ABSTRACT

MARRON, ANDREW J. Validation of Inertial Center of Mass Acceleration for Detection and Feedback of Asymmetric Walking. (Under the direction of Dr. Jason Franz).

The ability to walk is an important part of quality of life, and one of the main obstacles to that ability in an aging population is hemiparesis. Hemiparesis is a condition that can result from a stroke where one leg is left paretic (weakened) while the other is not. While hemiparesis may be a lifelong impairment, it can be reduced using rehabilitative measures. Further, many patients have a capacity for improvement called a propulsive reserve that can be tapped into using training with feedback. A limitation to current approaches is that the continuous measurement of propulsive forces requires an instrumented treadmill. Suitable alternatives are therefore needed.

This study investigated the use of an Inertial Measurement Unit (IMU) as a simple and portable device for gait measurement and feedback. Here, an IMU was attached to the body's approximate center of mass (COM) to measure anterior COM acceleration for comparison to the pelvis acceleration as measured by a motion capture system and propulsive ground reaction forces (GRF) measured by force plates in an instrumented two-belt treadmill. These measurements were taken for unimpaired control subjects under conditions of normal walking and a simulated hemiparetic gait condition induced by wearing a leg brace on one leg. In addition to validation comparisons, this study used the IMU's data to provide feedback to change the subjects' gait by increasing anterior acceleration peaks from both legs, decreasing acceleration from both legs, or increasing acceleration from only one leg.

This study found strong to moderate correlations between COM anterior acceleration as measured by the IMU with anterior acceleration as measured by the motion capture system, but only moderate to weak correlation with the propulsive GRF as measured by the treadmill. Induced asymmetry between legs was reliably detected, though the behavior of which had

decreased acceleration measurements did not match the unilateral decrease in force caused by the leg brace. Visual feedback was used to successfully produce increased acceleration peaks and asymmetric gait but was unsuccessful at decreased acceleration peaks and at determining which leg increased in the asymmetric case. This study demonstrated the ability of the IMU based system to provide comparable results to motion capture, which could potentially lead to similar systems being used to detect gait asymmetry or to assist in hemiparetic rehabilitation outside of a clinical environment.

Validation of Inertial Center of Mass Acceleration for Detection and Feedback of Asymmetric Walking

by
Andrew James Marron

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Master of Science

Biomedical Engineering

Raleigh, North Carolina

2018

APPROVED BY:

_____	_____
Jason Franz Committee Chair	Michael Lewek
_____	
Gregory Sawicki	

BIOGRAPHY

Andrew Marron was born in North Carolina. He studied mechanical engineering as an undergraduate at North Carolina State University. As a graduate student he is studying biomedical engineering in the joint department at NC State University and The University of North Carolina at Chapel Hill.

ACKNOWLEDGMENTS

I would like to thank Mike Lewek, my advisor, for his continual support in this project and as a Master's student.

I would like to thank my committee as a whole for their advice and understanding during my time as a graduate student.

I would like to thank my father James Marron for his advice and perspective on my research and this thesis.

I would also like to thank the subjects of this study for their time and participation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Hemiparesis is a weakness of muscles on one side of the body (ranging from individual limbs to the entire half of the body). Hemiparesis is usually caused by a stroke damaging one side of the brain, and occurs in 88% [1] of stroke survivors. Strokes are increasingly common in the aging modern population, with 665,000 non-fatal new strokes per year in the United States alone [2], [3]. This makes strokes the leading cause of disability in the United States [4], and hemiparesis the leading cause of stroke-related disability. Hemiparetic gait is an abnormal pattern of walking resulting from one leg being weaker than the contralateral limb. Such a walking pattern can require up to twice as much metabolic energy expenditure at a given speed as unimpaired gait [5], [6]. This substantial increase in the effort required for walking leads to a cycle of decreased walking activity and muscular atrophy, greatly reducing long-term mobility. The loss of mobility from hemiparesis can have major impacts on stroke survivors, as day to day self-care becomes more difficult, leading to a significant decrease in quality of life of those affected [7], [8].

Hemiparesis is a lifelong disability, as no current rehabilitation method leads to complete recovery from stroke. Current methods of treatment can reduce the effects of hemiparesis but recovery is often considered to plateau by six months after the stroke, with a significant amount of residual disability [9]. Current methods to restore gait function used in clinical practice are based on task-specific training, or consist of training exercises that mimic and challenge the specific activities (such as walking) that are difficult. Further increases in recovery from these methods after the first six months of rehabilitation have shown energy cost decreases of 10% to 15% [10], [11], which remain elevated above similarly aged unimpaired individuals.

