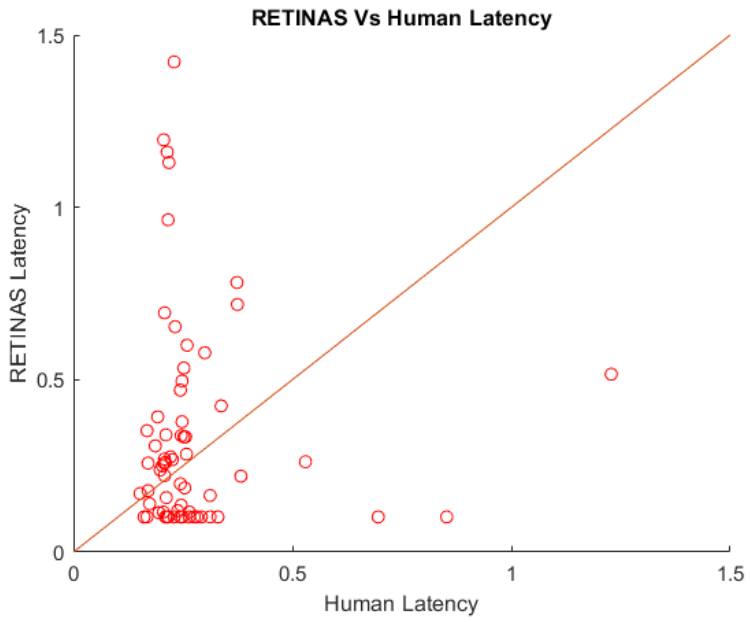

Figure 3.2 Correlation of Latency between human analyst and R.E.T.I.N.A.S.. On the x axis, it is the data analyzed by the human analyst and on the y axis it is the data analyzed by R.E.T.I.N.A.S.. The Pearson correlation value of -.0085 and a P-value of .946.

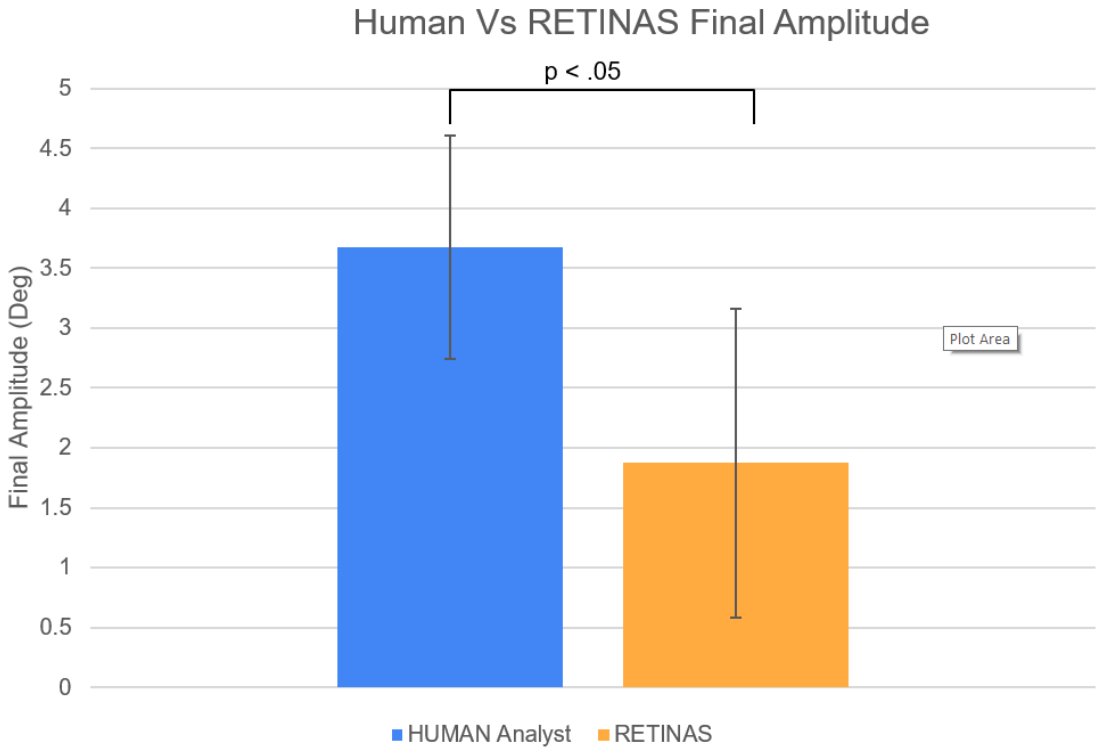

Figure 3.3 Average final amplitude comparison between the data analyzed by human analyst Vs R.E.T.I.N.A.S.. The human analyst achieved a final amplitude of 3.674° with a

standard of deviation of .936, whereas R.E.T.I.N.A.S. reported a value of 1.873° with a STD of 1.289.

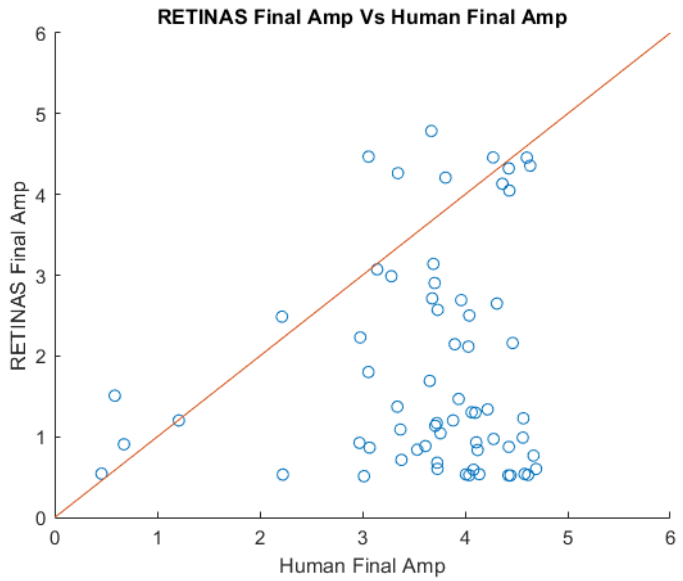

Figure 3.4 Correlation of final amplitude between human analyst and R.E.T.I.N.A.S.. The Pearson correlation value of .122 and a P-value of .332.

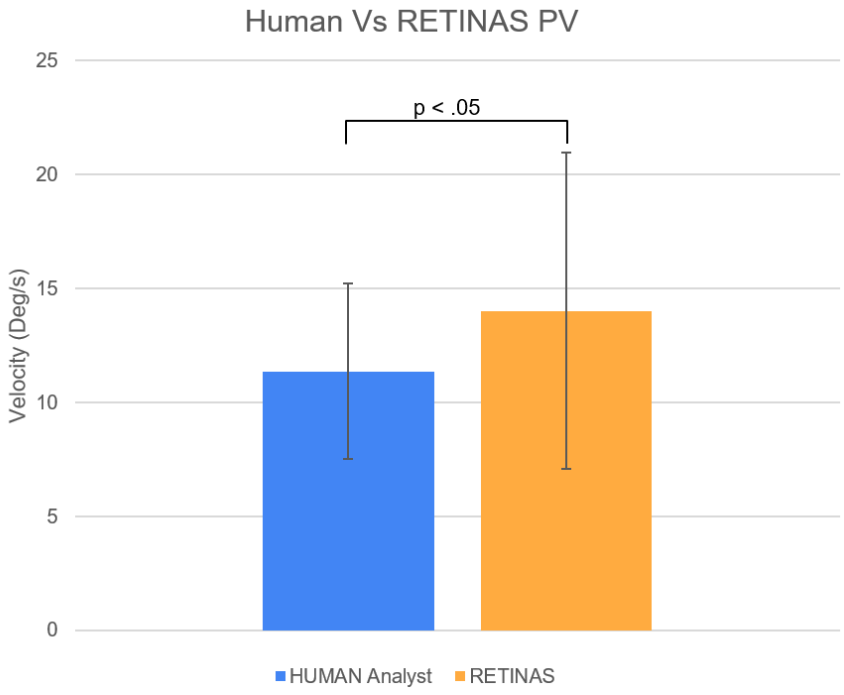

Figure 3.5 Average peak velocity comparison between the data analyzed by human analyst Vs R.E.T.I.N.A.S.. The human analyst achieved a final amplitude of 11.371°/s with a STD of 3.858, whereas R.E.T.I.N.A.S. reported a value of 14.035°/s with a STD of 6.929.

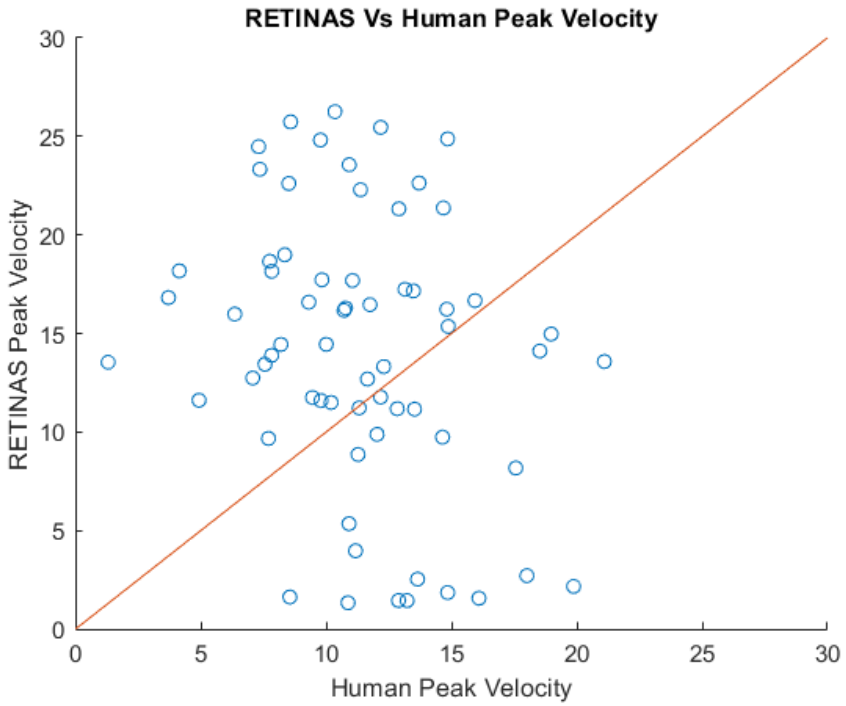

Figure 3.6 Correlation of peak velocity between human analyst and R.E.T.I.N.A.S.. The Pearson correlation value of -.265 and a P-value of .0331.