Despite the appearance of a plateau in functional gains [9], individuals in the chronic stages of stroke have the ability to walk faster than they typically do [12], [13]. This ability to walk faster may be due to the ability of those impaired by age [14] or hemiparesis [15] to exert more anteriorly directed force than they typically do with their weakened legs or leg. This is called a propulsive reserve. While the propulsive reserve demonstrates the capacity for immediate improvement, the paretic and healthy legs are not equally in need for improvement. Due to unilateral weakness, hemiparesis causes asymmetric gait as the paretic leg provides less propulsion than the non-paretic leg [16]. Asymmetry has been linked to increased energy expenditure (as measured by metabolic O2 consumption) in studies on the gait of both healthy [17] and post-stroke hemiparetic [18] subjects. Conversely, walking with increased symmetry and faster speed is associated with decreased energetic expenditure suggesting that decreasing the propulsive asymmetry between the paretic and healthy legs can reduce the energetic cost of walking [19].

Within each leg, propulsion is a function of multiple factors such as ankle moment, trailing limb angle, and knee compliance [20], and as any of these could be affected by paretic muscle, it is likely more fruitful to approach improvement on an entire-limb level and help patients intuitively find strategies in response to the specifics of their paresis. This study will provide that leg-level assistance using a center of mass (COM) based feedback system. COM based systems have been demonstrated previously to be effective in applications such as measuring gait in the context of Parkinson's patients [21] and in a previous hemiparesis study where it was used to give feedback on vertical and lateral COM movement [22]. The COM feedback used by Massaad [22] is of particular note due to its substantial energy cost reduction of 30% after six months of training while using methods that other research suggests is not

optimal [23]. This suggests that these results could be further improved on by targeting propulsion instead of vertical movement, since reducing center of mass movement is known to be ineffective or counterproductive in unimpaired control subjects [23]. Additionally, Massaad and colleagues focused the COM feedback on only medial-lateral and vertical directions, whereas anterior-posterior directions are more directly involved in propulsion [24]. An example of this propulsive approach has previously been used for individuals in the chronic phase post-stroke [25]. Specifically, visual feedback of anterior ground reaction forces produced a significant improvement in propulsive peaks after a single 18 minute training session. While these results are promising, the requirement of a split-belt instrumented treadmill to generate the feedback data limits the application of the feedback system to clinical settings that have instrumented treadmills.

Our approach combines the advantages of both these methods by using center of mass based tracking of anterior-posterior movement while reducing the bulk and expense of motion capture or instrumented treadmill systems by using an IMU for COM tracking in a similar manner to the one used previously for Parkinson's patents [21]. Motion capture systems are expensive, require significant calibration and setup time, and require the subject to be on a treadmill as the cameras are stationary, with the instrumented treadmills themselves being expensive and immobile equipment unsuited for use by individual patients. IMUs are in contrast portable, lightweight and inexpensive, which makes them potentially usable during overground walking in a portable gait analysis to provide assessment and feedback outside of a clinical setting [24].

This study has two parts, though for the sake of brevity it will be reported as one study. First, it will demonstrate the IMU's validity as a sensor for detecting regular and asymmetric gait

behavior while being responsive to change. This will be done by comparing the IMU's data to acceleration data derived from the current gold standard of motion capture data and by comparing the IMU's acceleration to the propulsive force for each leg as measured by the treadmill force plates. Second, this study will demonstrate the ability of the IMU to modify gait as part of a real-time feedback system. This will be done by having subjects modify their gait using visual feedback and measuring the changes in acceleration and force. The three hypotheses produced by these goals are as followed. First, it is hypothesized that the IMU measurements will correlate with motion capture acceleration and force plate propulsive GRF data. Second, it is hypothesized that the IMU will detect a difference between 'impaired' and unimpaired legs during simulated hemiparetic gait. Third, it is hypothesized that unimpaired subjects will be able to modify their gait in response to anterior COM visual feedback.

Methods:

Participants

Individuals were recruited for this study who were healthy unimpaired adults. The exclusion criteria for the study included any cardiovascular, neurologic or orthopedic injury or disorder that would limit or affect gait. Individuals with low back or leg pain were excluded. Subjects had to have normal, or corrected to normal, vision and needed to be English speaking in order to understand instructions. Participants were recruited from the local University area via fliers and word of mouth. Prior to participation, all subjects read and signed a consent form approved by the University of North Carolina Institutional Review Board.

Testing Procedures

Data for each subject were collected in a single session. Prior to testing, subjects walked across a 16 foot pressure mat (Zeno, Protokinetics, Havertown PA) to determine self-selected walking speed. Within-subject testing then consisted of walking on a dual-belt instrumented treadmill (Bertec Corp, Columbus, OH) for periods of 150 seconds under five conditions in the same order for each subject. In each condition the subject walked at their self-selected overground speed without holding onto the handrails. A minimum of a 5-minute rest break was taken between each condition to avoid cross-contamination between conditions. In the first two conditions, participants walked without receiving any instructions or feedback about their walking:

(1u) – subjects walked on the treadmill under a 'control' walking condition.

(1b) – subjects walked on the treadmill with both a knee immobilizer and a cast shoe on their left leg. These items were an established method of restricting movement in only that leg to approximate the impairments present in hemiparesis [26], [27].