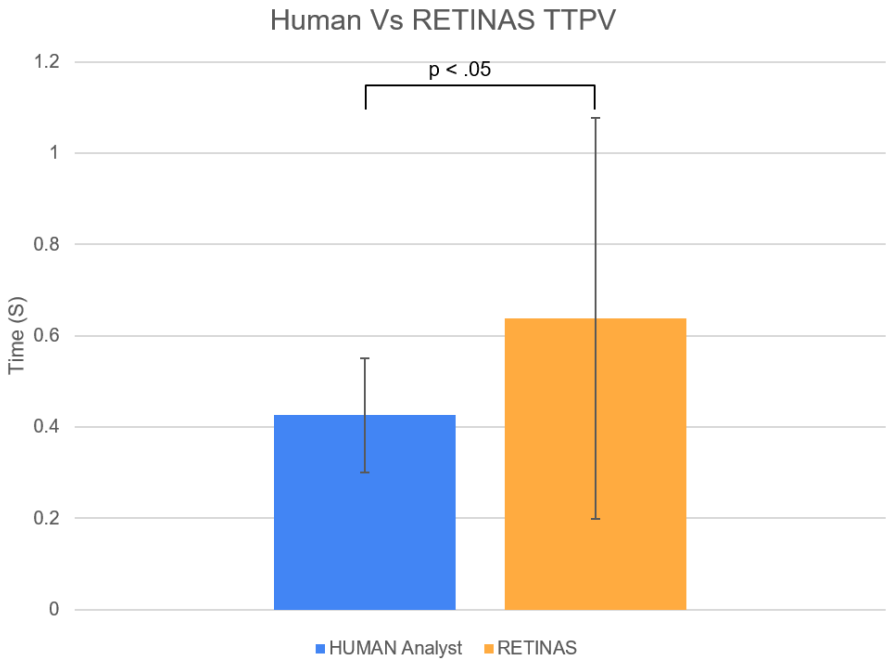

Figure 3.7 Average time to peak velocity comparison between the data analyzed by human analyst Vs R.E.T.I.N.A.S.. The human analyst achieved a final amplitude of .426s with a STD of .125, whereas R.E.T.I.N.A.S. reported a value of .638s with a STD of .439.

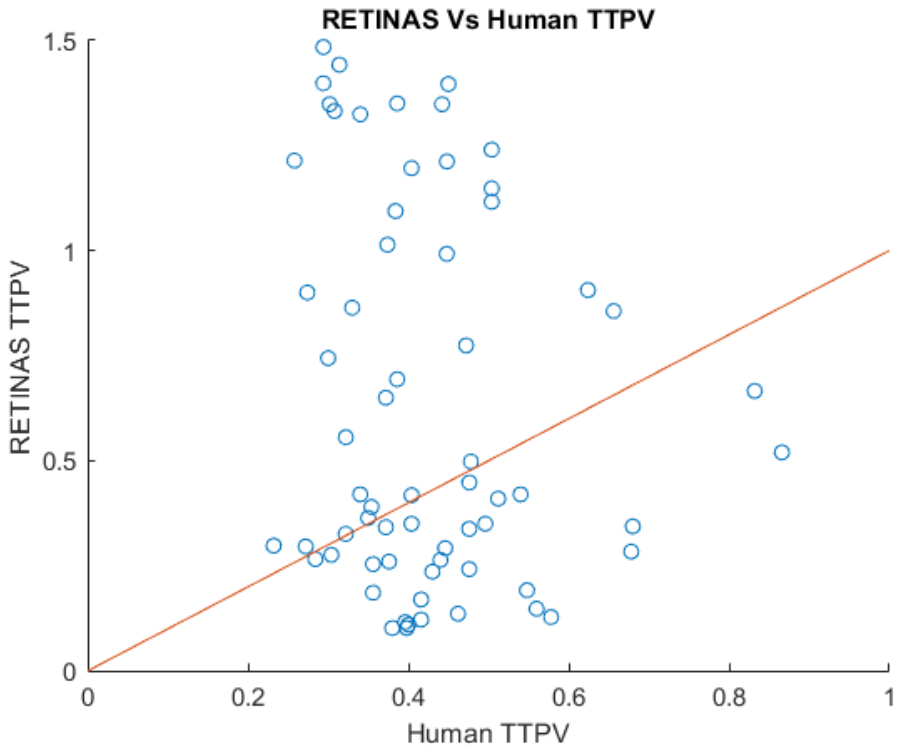

Figure 3.8 Correlation of time to peak velocity between human analyst and R.E.T.I.N.A.S.. The Pearson correlation value of -.151 and a P-value of .223.

3.2 Accuracy of R.E.T.I.N.A.S

The overall accuracy of R.E.T.I.N.A.S was measured using a confusion matrix. The matrix allows the for quick calculation for accuracy, sensitivity, and specificity. Table 3.5 shows the data used to create the confusion matrix. A true positive is considered when the R.E.T.I.N.A.S software and the human analyst classify an eye movement as good, true negative is the inverse when both classify the eye movement as bad. A false negative is when R.E.T.I.N.A.S classify an eye movement as bad, but the human analyst classifies the

data as good, false positive is the opposite when R.E.T.I.N.A.S classifies the eye movement as good, but the analyst classifies it as bad.

Table 3.5 Classification of Eye Movements

Subj Id	True Positive	True Negative	False Negative	False Positive	Total Eye Movements	Accuracy
23389	49	10	10	27	96	0.615
76947	67	5	1	23	96	0.750
41857	33	31	3	29	96	0.667
17530	59	9	1	27	96	0.677
4370	35	45	3	12	96	0.842

Table 3.6 The Confusion Matrix

	Analyst Classified Good	Analyst Classified Bad
R.E.T.I.N.A.S Good Classification	243	118
R.E.T.I.N.A.S Bad Classification	18	100

The average accuracy of R.E.T.I.N.A.S across the 5 subjects was 71.6% with a STD of 7.889%. The specificity of R.E.T.I.N.A.S is 45.9% and the sensitivity of 93.1%. This means that R.E.T.I.N.A.S is strong in identifying true negatives, in other words it is statistically likely to label a bad eye movement as bad.

CHAPTER 4

DISCUSSION AND FUTURE WORKS

4.1 Data Retrieval Methods and Improvements

R.E.T.I.N.A.S. as a whole was unable to retrieve data metrics at the same quality as the human analyst. This can mainly be attributed to the rigid way some of the data metrics are calculated. One such metric is final amplitude which is calculated as the average of the last one second of the eye movement. When artifacts are involved, they can drastically change the final amplitude reported, especially if multiple blinks occur with in a small duration. Artifact removal cannot remove artifacts when they are very close to each other. This is then used to calculate the latency and settling time which can be negatively impacted by it. As seen in **Figure 3.3**, the average final amplitude given by R.E.T.I.N.A.S. was nearly half as much as that of human analyst. This large discrepancy then went on to affect latency which had a $p = .164$, seen in **Figure 3.1.** One way to fix this would be to change how final amplitude is calculated by taking a smaller portion of the steady state to better estimate the final amplitude. Furthermore, the region where final amplitude is gotten can be shifted anywhere within the steady-state portion of the eye movement, a standard while-loop can be used to shift and decrease this region until a final amplitude that falls with in the ranges described previously. Finally, a check for the standard of deviation within the region where final amplitude is retrieved can help determine if a blink is still prevalent even after artifacts are removed.

Another metric that can be largely improved is peak velocity. The approach for this would follow similar steps to how final amplitude should be calculated. A set of while-

loop that are decreasing the region where the peak velocity is calculated until the standard of deviation falls below a certain range. The boundaries of this region can be easily determined by the settling time and latency calculated for the eye movement. This in combination with a while-loop that checks for the amplitude of the peak velocity with boundaries described previously.

4.2 Classification Methods and Improvements

This study provides credence to the fact that an automated eye movement analysis software can classify data at near equal efficacy to that of a human analyst. Furthermore, the R.E.T.I.N.A.S. package provides great artifact removal capabilities which are currently not found in most research settings. There are many avenues for improving R.E.T.I.N.A.S classification as a useful eye movement analysis software. The most important improvement should be focused on heightening true positive classifications. Since there is a large amount of variability in good eye movements the conservative and rigid approach of the software must dynamically change to better understand the behaviors of each subject. This can be implemented by using a more sophisticated classification model that keeps track of all the eye movements of the subject and changes the classification based on that. First a total average for all the data metrics can be calculated across multiple eye movements. Then a using the standard of deviation any eye movements that have 3 out of the 5 data metrics that fall outside of 2 times the standard deviation are classified as “bad”. The “bad” eye movements are then removed from the calculation for the mean and standard of deviation, and the process is repeated. If in the first run no eye movements are removed, then the multiplier for the standard of deviation can be decreased to 1.5 times. This type of

elimination process allows the classification change depending on the data and how the subject performs.

Another approach to refining the classification function is by using a more complicated neural network. Neural networks are series of algorithms that can recognize patterns in between vast amount of data. Creating a neural network might be time consuming upfront but can easily pay off after being trained. The application of neural networks for classification purposes are numerous. For purposes of R.E.T.I.N.A.S. 5 different model would be necessary for each of the data metrics. Then together the models would determine a score for each individual eye movement, if the total score reaches above a certain threshold, then the eye movement can be classified as "good". The threshold can be changed by the user and the weight each model has for the final score can be changed to better fit the importance of certain metrics. For example, final amplitude is necessary for calculation of settling time and latency, by increasing the weight of final amplitude it becomes a greater determinant of a "good" eye movement and the same can be said for peak velocity.