In the remaining conditions, the subjects were given different instructions for each condition to consciously manipulate their gait based on real-time visual feedback. Visual feedback was projected onto a screen in front of the treadmill for the subjects to see while walking. Specifically, the feedback provided information depicting their positive anterior acceleration peaks for left and right steps as measured by an IMU. The subjects were given practice with these conditions for several minutes before the testing of these conditions.

(2bd) – subjects were instructed to decrease the anterior acceleration peaks of both left and right sides.

(2bi) – subjects were instructed to increase the anterior acceleration peaks of both sides.

(2ui) – subjects were instructed to increase the anterior acceleration peaks associated with a left step only.

Anterior acceleration peaks were collected from an IMU secured firmly on the posterior pelvis with an elastic wrap. The IMU (3Space, Portsmouth, Ohio) itself was a rectangular device, 1.38 x 2.36 x 0.59“ in size and weighing 29 grams. It was secured to the subject with a custom casing designed to clip onto a belt. Sensor data were streamed directly to the processing computer using a standard USB cable (due to difficulties with the sensor’s Bluetooth). Relevant signals included the IMU’s time stamp and triaxial accelerometer data. These data were collected and processed in real-time using a custom Python program (see Appendix). Data were low-pass filtered using a 20 Hz continuous low pass filter. Displayed antero-posterior acceleration data were de-meaned (except for the first 3 subjects) with a window length of approximately two seconds to reduce the effects of pelvic/trunk lean (either forward or backward) on the IMU. Local peaks were extracted to represent each step. The determination of each step as a left or right step (with left step defined as occurring at left heel strike and a right step occurring at right heel strike) was determined by measuring a subject’s average frontal plane pelvic acceleration (i.e., IMU’s x-direction) during the middle of positive anterior acceleration as either positive or negative. The visual feedback as shown in figure 1 was displayed to subjects using a moving oscilloscope-like line of the sagittal plane movement of the pelvis, with step peak indicators to represent current movement and color-coded left / right bars on the right of the display to represent the average height of the previous three steps.

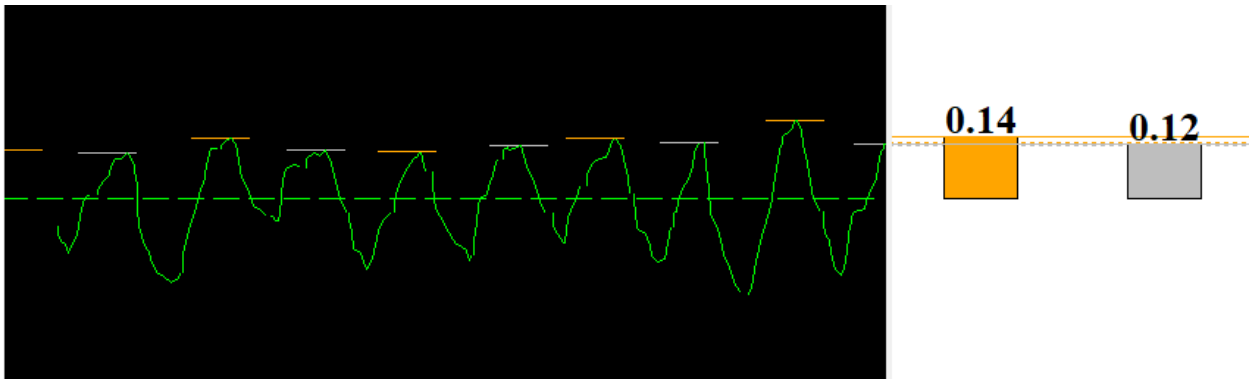

Figure 1, Sample subject-facing output. The green line represents pelvis acceleration, while the orange and grey horizontal lines respectively represent left and right step peaks. The orange and grey bars represent the average heights of the last three left or right peaks.

During each condition, data were collected using force plates built into the treadmill, a Vicon (Los Angeles, CA) motion capture system, and the accelerometer from the IMU. The force plates in the treadmill were used to capture ground reaction forces. The Vicon system consisted of an 8 camera passive motion capture system to identify the 3D location of 14mm reflective markers taped to rigid shells and affixed to the pelvis of each subject with an elastic wrap. The Vicon system along with the force plates was used to capture limb and pelvic movements and forces (kinematics and kinetics) sampled at 120 and 1200 Hz, respectively.

Data Processing

After testing, data recorded with the Vicon system and force plates were processed using Nexus (Ver 1.8.5) software and a Visual3D (C-Motion, Bethesda, MD) script to find acceleration from the pelvis marker's position and label heel strikes in the force data. Occasional treadmill belt crossover error (from a foot touching the other foot's belt) was excluded from the force data. The bottom pelvis marker from the motion capture data in particular was used as an approximation of the body's COM location (as in [28], [29]) and was used as the gold standard

reference for the IMU affixed to the subjects' pelvis. The data from all three sources (i.e., IMU anterior acceleration, pelvis anterior acceleration via Vicon, and anterior propulsive forces from the treadmill) were then analyzed in a Python script to synchronize signals and find peaks. Given the slower sampling frequency of the IMU data, a time scaling constant was identified to match the Vicon data's temporal structure. The IMU and Vicon data were then synchronized by matching early peaks and discarding the data before that point as shown in figure 2. Data were checked towards the end of the time stream to confirm synchronization. The force plate data was recorded with the same software as the Vicon data but was sampled at a higher frequency and thus required down-sampling to ensure synchrony. Heel strikes were plotted in the combined plot as vertical lines as shown in figure 3. Peaks of positive acceleration were found and labeled right or left using the same method as the live feedback (plotted as red or blue circles as shown in figure 2). Peaks were found again on the first positive anterior section of acceleration after each right or left heel strike as a more reliable right/left indicator method. The combined antero-posterior force on the force plates and the peaks of the combined force were also calculated. The IMU data was then re-sampled using linear interpolation to fit the Vicon acceleration data's time scale, and the resulting acceleration, force, peak magnitude and peak side data was output for statistical analysis.

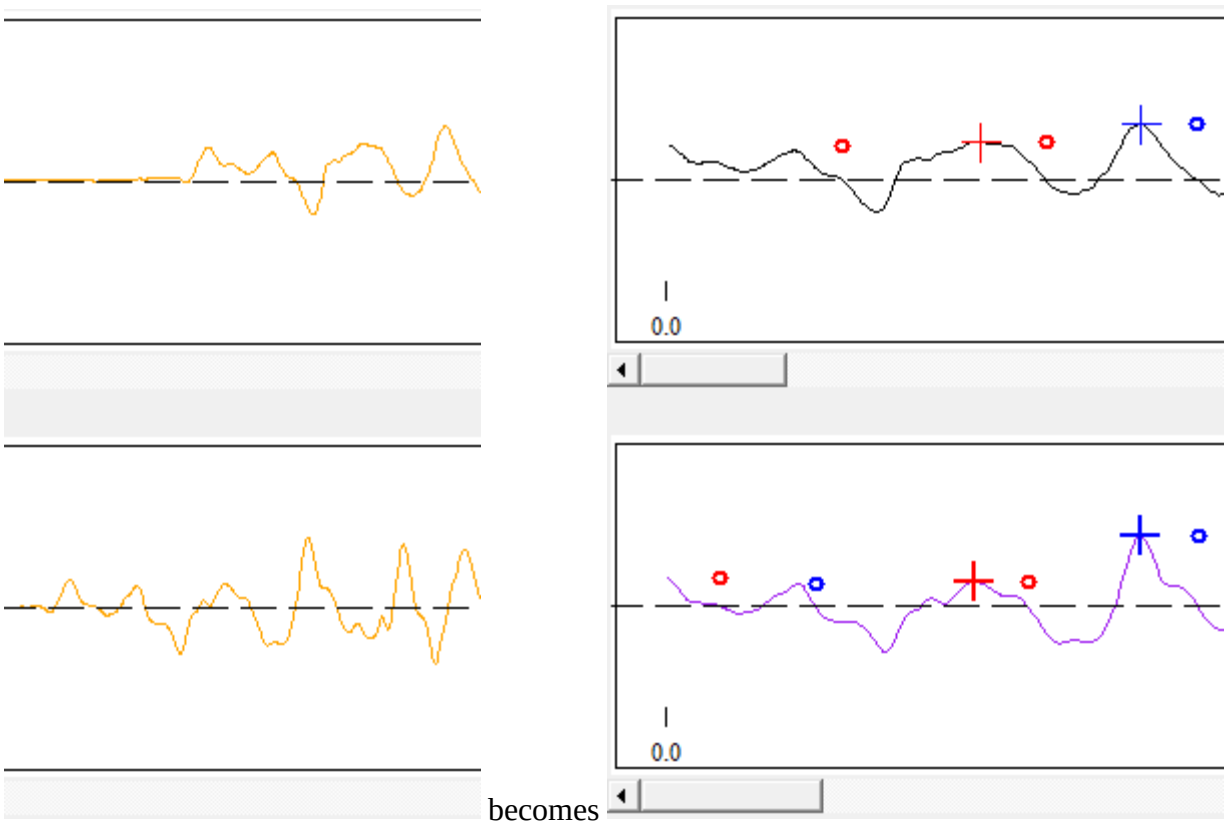

Figure 2, IMU and Motion Capture Synchronization. Displayed is IMU (top) and motion capture (bottom) anterior acceleration plot excerpts of motion start before and after synchronization. Red/blue plus marks represent right/left acceleration peaks found based on heel strike times and red/blue circles are peak heights found between zero crossings.