4.3 Future of Automated Data Analysis

As more data becomes computerized the importance of automation of the data analysis increases. More importantly, in areas where data is still analyzed by hand or by semi-automated programs, objectivity becomes another key factor. Increasing is the time demand or monetary cost of analyzing large sets of data by hand. R.E.T.I.N.A.S. is a step in the right direction in try to solve these problems simultaneously. If the correction stated above can be applied to R.E.T.I.N.A.S., then it can greatly impact how data in vision

research is analyzed. It can decrease the undue cost of labor on researchers that can better spend designing more complex studies. It can even decrease the monetary cost associated with hiring multiple data analyst. Furthermore, currently in the research setting data is analyzed by a human analyst which raises the question for subjectivity since each analyst may define certain data metrics differently. Finally, R.E.T.I.N.A.S. can be further developed into a screening tool for convergence insufficiency. This would eliminate the need for an optometrist or hours of analyzing eye movements. This type of quick and easy screening tool can greatly increase the accessibility for patients affected by CI to get necessary therapy, which can greatly improve the patient's quality of life.

APPENDIX A

PYTHON CODE

```
root = Tk()                          # Initialize GUI
root.title("RETINAS")                      # Title the GUI


def stop():                          # Break GUI loop and exit
   root.destroy()

def resize_image(event):
   new_width = int(event.width)
   new_height = int(event.height)
   image = copy_of_image.resize((new_width, new_height))
   photo = ImageTk.PhotoImage(image)
   VNEL_Label.config(image = photo)
   VNEL_Label.image = photo

def dataBWFilter(dataInput):
   o, p = sig.butter(10, .08, 'low')
   filteredData = sig.filtfilt(o,p,dataInput)
   return filteredData

def metrics (position,velocity,badval):
   from scipy.signal import find_peaks
   import numpy as np
   import matplotlib.pyplot as plt
   if badval != 1:
      finarr = position
      o, p = sig.butter(10, .08, 'low')
      finarr_filtered = sig.filtfilt(o,p,finarr) #Filter data
      time= numpy.arange(0., .002*finarr.size, .002) #Create x-axis

      #This function finds the final amplitude
      from statistics import mean
      a = finarr_filtered
      #this average is the average of the last 1/5 of the data Or the final Amplitude
      indices = a[-len(a)//5:]
      average = mean(indices)
      fa = average
      if testcase==2:
         print("The following is the average of the last second of the position trace")
         print("Final Amplitude value:")
         print(average)

      #This function finds the Latency and graphs it
```

```
latencydeg = .05*average

if testcase == 2:
   print("This is the latency degree value:")
   print(latencydeg)

#Find where this .05 value is taken and then find the corresponding time value
#it is important to understand that you can't assume that the latency degree value is plotted within the graph.

array1 = np.argwhere(a>latencydeg)
array2 = np.argwhere(array1>50)

if array2.shape[0] ==0:
   latency=9999
else:
   latency = ((np.argwhere(a>latencydeg)[array2[0]][0])/500)
   latency=round(latency[0],3)
   latpt=(np.argwhere(a>latencydeg)[array2[0]][0])



#print(latency.shape,latency)
#defining where degrees can be high enough
#Defining that it is going to be greater than 50 for now. Estimation.
if testcase==2:
   print("Latency in seconds:")
   print(array1, array2,latency)
   plt.clf()
   plt.plot(time,finarr_filtered,'b') #r stands for red.
   plt.plot(latency,latencydeg,"r*")
   plt.title('Filtered Position Trace Data')
   plt.xlabel('Seconds')
   plt.ylabel('Degrees')
   plt.figure(figsize=(w, h))

#This cell will find the settling time of the position trace data
settledeg=.95*average
#print(settledeg)
if testcase==1:
   print("This is the settling degree value:")
   print(settledeg)

#Find where this .05 value is taken and then find the corresponding time value
#it is important to understand that you can't assume that the latency degree value is plotted within the graph.
#in this example you have .1333 as the latency degree value and you must find the index which
```

```
    # Find the x value corresponding to the maximum y value
    #Because there is nothing exactly equal to the actual latencydeg
    array1=np.argwhere(a>settledeg)
    array2=np.argwhere(array1>50)
    if settledeg>-1:
      settletime = ((np.argwhere(a>settledeg)[array2[0]][0])/500)

      settlept=(np.argwhere(a>settledeg)[array2[0]][0])
      #print(settletime,settlept)
    else:
      settletime=[9999]
      settlept=[9999]

    #defining where degrees can be high enough
    #Defining that it is going to be greater than 50 for now. Estimation.
    if testcase==2:
      print("Settling time in seconds:")
      print(settletime)

      plt.plot(time,finarr_filtered,'b') #r stands for red.
      plt.plot(settletime,settledeg,"r*")
      plt.title('Filtered Position Trace Data')
      plt.xlabel('Seconds')
      plt.ylabel('Degrees')
      plt.figure(figsize=(w, h))

    #The following takes the derivative of the data to get the velocity of the data and plots
    #Refer to https://docs.scipy.org/doc/scipy-
0.16.1/reference/generated/scipy.signal.savgol_filter.html
    #Output is velocity function
    if latency !=9999 and latpt[0]<settlept[0]:
      outputvelocity=sig.savgol_filter(finarr_filtered,15,3,1,delta=.002)
      #outputvelocity.shape=(outputvelocity.size,1)
      #time.shape=(time.size,1)
      #print(outputvelocity.shape,latpt,settlept)
      outputvelocity2=outputvelocity [int(latpt[0]):int(settlept[0])]
      time2=time[int(latpt[0]):int(settlept[0])]
      if testcase==2:
        plt.figure(figsize=(w, h))
        plt.plot(time2,outputvelocity2,'r')
        #15 is window size, 3 is ? ,
        #1 is 1st derivative, delta is time between samples aka sampling rate
        print ("This is the Velocity Graph")
        plt.title('Velocity Trace Graph')
        plt.xlabel('Seconds')
        plt.ylabel('Degrees/Sec')

      #This code will Find the peaks of the data for Position
```

```
        #Refer to https://plotly.com/python/peak-finding/#peak-detection
        #Try runnin previous codes before running this.
        #import plotly.graph_objects as go


        time= np.arange(0., .002*finarr.size, .002)
        indices = find_peaks(time2, threshold=20)[0]
        if testcase==2:
           print("This is the maximum peak value of finarr using the max function")
           print (max(finarr))
        #Finding Peak Velocity and Graphs it https://thispointer.com/find-max-value-its-index-in-numpy-array-numpy-amax/
        #Finding time to Peak velocity.
        #Peak velocity
        max_y = max(outputvelocity2)  # Find the maximum y value
        max_x = time2[outputvelocity2.argmax()]  # Find the x value corresponding to the maximum y value
        pv=max_y
        ttpv=max_x
        if testcase==2:
           print ("The following is the coordinate when peak velocity occurs:")
           print ("[x,y]=",[max_x, max_y])

           #The next step would be to highlight these peaks within the graph.
           #This site helped me:https://stackoverflow.com/questions/41489543/using-matplotlib-how-to-highlight-one-point-in-the-final-plot/41489810

           plt.plot(time,outputvelocity,'r')
           plt.plot(max_x,max_y,"b*")
           plt.title('Velocity Trace Graph')
           plt.xlabel('Seconds')
           plt.ylabel('Degrees/Sec')
           plt.figure(figsize=(w, h))

     else:
        max_x=9999
     latency=latency
     if latency>99: #no good latency
        latency=9999
        pv=9999
        ttpv=9999
        st=9999
        fa=9999
     elif max_x>1.5: #late time to peak velocity
        latency=9999
        pv=9999
        ttpv=9999
        st=9999
```

```
            fa=9999
        elif fa<.5:
            latency=9999
            pv=9999
            ttpv=9999
            st=9999
            fa=9999
        else:
            pv=round(max_y,3)
            ttpv=round(max_x,3)
            st=round(settletime[0],3)
            fa=round(average,3)
    else:
        latency=9999
        pv=9999
        ttpv=9999
        st=9999
        fa=9999
    return (latency,pv,ttpv,st,fa)

def artrem(x):

    issame=1
    badval=0
    time= numpy.arange(0., .002*x.size, .002)

    if testcase==1:
        print(time.shape)
        print(time.size)
        print(x.shape)
        print(x.size)
    xorg=x
    xbeg=x[0:int(x.size/10)]
    xend=x[9*int(x.size/10):x.size]
    if abs(numpy.mean(xbeg)-numpy.mean(xend))>.5:
        if numpy.mean(xbeg)>numpy.mean(xend):
            x=-x
    else:
        xmid=x[2*int(x.size/10):3*int(x.size/10)]
        if numpy.mean(xbeg)>numpy.mean(xmid):
            x=-x
    x=x-x[0]

    # 60hz filter
    x_filtered = dataBWFilter(x)

    if testcase==1:print(x)
    if testcase==1:print(x_filtered)
```

```
x = x_filtered


num=0
siz=x.size
x_orignal=numpy.zeros((siz,1))
x_filtered=numpy.zeros((siz,1))