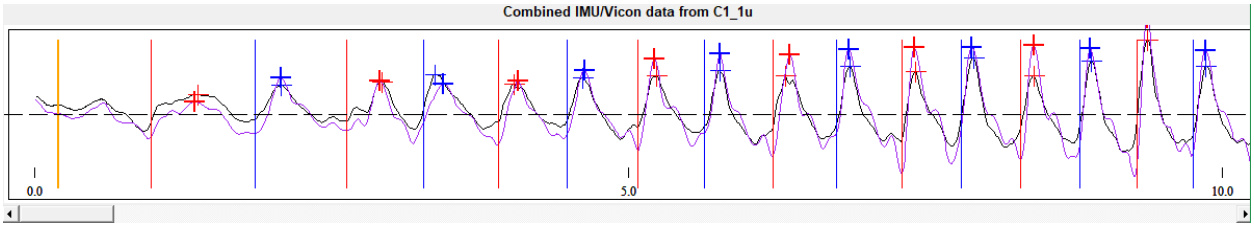

Figure 3, Synchronized IMU and Motion Capture Data Plot Overlay Excerpt. Red/blue vertical lines represent right/left heel strikes while red/blue plus marks represent anterior acceleration right/left peaks.

Statistical analyses of the synchronized data streams and peak data points were performed using SPSS (Ver 24 Chicago, Illinois). For each trial of each subject, Pearson Correlations were performed for the IMU peak magnitudes, Vicon peak magnitudes and the peaks of the corresponding individual propulsive forces. Another set of Pearson correlations were performed for all the time points of the IMU and Vicon acceleration as well as the summed horizontal force. Descriptive statistics of left and right IMU acceleration peak magnitudes and their standard deviations were calculated for comparison. Left and right in the analysis were found based on the vertical ground reaction force (heel strike) when not specified otherwise, as this was found to be significantly more reliable. A repeated measures ANOVA was used to compare the IMU acceleration peak magnitude between left and right steps for each participant. Descriptive statistics were calculated for averages of |IMU – Vicon| peak error, IMU and Vicon peak magnitude, |IMU – Vicon| all point acceleration, and right/left force peak magnitude. For conditions with intended leg difference (1u and 2ui), each subject's leg peak means were compared to each other via an ANOVA test to check for interaction between limb and condition as a measurement for different behavior between conditions.

Results:

Subjects

Nine unimpaired individuals were recruited for this study. Subjects were a mix of college aged to younger middle aged, with most participants being college students. This study recruited 4 male and 5 female subjects. Recurited subjects were mostly young adults of average age 26.22 ± 5.094; with average height 68.39 ± 3.257 inches and average weight 149.4 ± 25.42 lbs. Subjects walked at an average self-selected speed of 1.39±0.06 m/s.

Validation of Sensor

The control condition of 1u (walking unimpaired with no feedback) was used to validate the IMU sagittal plane acceleration data against a motion capture system and the summed force of both treadmill force plates. Correlation of all data points was consistently very high between the IMU and the Vicon motion capture system (I/V), averaging 0.931±0.031 as shown in table 1 and figure 4. A moderate negative (likely negative due to the phase difference) correlation was observed between the IMU acceleration and the sum of the A/P forces (I/F), averaging -0.62±0.112 as shown in table 1. As displayed by the scatter plots of IMU all point acceleration vs all point motion capture acceleration and all point summed force in figure 4, the all point data for I/F occurred in a hollow middle pattern (which some I/V all point scatter plots showed to a minor extent), suggesting a phase shift.

Table 1, Condition 1u All Point Correlations. All values significant $p < 0.001$.

Subject	C1	C2	C3	C4	C5	C6	C7	C8	C9	Subj Avg	Subj Stdv
1u I/V All Point Correlation	0.921	0.921	0.972	0.954	0.935	0.96	0.916	0.937	0.865	0.931	0.031
1u I/F All Point Correlation	-0.53	-0.58	-0.73	-0.76	-0.58	-0.53	-0.68	-0.77	-0.47	-0.62	0.112

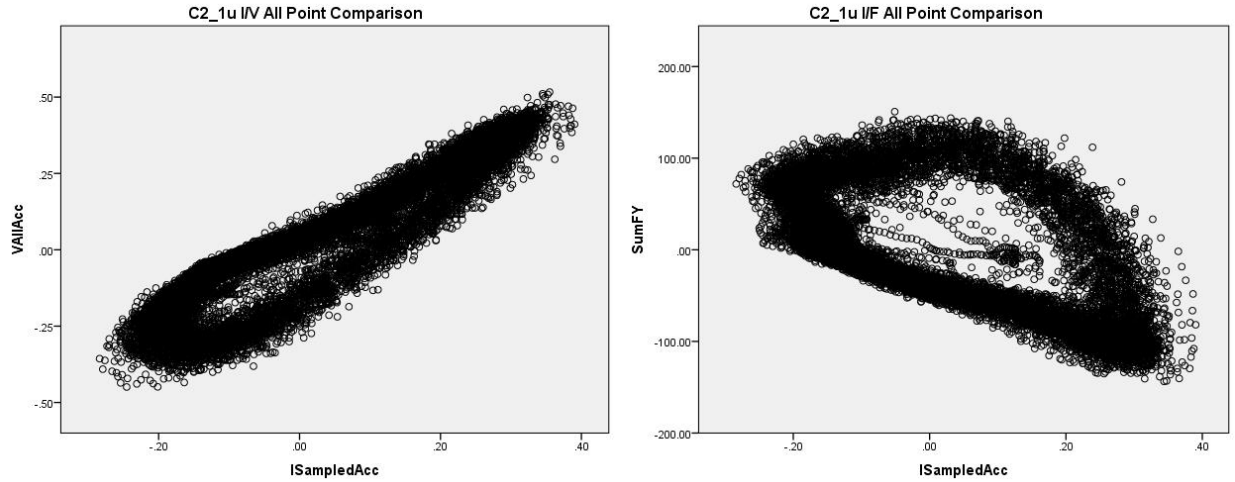

Figure 4, Condition 1u I/V and I/F All Point Examples. Left, all points of IMU acceleration vs motion capture acceleration. Right, all points of IMU acceleration vs summed force.

Correlations of peak values were also determined between the IMU, the motion capture system, and the peak anteriorly directed GRF for each step. I/V peak correlation for this condition was moderate, averaging 0.683±0.095 as shown in table 2 and figure 5, a scatter plot of peak IMU vs motion capture and IMU vs Force. I/F peaks had relatively low correlations, averaging 0.298±0.113, when significant. While significant, I/F peak correlations may have been due to outliers as shown in figure 5, where there is little visible correlation in general. Comparisons of motion capture peaks to force peaks demonstrated a similar lack of correlation rather than this being a problem specific to the IMU, as shown in figure 6.

Table 2, Condition 1u Peak Correlations. With the exception of C4 I/F (which had $p > 0.05$), all values significant $p < 0.001$.

Subject	C1	C2	C3	C4	C5	C6	C7	C8	C9	Subj Avg	Subj Stdv
1u I/V Peak Correlation	0.758	0.768	0.658	0.587	0.597	0.743	0.798	0.524	0.71	0.683	0.095
1u I/F Peak Correlation	0.546	0.357	0.262	not sig	0.277	0.216	0.283	0.18	0.26	0.298	0.113
1u I/V Peak Error	0.111	0.091	0.113	0.055	0.086	0.049	0.129	0.038	0.055	0.081	0.033

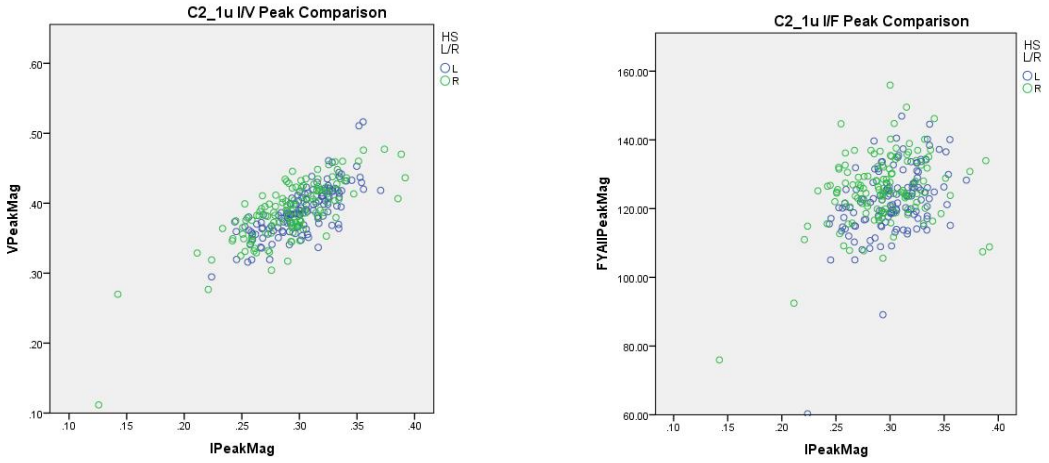

Figure 5, Condition 1u I/V and I/F Peak Examples Displaying Left and Right Step Peaks. There is visible correlation for IMU acceleration vs motion capture but not vs force.

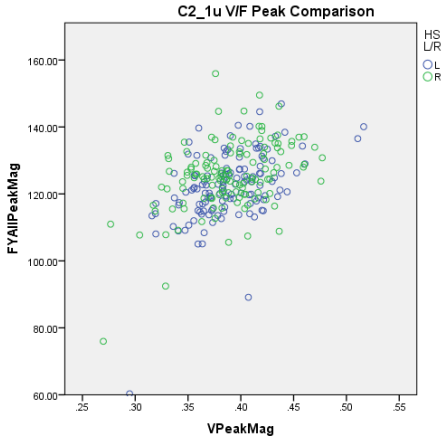

Figure 6, Condition 1u Motion Capture vs Force (V/F) Example. The acceleration here from motion capture has little correlation to force peaks, similar to the IMU results.