x_vel=numpy.zeros((siz,1))
x_acc=numpy.zeros((siz,1))

if testcase==1:
   w=6
   h=4
   plt.figure(figsize=(w, h))

val=18
for m in range (15,16,2): # savgol filter inputs m & p can be changed @Farhan
    for p in range (3,4,1):
      #print(m)
      if p<m:
         x_orignal =1*sig.savgol_filter(x,int(m),int(p),0,delta=.002)
         x_filtered[0:siz,num]=x_orignal
         x_vel[0:siz,num]=1*sig.savgol_filter(x,int(m),int(p),1,delta=.002)
         x_acc[0:siz,num]=.002*sig.savgol_filter(x,int(m),int(p),2,delta=.002)

      if testcase==1:
         plt.plot(time,x_orignal,'g',time,x_filtered,'k',time,x_vel,'b',time,x_acc,'r')
         plt.title('window size '+str(m) +' polynomial fit '+ str(p))
         plt.legend(['orig','x', 'xsm', 'v', 'acc'])
         plt.axis((0.002*0,.002*300,-10, 10))
         plt.show()

#part 4 get blinks

acc = x_acc[0:siz,num]
bigacc=numpy.nonzero((acc) > 3) #The acceleration threshold can be changed @Farhan
#print(bigacc)
bigacc=bigacc[0]
if testcase==1:
   print(bigacc,bigacc.size)
if testcase==1:
   if bigacc.size>0:
      w=6
      h=4

      plt.plot(time[bigacc[0]-100:bigacc[-1]+100],x[bigacc[0]-100:bigacc[-
1]+100],'k',time[bigacc[0]:bigacc[-1]],output[bigacc[0]:bigacc[-
```

```
1],num],'b',time[bigacc[0]:bigacc[-1]],output2[bigacc[0]:bigacc[-
1],num],'r',time[bigacc[0]:bigacc[-1]],output3[bigacc[0]:bigacc[-1],num],'m')

#plt.plot(time,output[0:siz,num],'b',time,output2[0:siz,num],'r',time,output3[0:siz,num],'m')
#(has some changes)
        plt.title('zoomed on artifacts')
        plt.legend(['orig','pos savgol', 'vel savgol', 'acc savgol'])
        plt.figure(figsize=(w, h))
        #plt.axis((0,.002*x.size,-50, 50))
    #print(num)


  #hifixinghere
  #part 5 - take out filtered x,v,a
  #take out middle points

  x = x_orignal
  x2 = x_filtered[0:siz,num]
  v2= x_vel[0:siz,num]
  vorg=v2
  a2= x_acc[0:siz,num]
  numberbuffer=20
  (latency,pv,ttpv,st,fa)=metrics(x,vorg,0)
#    print(latency,pv,ttpv,st,fa) #@Farhan



  #part 6 - blink fix -
  #check near
  if testcase==1:print(bigacc)
  if bigacc.size>0:
    breakacc=numpy.zeros(100)
    breakid=1
    if testcase==1:print(bigacc.shape)
    for i in range (1,bigacc.size):
      if abs(bigacc[i]- bigacc[i-1])>30:
        breakacc[breakid]=i
        breakid=breakid+1
    if bigacc.size==1:
      breakacc[breakid]=1
    else:
      breakacc[breakid]=i+1
    if testcase==1:print(breakacc,breakid)
    #print(bigacc[int(breakacc[i-1])],bigacc[int(breakacc[i])-1])

    #interpolate pos

    fixblink1=v2
```

```
    fixblink2=x2

    countsaccades=0
    bufferzonesacc=18 #I can change buffer zones @Farhan
    for i in range (1,breakid+1):
      countsaccades=countsaccades+1
      if testcase==1:print("blink detected ranges",bigacc[int(breakacc[i-
1])],bigacc[int(breakacc[i])-1])
      if bigacc[int(breakacc[i-1])]<=300:
        badval=1
      if bigacc[int(breakacc[i-1])]<=bufferzonesacc:
        startpoint= 1
      else:
        index=int(breakacc[i-1])
        startpoint= bigacc[index] -bufferzonesacc
      if bigacc[int(breakacc[i])-1]>=time.size-bufferzonesacc:
        endpoint=time.size-2
      else:
        index=int(breakacc[i]-1)
        #print(index)
        endpoint= bigacc[index] +bufferzonesacc
      if testcase==1:print("buffered ranges",startpoint,endpoint)
      if bigacc[int(breakacc[i-1])]<bufferzonesacc:
#       print('Test is working')
        if testcase==1:print(fixblink1[0],fixblink2[0])

#fixblink1[0]=numpy.mean(fixblink1[endpoint+bufferzonesacc:endpoint+bufferzonesacc+3])

fixblink2[0]=numpy.mean(fixblink2[endpoint+bufferzonesacc:endpoint+bufferzonesacc+3])
        if testcase==1:print(fixblink1[0],fixblink2[0])
      if bigacc[int(breakacc[i])-1]>time.size-bufferzonesacc:
#       print('Test is working 2')
        if testcase==1:print(fixblink1[-1])
        #fixblink1[-1]=numpy.mean(fixblink1[endpoint-bufferzonesacc-3:endpoint-
bufferzonesacc])
        fixblink2[-1]=numpy.mean(fixblink2[endpoint-bufferzonesacc-3:endpoint-
bufferzonesacc])
        if testcase==1:print(fixvel1[-1])
      tshort=numpy.delete(time,numpy.s_[startpoint:endpoint])
      #remo=numpy.delete (fixblink1,numpy.s_[startpoint:endpoint])
      rem=numpy.delete (fixblink2,numpy.s_[startpoint:endpoint])

      #f1blv = interp1d(tshort, remo,kind='slinear')
      f1sblp = interp1d(tshort, rem,kind='slinear')
      fixblink2=f1sblp(time)

      if testcase==1:
        print(tshort.shape,tshort.size,startpoint,endpoint)
```

```
            print("artremerr",f1sblp,rem.shape,rem.size)
            print("artremerr2",fixblink2.shape)
            #fixblink1=f1blv(time)

        #if testcase==1:print(breakacc,breakacc)

        #integrate velocity
        #yint=integrate.cumtrapz(fixblink1,time,initial=0)


        tshort=numpy.delete(time,numpy.s_[bigacc[0]-numberbuffer:bigacc[-1]+numberbuffer])
        rem=numpy.delete(x2,numpy.s_[bigacc[0]-numberbuffer:bigacc[-1]+numberbuffer])

        f = interp1d(tshort, rem,kind='slinear')
        f2=interp1d(tshort, rem, kind='cubic')
        f3=interp1d(tshort, rem, kind='nearest')
        f4=interp1d(tshort, rem, kind='zero')
        f5=interp1d(tshort, rem, kind='slinear')
        f6=interp1d(tshort, rem, kind='quadratic')
        f7=interp1d(tshort, rem, kind='previous')
        f8=interp1d(tshort, rem, kind='next')
#                                                                                                   fig1            =
plt.plot(time,x,'k',time,f(time),time,f1sblp(time))#,time,f2(time),time,f5(time),time,f6(time),line
width=2)
#       plt.legend(['orig','slinear', 'cubic'])

        if testcase == 1:

            w=6
            h=4
            plt.figure(figsize=(w, h))
            plt.plot(time,x,'k',time,f(time),time,f2(time),time,f5(time),time,f6(time),linewidth=2)
            plt.plot(time, fixblink2,'m')
            #plt.plot(time, fixblink2,'m',time,yint,'c')
            #plt.axis((0,.002*x.size,-1, 1))
            plt.title('After fixing blinks: position')
            plt.legend(['orig','pos interp', 'vel interp'])
            xmin, xmax, ymin, ymax = plt.axis()
            plt.axis((0,.002*x.size,ymin, ymax))

        x3=fixblink2
        v3f1=1*sig.savgol_filter(x3,int(m),int(p),1,delta=.002)
        a3=.002*sig.savgol_filter(x3,int(m),int(p),2,delta=.002)
        v3=v3f1
    else:
        x3=x2
        v3f1=v2
        v3=v3f1
```

```
    a3=a2


  #part 7 plot velocities after blinks
  #velocity after blinks
  #v2 is before blinks
  #v3 is after

  if testcase==1:
     print(x3)
     w=6
     h=4
     plt.figure(figsize=(w, h))

     plt.plot(time,v2,'k',time, v3f1,'r')#time, v3f2,'r',time, v3f5,'b',time, v3f6,'g')
     plt.plot(time,a3,'b')
     plt.plot(time,x3,'m')#time, v3f2,'r',time, v3f5,'b',time, v3f6,'g')
     plt.legend(['orig vel','after:vel', 'after: acc', 'pos', 'quadratic'])
     #plt.axis((0,.002*x.size,-5, 5))
     plt.title('After fixing blinks: velocity/acc')
     xmin, xmax, ymin, ymax = plt.axis()
     plt.axis((0,.002*500,ymin, ymax))



 #part 8 find saccades, the thresholds for bigvel1 & 2 can be changed
  bigvel1=numpy.nonzero(abs(a3) > 1.6) #Accelaration
  bigvel2=numpy.nonzero(abs(v3) > 15) #Velocity

  bigvel1=bigvel1[0]
  bigvel2=bigvel2[0]
  bigvel=numpy.intersect1d(bigvel1, bigvel2)

  if bigvel.size>0:
     if testcase==1:
        print(bigvel1)
        #print(bigvel[0])
        print(bigvel2)
        print(bigvel)

        w=6
        h=4
        #plt.figure(figsize=(w, h))
        #plt.plot(time[bigacc[0]-100:bigacc[-1]+100],x[bigacc[0]-100:bigacc[-
1]+100],'k',time[bigacc[0]:bigacc[-1]],output[bigacc[0]:bigacc[-
1],num],'b',time[bigacc[0]:bigacc[-1]],output2[bigacc[0]:bigacc[-
1],num],'r',time[bigacc[0]:bigacc[-1]],output3[bigacc[0]:bigacc[-1],num],'m')
```

```
#plt.plot(time,output[0:siz,num],'b',time,output2[0:siz,num],'r',time,output3[0:siz,num],'m')
        #plt.title('window size '+str(m) +' polynomial fit '+ str(p))



   #print(num)

   #part 9 break down locations of saccades
   #print(bigvel.size)
   if bigvel.size>0:
      breakvel=numpy.zeros(100)
      breakid=1
      for i in range (1,bigvel.size):
         if abs(bigvel[i]- bigvel[i-1])>1:
            breakvel[breakid]=i
            breakid=breakid+1
         breakvel[breakid]=i+1
      if testcase==1:print(breakvel)
      if testcase>2:
         #print(x3)
         w=6
         h=4
         plt.figure(figsize=(w, h))

         plt.plot(time,v2,'k',time, v3f1,'r')#time, v3f2,'r',time, v3f5,'b',time, v3f6,'g')
         plt.plot(time,a3,'b')#time, v3f2,'r',time, v3f5,'b',time, v3f6,'g')
         plt.legend(['orig vel','after:vel', 'after: acc', 'slinear', 'quadratic'])
         plt.axis((385*.002,425*.002,-50, 50))
         plt.title('zoom 1After fixing blinks: velocity/acc')
         plt.axis((0*.002,1500*.002,-50, 50))
         xmin, xmax, ymin, ymax = plt.axis()
         #plt.axis((0,.002*x.size,ymin, ymax))


   #part 10 - fix saccades
   if bigvel.size>0:
      fixvel1=v3f1
      fixpos1=x3
      if testcase==1:print(bigvel)

      countsaccades=0
      bufferzonesacc=16 #I can change buffer @Farhan
      for i in range (1,breakid+1):
         countsaccades=countsaccades+1