The IMU measured an average absolute peak error of 0.081±0.033 g. When distributed as left and right peaks, only subtle differences were observed between peaks (average η_p^2 = 0.07±0.08 indicating a small effect size).

Validation of Sensor 2ith Simulated Hemiparetic Gait

The braced, no-feedback condition (1b) was used to validate the IMU's ability to function in asymmetric gait and detect asymmetry. As shown in table 3, the all point correlations for I/V and I/F were high (though also negative for force), respectively averaging 0.914±0.023 and -0.71±0.065, much like in 1u.

Subject	C1	C2	C3	C4	C5	C6	C7	C8	C9	Subj Avg	Subj Stdv
1b I/V All Point Correlation	0.914	0.943	0.947	0.883	0.914	0.932	0.899	0.899	0.892	0.914	0.023
1b I/F All Point Correlation	-0.72	-0.78	-0.73	-0.64	-0.71	-0.66	-0.72	-0.83	-0.63	-0.71	0.065

Table 3, I/V Condition 1b All Point Correlations. All values significant $p < 0.001$.

The I/V peak correlation averaged 0.417±0.269 when significant, as seen in table 4. I/F peak correlations were found to be even lower and highly variable, averaging -0.08±0.372. These correlations appeared to be influenced by different behaviors of the left and right legs as seen in figure 7. An extreme case of this can be seen with the data for C7 as seen in figure 8, where between-leg effects create an overall negative correlation despite the left leg having a clear positive correlation. Each leg was therefore analyzed separately as seen in tables 5 and 6.

Table 4, I/V Condition 1b Peak Correlations. All significant values significant $p < 0.05$.

Subject	C1	C2	C3	C4	C5	C6	C7	C8	C9	Subj Avg	Subj Stdv
1b I/V Peak Correlation	0.338	0.492	0.408	not sig	0.273	0.691	-0.13	0.595	0.673	0.417	0.269
1b I/F Peak Correlation	-0.4	0.422	0.268	-0.44	-0.22	-0.3	-0.36	not sig	0.388	-0.08	0.372
1b I/V Peak Error	0.074	0.076	0.052	0.08	0.076	0.043	0.108	0.068	0.042	0.069	0.021

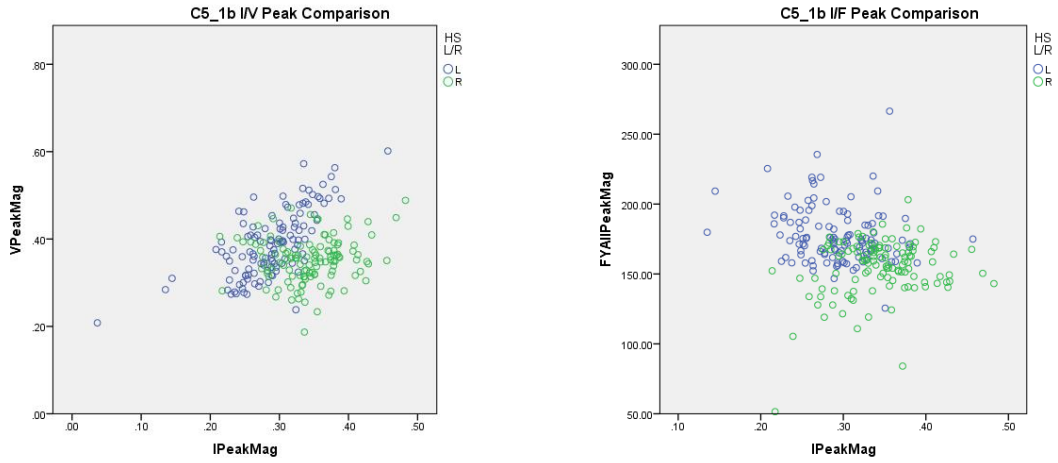

Figure 7, Example of Typical 1b L vs R Correlation Differences. The left leg has some correlation between IMU and motion capture, but neither has much correlation with force.