         if testcase==1:print("sacc detected ranges",bigvel[int(breakvel[i-
1])],bigvel[int(breakvel[i])-1])
         if bigvel[int(breakvel[i-1])]<=bufferzonesacc:
```

```
            startpoint= 1
        else:
            index=int(breakvel[i-1])
            startpoint= bigvel[index] -bufferzonesacc
        if bigvel[int(breakvel[i])-1]>=time.size-bufferzonesacc:
            endpoint=time.size-2
        else:
            index=int(breakvel[i]-1)
            #print(index)
            endpoint= bigvel[index] +bufferzonesacc
        if testcase==1:print(startpoint,endpoint)
        if bigvel[int(breakvel[i-1])]<bufferzonesacc:
            if testcase==1:print(fixvel1[0],fixpos1[0])

#fixvel1[0]=numpy.mean(fixvel1[endpoint+bufferzonesacc:endpoint+bufferzonesacc+3])

fixpos1[0]=numpy.mean(fixpos1[endpoint+bufferzonesacc:endpoint+bufferzonesacc+3])

#fixpos2[0]=numpy.mean(fixpos2[endpoint+bufferzonesacc:endpoint+bufferzonesacc+3])

#fixpos5[0]=numpy.mean(fixpos5[endpoint+bufferzonesacc:endpoint+bufferzonesacc+3])

#fixpos6[0]=numpy.mean(fixpos6[endpoint+bufferzonesacc:endpoint+bufferzonesacc+3])
            if testcase==1:print(fixvel1[0],fixpos1[0])
        if bigvel[int(breakvel[i])-1]>time.size-bufferzonesacc:
            if testcase==1:print(fixvel1[-1])
            #fixvel1[-
1]=numpy.mean(fixvel1[endpoint+bufferzonesacc:endpoint+bufferzonesacc+3])
            fixpos1[-1]=numpy.mean(fixpos1[endpoint-bufferzonesacc-3:endpoint-
bufferzonesacc])
            #fixpos2[-
1]=numpy.mean(fixpos2[endpoint+bufferzonesacc:endpoint+bufferzonesacc+3])
            #fixpos5[-
1]=numpy.mean(fixpos5[endpoint+bufferzonesacc:endpoint+bufferzonesacc+3])
            #fixpos6[-
1]=numpy.mean(fixpos6[endpoint+bufferzonesacc:endpoint+bufferzonesacc+3])
            if testcase==1:print(fixvel1[-1])
        tshort=numpy.delete(time,numpy.s_[startpoint:endpoint])
        #remo=numpy.delete (fixvel1,numpy.s_[startpoint:endpoint])
        rem=numpy.delete (fixpos1,numpy.s_[startpoint:endpoint])

        #f1saccv = interp1d(tshort, remo,kind='slinear')
        f1saccp = interp1d(tshort, rem,kind='slinear')
        fixpos1=f1saccp(time)
        #fixvel1=f1saccv(time)
    #yint=integrate.cumtrapz(fixvel1,time,initial=0)

    if testcase==1:
```

```
        w=6
        h=4

        plt.plot(time, x,'k')
        plt.plot(time, fixpos1,'r')
        #plt.plot(time, yint,'g')
        plt.legend(['orig pos','after: interp pos', 'after: interp vel', 'slinear', 'quadratic'])
        plt.axis((0,x.size*.002,-6, 6))
        plt.title('zoom 1After fixing blinks: velocity/acc')
        plt.figure(figsize=(w, h))
        xmin, xmax, ymin, ymax = plt.axis()
        #plt.axis((0,.002*x.size,ymin, ymax))
        print(breakvel)

        plt.show
        #plt.legend()
    else:
      #fixvel1=v3
      fixpos1=x3

#    plt.plot(time, x,'k')
#    plt.plot(time, fixpos1,'r')

    #print(fixpos1.shape,fixpos1)
    x3 = fixpos1;
    v3 = sig.savgol_filter(x3,int(m),int(p),1,delta=.002)
    a3 = .002*sig.savgol_filter(x,int(m),int(p),2,delta=.002)

    ##inserting start
    bigvel1=numpy.nonzero(abs(a3) > 1.6)
    bigvel2=numpy.nonzero(abs(v3) > 15)

    bigvel1=bigvel1[0]
    bigvel2=bigvel2[0]
    bigvel=numpy.intersect1d(bigvel1, bigvel2)
#                                                                                   print("ROUND
333333333333333333333333333333333333333333333333333333333333333333333333333333")
    if bigvel.size>0:
      if testcase==1:
        print(bigvel1)
        #print(bigvel[0])
        print(bigvel2)
        print(bigvel)

        w=6
        h=4
        #plt.figure(figsize=(w, h))
```

```
        #plt.plot(time[bigacc[0]-100:bigacc[-1]+100],x[bigacc[0]-100:bigacc[-
1]+100],'k',time[bigacc[0]:bigacc[-1]],output[bigacc[0]:bigacc[-
1],num],'b',time[bigacc[0]:bigacc[-1]],output2[bigacc[0]:bigacc[-
1],num],'r',time[bigacc[0]:bigacc[-1]],output3[bigacc[0]:bigacc[-1],num],'m')

#plt.plot(time,output[0:siz,num],'b',time,output2[0:siz,num],'r',time,output3[0:siz,num],'m')
        #plt.title('window size '+str(m) +' polynomial fit '+ str(p))



   #print(num)

   #part 9 break down locations of saccades
   #print(bigvel.size)
   if bigvel.size>0:
      breakvel=numpy.zeros(100)
      breakid=1
      for i in range (1,bigvel.size):
         if abs(bigvel[i]- bigvel[i-1])>1:
            breakvel[breakid]=i
            breakid=breakid+1
         breakvel[breakid]=i+1
      if testcase==1:print(breakvel)
      if testcase>2:
         #print(x3)
         w=6
         h=4
         plt.figure(figsize=(w, h))

         plt.plot(time,v2,'k',time, v3f1,'r')#time, v3f2,'r',time, v3f5,'b',time, v3f6,'g')
         plt.plot(time,a3,'b')#time, v3f2,'r',time, v3f5,'b',time, v3f6,'g')
         plt.legend(['orig vel','after:vel', 'after: acc', 'slinear', 'quadratic'])
         plt.axis((385*.002,425*.002,-50, 50))
         plt.title('zoom 1After fixing blinks: velocity/acc')
         plt.axis((0*.002,1500*.002,-50, 50))
         xmin, xmax, ymin, ymax = plt.axis()
         #plt.axis((0,.002*x.size,ymin, ymax))


   #part 10 - fix saccades
   if bigvel.size>0:
      fixvel1=v3f1
      fixpos1=x3
      if testcase==1:print(bigvel)

      countsaccades=0
      bufferzonesacc=16 #15
      for i in range (1,breakid+1):
```

```
        countsaccades=countsaccades+1
        if testcase==1:print("sacc detected ranges",bigvel[int(breakvel[i-
1])],bigvel[int(breakvel[i])-1])
        if bigvel[int(breakvel[i-1])]<=bufferzonesacc:
           startpoint= 1
        else:
           index=int(breakvel[i-1])
           startpoint= bigvel[index] -bufferzonesacc
        if bigvel[int(breakvel[i])-1]>=time.size-bufferzonesacc:
           endpoint=time.size-2
        else:
           index=int(breakvel[i]-1)
           #print(index)
           endpoint= bigvel[index] +bufferzonesacc
        if testcase==1:print(startpoint,endpoint)
        if bigvel[int(breakvel[i-1])]<bufferzonesacc:
           if testcase==1:print(fixvel1[0],fixpos1[0])

#fixvel1[0]=numpy.mean(fixvel1[endpoint+bufferzonesacc:endpoint+bufferzonesacc+3])

fixpos1[0]=numpy.mean(fixpos1[endpoint+bufferzonesacc:endpoint+bufferzonesacc+3])

#fixpos2[0]=numpy.mean(fixpos2[endpoint+bufferzonesacc:endpoint+bufferzonesacc+3])

#fixpos5[0]=numpy.mean(fixpos5[endpoint+bufferzonesacc:endpoint+bufferzonesacc+3])

#fixpos6[0]=numpy.mean(fixpos6[endpoint+bufferzonesacc:endpoint+bufferzonesacc+3])
           if testcase==1:print(fixvel1[0],fixpos1[0])
        if bigvel[int(breakvel[i])-1]>time.size-bufferzonesacc:
           if testcase==1:print(fixvel1[-1])
           #fixvel1[-
1]=numpy.mean(fixvel1[endpoint+bufferzonesacc:endpoint+bufferzonesacc+3])
           fixpos1[-1]=numpy.mean(fixpos1[endpoint-bufferzonesacc-3:endpoint-
bufferzonesacc])
           #fixpos2[-
1]=numpy.mean(fixpos2[endpoint+bufferzonesacc:endpoint+bufferzonesacc+3])
           #fixpos5[-
1]=numpy.mean(fixpos5[endpoint+bufferzonesacc:endpoint+bufferzonesacc+3])
           #fixpos6[-
1]=numpy.mean(fixpos6[endpoint+bufferzonesacc:endpoint+bufferzonesacc+3])
           if testcase==1:print(fixvel1[-1])
        tshort=numpy.delete(time,numpy.s_[startpoint:endpoint])
        #remo=numpy.delete (fixvel1,numpy.s_[startpoint:endpoint])
        rem=numpy.delete (fixpos1,numpy.s_[startpoint:endpoint])

        #f1saccv = interp1d(tshort, remo,kind='slinear')
        f1saccp = interp1d(tshort, rem,kind='slinear')
        fixpos1=f1saccp(time)
```

```
      #fixvel1=f1saccv(time)
   #yint=integrate.cumtrapz(fixvel1,time,initial=0)
   if testcase==1:
      w=6
      h=4

      plt.plot(time, x,'k')
      plt.plot(time, fixpos1,'r')
      #plt.plot(time, yint,'g')
      plt.legend(['orig pos','after: interp pos', 'after: interp vel', 'slinear', 'quadratic'])
      plt.axis((0,x.size*.002,-6, 6))
      plt.title('zoom 1After fixing blinks: velocity/acc')
      plt.figure(figsize=(w, h))
      xmin, xmax, ymin, ymax = plt.axis()
      #plt.axis((0,.002*x.size,ymin, ymax))
      print(breakvel)

      plt.show
      #plt.legend()
else:
   #fixvel1=v3
   fixpos1=x3

#print(fixpos1.shape,fixpos1)


##inserting end

   #part 11
#integration for velocity
aftersaccadevel=1*sig.savgol_filter(fixpos1,int(m),int(p),1,delta=.002)
aftersaccadeacc=1*sig.savgol_filter(fixpos1,int(m),int(p),2,delta=.002)
if testcase==1:
   plt.figure(figsize=(w, h))
   plt.plot(time,v2,'k',time,aftersaccadevel ,'r')
   plt.legend(['orig vel','vel with pos interp', 'vel with vel interp integ'])
   plt.axis((0,.002*x.size,-30, 30))
   plt.title('After fixing saccades: velocity/acc')


for element in range (0,len(xorg))  :
   if abs(xorg[element]-fixpos1[element])>.01:
      issame=0

if issame==1:
   badval=-1

#part 12 final output
```

```
    finaloutputx=fixpos1
    finaloutputv=aftersaccadevel
    finaloutputa=aftersaccadeacc
    w=6
    h=4
    if testcase==1:
        print(finaloutputx,finaloutputv)
        plt.figure(figsize=(w, h))
        plt.plot(time,x,'k',time,finaloutputx ,'g')
        plt.plot(time,finaloutputv,'b',time, finaloutputv,'c')
        plt.plot(time,.002*a2,'r',time, .002*finaloutputa,'m')
        plt.legend(['orig pos','after pos', 'orig v', 'after v','orig a','after a'])
        plt.axis((0,.002*x.size,-5, 15))
        plt.title('After fixing saccades: velocity/acc')

    return(finaloutputx,finaloutputv,finaloutputa,badval)

def loaddata():                         # Open file dialog and setup initial directory
    t = time.time()
    global  protodata, filepath, filename, matdata, caldata

    root.choosefile = filedialog.askopenfilename(initialdir = r"C:\Users\Farhan
Ahmad\Dropbox\DoD_NewMexico\Analysis\VEMAP_DualMode_Farhan\Data\Preprocessed")
    filepath = root.choosefile                  # Get file path from loaddata function
    filename = os.path.split(filepath)          # Split path into 2, file name and directory
    os.chdir(filename[0])                       # Set working directory using filepath
    mat = loadmat(filename[1])                  # load selected matlab file
    matdata  = mat['RawData']                   # Select RawData array which contains the raw data
    caldata  = mat['CalData']                   # Select CalDaata array which contains the calibration
data
    protocolpath = filename[0].strip('Data/Preprocessed') # strip file path to general directory
    protocolpath = protocolpath+"/Protocols/TRANSITIONS_DUALMODE.protocol.mat" # add
directions to TRANSITIONS_DUALMODE protocol file
    protocol = loadmat(protocolpath)            # load TRANSITIONS_DUALMODE.protocol.mat
    protodata = protocol['protocol'][0, 0]      # Select protocol array and select first cell which
holds all the index values



def analyze():
    global testcase
    excelpath = filename[0].strip("/Preprocessed")+"/RETINAS/"+filename[1].strip(".mat")+".xlsx"
    writer = pandas.ExcelWriter(excelpath, engine='xlsxwriter')
    testcase = 0
    k = 0
    for  mvmttype in numpy.arange(0,len(eye_movments)):
        for mvmtlen in numpy.arange(0,len(eye_movments[mvmttype])):
            Latency = []
```

```
        Peak_Velocity = []
        TimetoPV = []
        SettlingT = []
        Final_AMP = []
        Classification = []
        for i in numpy.arange(0,len(eye_movments[mvmttype][mvmtlen])):
          print(mvmttype,mvmtlen,i)
          if mvmttype == "SACCADES":
            try:
              vergence = eye_movments[mvmttype][mvmtlen][i][0]+eye_movments[mvmttype][mvmtlen][i][1]
              a = artrem(vergence)
            except(RuntimeError, TypeError, NameError, ValueError, LinAlgError):
              velocity = 1*sig.savgol_filter(vergence,15,3,1,delta=.002)
              a = [vergence,velocity,1]
              continue
          else:
            try:
              vergence = eye_movments[mvmttype][mvmtlen][i][0]-eye_movments[mvmttype][mvmtlen][i][1]
              a = artrem(vergence)
            except(RuntimeError, TypeError, NameError, ValueError, LinAlgError):
              velocity = 1*sig.savgol_filter(vergence,15,3,1,delta=.002)
              a = [vergence,velocity,1]
              continue
          (latency,pv,ttpv,st,fa) = metrics(a[0], a[1], 0)
          Latency.append(latency)
          Peak_Velocity.append(pv)
          TimetoPV.append(ttpv)
          SettlingT.append(st)
          Final_AMP.append(fa)
          Classification.append(a[3])
        RETINAS_OUTPUT = {'Latency':Latency,'Peak Velocity':Peak_Velocity,'Time to PV':TimetoPV,'Settling Time':SettlingT, 'Final Amplitude':Final_AMP, 'RETINAS Classification': Classification}
        df = pandas.DataFrame(RETINAS_OUTPUT)
        Sheet_Name = str(mvmttype)+"_"+str(mvmtlen)+"_"+str(i)
        df.to_excel(writer, sheet_name = Sheet_Name)
    writer.save()

def preprocess():                       # Find gain values and load the subject data into memory
    global eye_movments, Cal_option, testcase
    print(Cal_option.get())
    # Cal_option can be "NORMAL", "SACCADES" or "AUTOMATIC"
    GAINS_FOR_TESTING,RE_ALL_Gains,LE_ALL_Gains = ([] for i in range(3))
    test_arr = numpy.zeros((3,2))
    testcase = 0
    x = numpy.arange(1,6,2)
```

```
    y = numpy.zeros((3,2))
    Gain_values = numpy.zeros((3,2))
    Best_Gains = numpy.zeros((3,2))

    k = 0
    if Cal_option.get() == "NORMAL":
        for i in [0,6,12]:

            RE_avg_1deg = numpy.mean(caldata[0][i+3][0])
            RE_avg_3deg = numpy.mean(caldata[0][i+4][0])
            RE_avg_5deg = numpy.mean(caldata[0][i+5][0])

            LE_avg_1deg = numpy.mean(caldata[0][i+0][1])
            LE_avg_3deg = numpy.mean(caldata[0][i+1][1])
            LE_avg_5deg = numpy.mean(caldata[0][i+2][1])

            RE_avg = numpy.array([RE_avg_1deg,RE_avg_3deg, RE_avg_5deg])
            LE_avg = numpy.array([LE_avg_1deg,LE_avg_3deg, LE_avg_5deg])


            RE_slope, RE_intercept, RE_r_value, RE_p_value, RE_std_err = stats.linregress(x,RE_avg)
            LE_slope, LE_intercept, LE_r_value, LE_p_value, LE_std_err = stats.linregress(x,LE_avg)

            Right_Rvalue_string = "Right R^2 vlaue is:" +" "+ str(round(RE_r_value**2,3))
            Left_Rvalue_string = "Left R^2 vlaue is:" +" "+ str(round(LE_r_value**2,3))

            y[k,0] = LE_slope
            y[k,1] = RE_slope

#               print("Left Eye slope: %f, intercept: %f and R^2: %f" %(1/LE_slope, LE_intercept,
LE_r_value**2))
#              print("Right Eye slope: %f, intercept: %f and R^2: %f" %(1/RE_slope, RE_intercept,
RE_r_value**2))

            if .9*y[k,1] > y[k,0] > 1.1*y[k,1] and .9*y[k,0] > y[k,1] > 1.1*y[k,0]:
                Gain_values[k,0] = 1/RE_slope
                Gain_values[k,1] = 1/LE_slope

            elif LE_r_value**2 and RE_r_value**2 > .9:
                Gain_values[k,0] = 1/RE_slope
                Gain_values[k,1] = 1/LE_slope
            k= k+1

        if numpy.all(Gain_values == test_arr) == 1:
            Gain_values[0,0] = 9999
            Gain_values[0,1] = -9999
        else:
            sign = numpy.sign(Gain_values)
```

```
        Gain_values[0,0] = numpy.max((abs(Gain_values[:,0])))
        Gain_values[0,1] = numpy.max((abs(Gain_values[:,1])))

        Gain_values = Gain_values*sign
        Gain_values = Gain_values[~(Gain_values==0).any(1), :]

    if Cal_option.get() == "SACCADES":

        RE_sacades_10 = numpy.mean(caldata[0][18][0])
        RE_sacades_0 = numpy.mean(caldata[0][19][0])
        RE_sacades_neg10 = numpy.mean(caldata[0][20][0])

        LE_sacades_10 = numpy.mean(caldata[0][21][0])
        LE_sacades_0 = numpy.mean(caldata[0][22][0])
        LE_sacades_neg10 = numpy.mean(caldata[0][23][0])

        LE_saccades = numpy.array([LE_sacades_10,LE_sacades_0,LE_sacades_neg10])
        RE_saccades = numpy.array([RE_sacades_10,RE_sacades_0,RE_sacades_neg10])

        RE_Sslope, RE_Sintercept, RE_r_Svalue, RE_p_Svalue, RE_std_Serr =
stats.linregress(x,RE_saccades)
        LE_Sslope, LE_Sintercept, LE_r_Svalue, LE_p_Svalue, LE_std_Serr =
stats.linregress(x,LE_saccades)

        Right_Rvalue_string = "Right R^2 vlaue is:" +" "+ str(round(RE_r_Svalue**2,3))
        Left_Rvalue_string = "Left R^2 vlaue is:" +" "+ str(round(LE_r_Svalue**2,3))

        y[0,0] = RE_Sslope
        y[0,1] = LE_Sslope


#               print("Left Eye slope: %f, intercept: %f and R^2: %f" %(LE_Sslope, LE_Sintercept,
LE_r_Svalue**2))
#               print("Right Eye slope: %f, intercept: %f and R^2: %f" %(RE_Sslope, RE_Sintercept,
RE_r_Svalue**2))

        if .9*y[0,1] > y[0,0] > 1.1*y[0,1] and .9*y[0,0] > y[0,1] > 1.1*y[0,0]:
           Gain_values[0,0] = RE_Sslope
           Gain_values[0,1] = LE_Sslope

        elif LE_r_Svalue**2 and RE_r_Svalue**2 > .9:
           Gain_values[0,0] = RE_Sslope
           Gain_values[0,1] = LE_Sslope

        if numpy.all(Gain_values == test_arr) == 1:
           Gain_values[0,0] = 9999
           Gain_values[0,1] = -9999
```

```
if Cal_option.get() == "AUTOMATIC":
    RE_gain = 1
    LE_gain = 1
    R05_seq = protodata[2][0][0][0]
    L05_seq = protodata[2][0][0][2]
    RE_gain_arr = []
    LE_gain_arr = []
    fa = 0
    pv = 0
    badval = 0
    for seq in [R05_seq[0],L05_seq[0]]:
        for i in seq:
            print(i)
            data = matdata[0][i-1]
            fa = 0
            pv = 0
            RE_gain = 1
            LE_gain = 1
            while RE_gain and LE_gain < 2.5:
                RE_gain = RE_gain +.01
                LE_gain = LE_gain +.01

                right_eye = data[0]
                left_eye = data[1]

                RE_filteredData = dataBWFilter(right_eye)
                LE_filteredData = dataBWFilter(left_eye)

                RE_Offset = numpy.mean(RE_filteredData[0:20])
                LE_Offset = numpy.mean(LE_filteredData[0:20])

                RE_OffsetData = RE_filteredData - RE_Offset
                LE_OffsetData = LE_filteredData - LE_Offset

                RE_CalibratedData = numpy.dot(LE_OffsetData, RE_gain)*1
                LE_CalibratedData = numpy.dot(RE_OffsetData, LE_gain)*1

                vergence = RE_CalibratedData - LE_CalibratedData

                try:
                    position = artrem(vergence)
                    (latency,pv,ttpv,st,fa) = metrics (position[0],position[1],0)
                    transient = abs(numpy.mean(vergence[-round(len(vergence)/2):]))
                    if abs(fa) == 9999 and abs(pv) == 9999:
                        fa = 0
                        pv = 0
                    if transient > 5 or pv > 50:
                        break
```

```
            except(RuntimeError, TypeError, NameError, ValueError, LinAlgError):
                RE_gain = numpy.nan
                LE_gain = numpy.nan
                break

        RE_gain_arr.append(RE_gain)
        LE_gain_arr.append(LE_gain)
    k = 0
    for i in numpy.arange(0,len(RE_gain_arr)-1):
        if RE_gain_arr[i-k] and LE_gain_arr[i-k] > 1.1:
            RE_gain_arr[i-k] = RE_gain_arr[i-k]
            LE_gain_arr[i-k] = LE_gain_arr[i-k]
        else:
            RE_gain_arr.remove(RE_gain_arr[i-k])
            LE_gain_arr.remove(LE_gain_arr[i-k])
            k = k + 1

    if len(RE_gain_arr) == 0:
        Gain_values[0,0] = 9999
        Gain_values[0,1] = -9999
    else:
        Gain_values[0,0] = numpy.nanmean(RE_gain_arr)
        Gain_values[0,1] = numpy.nanmean(LE_gain_arr)

    Right_Rvalue_string = "Right R^2 vlaue is:" +" "+ "NAN"
    Left_Rvalue_string = "Left R^2 vlaue is:" +" "+ "NAN"

sign = numpy.sign(Gain_values)
sign[0,0] = 1
sign[0,1] = -1
Best_Gains[0,0] = numpy.max((abs(Gain_values[:,0])))
Best_Gains[0,1] = numpy.max((abs(Gain_values[:,1])))

Best_Gains = Best_Gains*sign
Best_Gains = Best_Gains[~(Best_Gains==0).any(1), :]
Gain_values = Best_Gains

Right_Gain_string = "Right Gain vlaue is:" +" "+ str(round(Gain_values[0][0],3))

Left_Gain_string = "Left Gain vlaue is:" +" "+ str(round(Gain_values[0][1],3))

mylist.insert(END, Right_Gain_string)
mylist.insert(END, Right_Rvalue_string)
mylist.insert(END, Left_Gain_string)
mylist.insert(END, Left_Rvalue_string)

#    Create empty lists contaning all the different eye movments
t = time.time()
```

```
  CON_04_08, CON_02_06, CON_02_08, DIV_08_04, DIV_06_02, DIV_08_02,
CON_STEP_2to6_RAMP_6to8, DIV_STEP_8to4_RAMP_4to2 = ([] for i in range(8))
  CON_08_12, CON_06_10, CON_06_12, DIV_12_08, DIV_10_06, DIV_12_06,
CON_STEP_6to10_RAMP_10to12, DIV_STEP_12to8_RAMP_8to6 = ([] for i in range(8))
  R05, R10, L05, L10 = ([] for i in range(4))
  FAR = [CON_04_08, CON_02_06, CON_02_08, DIV_08_04, DIV_06_02, DIV_08_02,
CON_STEP_2to6_RAMP_6to8, DIV_STEP_8to4_RAMP_4to2]
  NEAR = [CON_08_12, CON_06_10, CON_06_12, DIV_12_08, DIV_10_06, DIV_12_06,
CON_STEP_6to10_RAMP_10to12, DIV_STEP_12to8_RAMP_8to6]
  SACCADES = [R05, R10, L05, L10]
  eye_movments = [FAR, NEAR, SACCADES]

  l = 0
  for mvmttype in ["FAR","NEAR","SACCADES"]:
    mvmtlen = protodata[mvmttype][0][0].__len__()
    j = 0
    for k in numpy.arange(0,mvmtlen):
      sequnumber = protodata[mvmttype][0][0][k]-1
      for i in numpy.arange(0,sequnumber.shape[1]):
        data = matdata[0][sequnumber[0][i]]
        right_eye = data[0]
        left_eye = data[1]

        RE_filteredData = dataBWFilter(right_eye)
        LE_filteredData = dataBWFilter(left_eye)

        LE_Offset = numpy.mean(LE_filteredData[0:20])
        RE_Offset = numpy.mean(RE_filteredData[0:20])

        LE_OffsetData = LE_filteredData - LE_Offset
        RE_OffsetData = RE_filteredData - RE_Offset

        RE_CalibratedData = numpy.dot(LE_OffsetData,(1/Gain_values[0][0]))*1
        LE_CalibratedData = numpy.dot(RE_OffsetData,(1/Gain_values[0][1]))*1

        eyemvmt = [RE_CalibratedData,LE_CalibratedData]

        if i == sequnumber[0][0]:
          eye_movments[l][j].append(eyemvmt)
        elif data.shape[1] != 2000:
          eyemvmt = [RE_CalibratedData[0:2000],LE_CalibratedData[0:2000]]
          eye_movments[l][j].append(eyemvmt)
          continue
        else:
          eye_movments[l][j].append(eyemvmt)
      j = j + 1
    l = l + 1
```

```
    t = time.time() - t
    print(t)

def openNewWindow():

    # Toplevel object which will
    # be treated as a new window
    plotWindow = Toplevel(root)

    # sets the title of the
    # Toplevel widget
    plotWindow.title("Plot Window")

    def plot():
        # the figure that will contain the plot
        fig = Figure(figsize = (2.5, 2.5),
                     dpi = 100)

        # adding the subplot
        plot1 = fig.add_subplot(111)

        vergence = eye_movments[0][0][2][0] + eye_movments[0][0][2][1]

        # plotting the graph
        plot1.plot(vergence)

        # creating the Tkinter canvas
        # containing the Matplotlib figure
        canvas = FigureCanvasTkAgg(fig,
                                   master = plotWindow)
        canvas.draw()

        # placing the canvas on the Tkinter window
        canvas.get_tk_widget().place(anchor = CENTER, relheight=.250, relwidth=.250, relx = .50, rely
= .50)

        # creating the Matplotlib toolbar
        toolbar = NavigationToolbar2Tk(canvas,
                                       plotWindow)
        toolbar.update()

        # placing the toolbar on the Tkinter window
        canvas.get_tk_widget().pack()
    def setEyemvmnt_option(event):
        global Eyemvmnt_option
        if Eyemvmnt_option.get() == "FAR":
            eye_movments[0][0][k][0]
        if Eyemvmnt_option.get() == "NEAR":
```

```
        eye_movments[0][0][k][0]
    if Eyemvmnt_option.get() == "SACCADES":
        eye_movments[0][0][k][0]

  FAR_mvmttype = ["CON_04_08", "CON_02_06", "CON_02_08", "DIV_08_04", "DIV_06_02",
"DIV_08_02", "CON_STEP_2to6_RAMP_6to8", "DIV_STEP_8to4_RAMP_4to2"]
  NEAR_mvmttype = ["CON_08_12", "CON_06_10", "CON_06_12", "DIV_12_08", "DIV_10_06",
"DIV_12_06", "CON_STEP_6to10_RAMP_10to12","DIV_STEP_12to8_RAMP_8to6"]
  SACCADES_mvmttype = ["R05", "R10", "L05", "L10"]
  EyemvmntList = ["FAR","NEAR","SACCADES"]

#   MvmntType_option = FAR_mvmttype
  MvmntType_option = StringVar()
  MvmntType_option.set( FAR_mvmttype[0])

  Eyemvmnt_option = StringVar()
  Eyemvmnt_option.set(EyemvmntList[0])

  MvmntTypeDropDown = OptionMenu(plotWindow, MvmntType_option, *FAR_mvmttype)
#Dropdown Menu
  MvmntTypeDropDown.place(anchor = CENTER, relheight=.150, relwidth=.150, relx = .35, rely =
0.15)

  EyemvmntDropDown = OptionMenu(plotWindow, Eyemvmnt_option, *EyemvmntList)
#Dropdown Menu
  EyemvmntDropDown.place(anchor = CENTER, relheight=.150, relwidth=.150, relx = .15, rely =
0.15)


  plot_button = Button(master = plotWindow,
                       command = plot,
                       height = 2,
                       width = 10,
                       text = "Plot")

  # place the button
  # in main window
  plot_button.place(anchor = CENTER, relheight=.150, relwidth=.150, relx = 0.15, rely = 0.35)

  # sets the geometry of toplevel
  plotWindow.geometry("1080x720")



VNEL_Image = Image.open(r"C:\Users\Farhan Ahmad\Desktop\rawData\RETINAS
Image\VNEL_final.png")
VNEL_photo = ImageTk.PhotoImage(VNEL_Image)
copy_of_image = VNEL_Image.copy()
```

```
VNEL_Label = Label(root, image = VNEL_photo, width = 100, height = 50)
VNEL_Label.bind('<Configure>', resize_image)
VNEL_Label.place(anchor = SE, relheight= .250, relwidth= .250, relx= 1, rely= 1)

quitButton = Button(root, text = "Quit", width = 10, height = 5, command = stop) #Quit Button
quitButton.place(anchor = CENTER, relheight=.150, relwidth=.150, relx = .15, rely = .75)

LoadModeButton = Button(root, text = "Load Preprocessed Data", width = 10, height = 5,
command = loaddata) #Load Button
LoadModeButton.place(anchor = CENTER, relheight=.150, relwidth=.150, relx = 0.15, rely = .5)

def setgain_type(event):
   global Cal_option
   if Cal_option.get() == "NORMAL":
      mylist.insert(END, "Gain selection using NORMAL procedure")
   elif Cal_option.get() == "SACCADES":
      mylist.insert(END, "Gain selection using SACCADES procedure")
   elif Cal_option.get() == "AUTOMATIC":
      mylist.insert(END, "Gain selection using AUTOMATIC procedure")

Cal_options = ["NORMAL", "SACCADES", "AUTOMATIC"]
Cal_option = StringVar()
Cal_option.set(Cal_options[0])

GainDropDown = OptionMenu(root, Cal_option, *Cal_options, command = setgain_type)
#Dropdown Menu
GainDropDown.place(anchor = CENTER, relheight=.150, relwidth=.150, relx = .35, rely = 0.75)

PreprocessModeButton = Button(root, text = "Preprocess", width = 10, height = 5, command =
preprocess) #Preprocess button
PreprocessModeButton.place(anchor = CENTER, relheight=.150, relwidth=.150, relx = .35, rely =
0.5)

AnalyzeModeButton = Button(root, text = "Analyze", width = 10, height = 5, command = analyze)
#Analyze button
AnalyzeModeButton.place(anchor = CENTER, relheight=.150, relwidth=.1500, relx = 0.55, rely =
0.5)

scrollbar = Scrollbar(root)
scrollbar.place(anchor = CENTER, relheight=.150, relwidth=.025, relx = 0.85, rely = 0.25)
mylist = Listbox(root, yscrollcommand = scrollbar.set )
mylist.place(anchor = CENTER, relheight=.150, relwidth=.150, relx = 0.75, rely = 0.25)
scrollbar.config( command = mylist.yview )

PlotModeButton = Button(root, text = "Plot Data", width = 10, height = 5, command =
openNewWindow) #Plot data button
PlotModeButton.place(anchor = CENTER, relheight=.150, relwidth=.150, relx = 0.55, rely = 0.75)
```

```
myLabel = Label(root, text = "Select The Action You Wish To Choose", font = ("Purisa", 15), width
= 10, height = 5)
myLabel.place(anchor = CENTER, relheight=.100, relwidth=.250, relx = 0.15, rely = 0.15)

root.geometry("1080x720")
root.mainloop()
```

APPENDIX B

HUMAN VS R.E.T.I.N.A.S. LATENCY DATA

Human	R.E.T.I.N.A.S.
0.246	0.136
0.238	0.120
0.232	0.654
0.696	0.102
0.53	0.262
0.2218	0.276
0.256	0.334
0.248	0.378
0.208	0.222
0.21	0.260
0.254	0.186
0.1982	0.238
0.312	0.102
0.292	0.102
0.3744	0.718
0.206	1.196
0.21	0.102
0.208	0.270
0.214	1.160
0.2648	0.116
0.3	0.578
0.282	0.102
0.3374	0.424
0.2442	0.102
0.2258	0.268
0.23	0.102
0.8525	0.102
0.3733	0.782
0.3825	0.220
0.2487	0.102
0.312	0.164
0.174	0.140
0.248	0.496
0.1705	0.178
0.214	0.102
0.1521	0.170
0.2074	0.258
1.228	0.516
0.218	1.130
0.2442	0.198
0.2298	1.422

0.264	0.102
0.33	0.102
0.168	0.102
0.2166	0.102
0.216	0.964
0.1705	0.258
0.2056	0.116
0.168	0.352
0.1922	0.392
0.252	0.267
0.1949	0.114
0.1868	0.308
0.2298	0.102
0.211	0.340
0.1613	0.102
0.2594	0.600
0.246	0.338
0.2442	0.470
0.208	0.347
0.203	0.250
0.252	0.334
0.212	0.158
0.258	0.284
0.276	0.102

Human Vs R.E.T.I.N.A.S. Final Amplitude Data

Human	R.E.T.I.N.A.S.
3.3797	1.714
3.5338	1.841
3.7288	1.679
3.3453	4.264
2.9713	1.925
4.6135	1.528
4.667	1.767
4.5677	1.229
3.0695	1.866
0.6753	1.907
1.2109	1.203
0.4571	1.544
3.0596	4.468
2.2158	2.487
0.5865	1.509
4.0639	1.304
3.7315	1.603
4.1384	1.537
3.3699	1.090
4.009	1.535
4.1006	1.298
4.0406	1.525
3.7601	1.044
3.0585	1.802
2.2243	1.532
3.7032	2.904
3.0147	1.026
3.6798	2.713
3.8092	4.208
2.9789	2.230
3.8996	2.146
4.6016	4.455
4.4428	1.522
4.4317	4.048
4.6919	1.602
4.6346	4.356
4.2732	4.458
4.3098	2.649
4.3639	4.132
3.3398	1.373
3.2817	2.988
4.2196	1.339
4.0412	2.502

4.423	1.046
3.8826	1.202
4.5793	1.080
4.0814	1.592
4.1221	1.836
3.7091	1.136
4.4266	1.875
4.2778	1.973
4.1085	1.931
3.9618	2.692
3.144	3.073
3.7337	2.572
3.6906	3.141
3.656	1.693
3.6137	1.885
3.7244	1.170
3.9377	1.468
4.4257	4.322
4.4636	2.162
4.0322	2.117
4.5621	1.980
3.6715	4.786

Human Vs R.E.T.I.N.A.S. Peak Velocity Data

Human	R.E.T.I.N.A.S.
10.7165	16.172
10.009	14.447
10.8732	11.342
7.0793	12.742
4.9345	11.610
16.0948	11.573
14.6448	9.740
15.9397	16.670
12.891	11.456
8.5144	22.610
7.835	18.153
3.7159	16.820
7.7495	18.664
4.1456	18.176
1.3047	13.540
10.774	16.285
14.6801	21.370
12.0319	9.881
14.8472	24.869
11.6516	12.691
9.774	24.812
9.3139	16.586
8.5854	25.737
9.4598	11.750
9.7972	11.600
10.2007	11.500
11.274	8.858
8.352	18.994
7.3087	24.480
10.9129	5.350
7.8292	13.890
18.0064	2.715
7.7009	9.675
14.8706	15.358
19.8708	2.170
21.097	13.579
10.3521	26.250
18.9755	14.970
13.7074	22.630
6.3565	15.982
7.5614	13.437
12.8372	11.186
10.9242	23.560

8.5538	11.631
7.361	23.329
13.1491	17.240
13.6461	2.540
14.8495	11.860
14.8158	16.229
12.9024	21.323
18.5257	14.110
17.5655	8.178
13.4847	17.160
11.7514	16.461
11.1788	3.978
9.8343	17.720
13.5276	11.160
13.2339	11.450
12.1802	25.456
11.3719	22.290
12.2976	13.318
11.0531	17.684
11.3179	11.222
12.1755	11.772
8.1986	14.444

Human Vs R.E.T.I.N.A.S. Time to Peak Velocity Data

Human	R.E.T.I.N.A.S.
0.56	0.148
0.512	0.410
0.462	0.136
0.832	0.666
0.678	0.284
0.448	0.992
0.404	0.350
0.354	0.390
0.356	0.254
0.376	0.260
0.404	0.418
0.232	0.298
0.4	0.220
0.372	0.650
0.656	0.856
0.404	1.196
0.374	1.014
0.272	0.296
0.258	1.214
0.476	0.448
0.504	1.148
0.54	0.420
0.504	1.240
0.578	0.128
0.446	0.292
0.476	0.242
0.866	0.520
0.624	0.906
0.68	0.344
0.548	0.192
0.44	0.264
0.322	0.326
0.33	0.864
0.43	0.236
0.274	0.900
0.416	0.340
0.304	0.276
0.386	0.694
0.448	1.212
0.284	0.266

0.294	1.484
0.416	0.122
0.386	1.350
0.38	0.102
0.308	1.332
0.314	1.442
0.478	0.498
0.34	0.420
0.35	0.364
0.322	0.556
0.34	1.324
0.356	0.186
0.384	1.094
0.3	0.744
0.442	1.348
0.398	0.102
0.472	0.774
0.476	0.338
0.504	1.116
0.302	1.348
0.294	1.398
0.496	0.350
0.45	1.396
0.372	0.342
0.396	0.116

REFERENCES

1. Daniel M. Albert, D.M.G. *Ophthalmoplegia*. 2018; https://www.britannica.com/science/ophthalmoplegia].
2. Coubard, O., et al., *Effects of TMS over the right prefrontal cortex on latency of saccades and convergence.* Invest Ophthalmol Vis Sci, 2003. **44**(2): p. 600-9.
3. Giesel, M., et al., *Relative contributions to vergence eye movements of two binocular cues for motion-in-depth.* Scientific Reports, 2019. **9**(1): p. 17412.
4. Scheiman, M., J. Gwiazda, and T. Li, *Non-surgical interventions for convergence insufficiency.* The Cochrane database of systematic reviews, 2011(3): p. CD006768-CD006768.
5. Ciuffreda, K.J., *Eye Movements; Vergence*, in *Encyclopedia of the Neurological Sciences (Second Edition)*, M.J. Aminoff and R.B. Daroff, Editors. 2014, Academic Press: Oxford. p. 258-259.
6. Convergence Insufficiency Treatment Trial Study, G., *Randomized clinical trial of treatments for symptomatic convergence insufficiency in children.* Archives of ophthalmology (Chicago, Ill. : 1960), 2008. **126**(10): p. 1336-1349.
7. *Convergence insufficiency/excess*. 2018; Available from: https://visiontherapypaducah.com/convergence-insufficiency-excess/.
8. Alvarez, T.L., et al., *The Convergence Insufficiency Neuro-mechanism in Adult Population Study (CINAPS) Randomized Clinical Trial: Design, Methods, and Clinical Data.* Ophthalmic epidemiology, 2020. **27**(1): p. 52-72.
9. Dalton, P.D., et al., *Chapter 17 - Tissue Engineering of the Nervous System*, in *Tissue Engineering (Second Edition)*, C.A.V. Blitterswijk and J. De Boer, Editors. 2014, Academic Press: Oxford. p. 583-625.