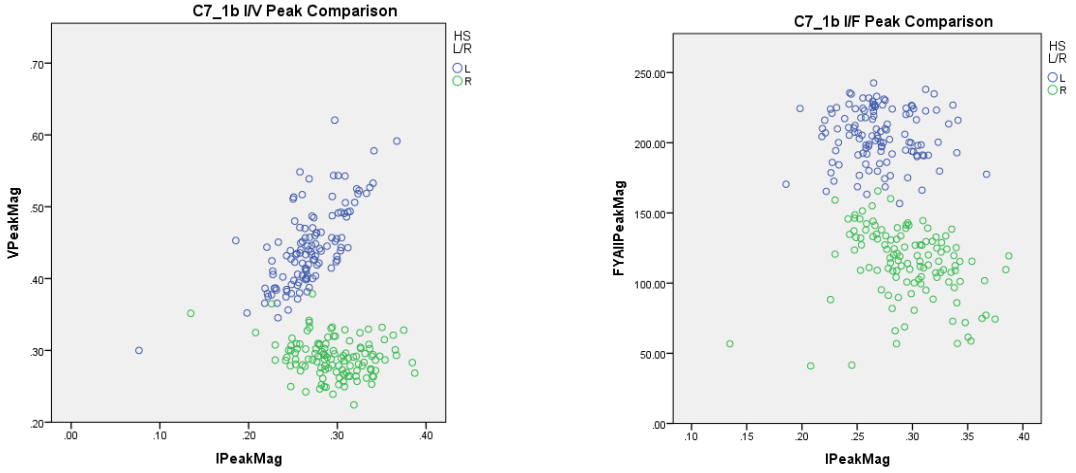

Figure 8, Example of Extreme 1b L vs R Correlation Differences. Between-leg effects create an overall negative correlation despite the left leg having a clear positive correlation.

Peak I/V correlation was found to be generally higher for the left (braced) leg than the right (unbraced) leg, averaging 0.632±0.160 as opposed to the right leg's average of 0.360±0.285 when significant as seen in table 5. I/F correlations within left and right were generally weak and not significant in most cases.

Table 5, I/V Condition 1b L R Split Peak Correlations. All significant values significant $p < 0.05$.

Subject	1b I/V L Only Peak Correlation	1b I/F L Only Peak Correlation	1b I/V R Only Peak Correlation	1b I/F R Only Peak Correlation
C1	0.763814	0.403612	0.601987	not sig
C2	0.346771	not sig	0.655298	not sig
C3	0.406	not sig	0.238434	0.270308
C4	0.82	not sig	0.361	not sig
C5	0.691	-0.195671	not sig	0.232384
C6	0.740346	0.323079	0.619951	not sig
C7	0.695	not sig	-0.21333	-0.204913
C8	0.6	0.253949	0.259916	-0.230899
C9	0.624358	not sig	0.354	not sig
Subj Avg	0.631921	0.19624225	0.359657	0.01672
Subj Stdv	0.160242673	0.268337995	0.284572759	0.271572084

Comparisons were also done between the magnitudes of the left and right steps for each subject. As seen in table 6, left leg force was smaller than right leg force for all subjects other than C2, who due to testing error was braced on the right leg instead of the left leg. Acceleration was less consistent, with left leg acceleration was smaller for most subjects but not in C3, C8, and C9. This difference between legs was found to be significant for all subjects, though lower for the accelerometer than for the force. Left and right leg acceleration were found to be significantly different ($\eta_p^2 = 0.197 \pm 0.0623$), though less different than the left and right forces

(η_p^2 = 0.584 ± 0.264). These were higher than the between leg differences in condition 1u, which were mostly not significant for acceleration and for force averaged η_p^2 = 0.0864 ± 0.0701.

Table 6, I Condition 1b Acceleration Peak Means and F Force Peak Means. η_p^2 significant p < .001. Due to testing error, Subject C2 was braced on the left leg instead of the right leg.

Subject	1b I L Peak Mean	1b I L Peak Stdv	1b I R Peak Mean	1b I R Peak Stdv	1b I L Mean / I R Mean	1b F L Peak Mean	1b F R Peak Mean	1b F L Mean / F R Mean	1b I L/R Partial Eta Sqd	1b F L/R Partial Eta Sqd
C1	0.363	0.058	0.431	0.071	0.841082	121.5	196.3	0.618892	0.22	0.85
C2	0.293	0.037	0.343	0.049	0.854459	121.66	91.31	1.332299	0.248	0.591
C3	0.321	0.051	0.288	0.06	1.116458	104.99	144.6	0.726232	0.084	0.597
C4	0.332	0.047	0.385	0.059	0.861058	104.41	168.4	0.61995	0.203	0.771
C5	0.289	0.054	0.345	0.05	0.839226	154.62	177.8	0.869632	0.225	0.251
C6	0.342	0.054	0.39	0.055	0.87761	133.72	189.7	0.704755	0.162	0.667
C7	0.269	0.036	0.296	0.039	0.907093	114.23	205.3	0.556413	0.12	0.802
C8	0.385	0.071	0.299	0.08	1.28504	134.54	143.6	0.936854	0.243	0.056
C9	0.332	0.042	0.282	0.041	1.178718	110.22	155.7	0.707941	0.268	0.671
Subj Avg	0.325	0.05	0.34	0.056	0.973416	122.21	163.6	0.785885	0.197	0.584
Subj Stdv	0.037	0.011	0.053	0.013	0.171607	16.42	34.94	0.237873	0.0623	0.264

Validation of Live Feedback

The feedback provided by the device was tested by using the following conditions: bilateral increase 2bi, bilateral decrease 2bd and unilateral increase 2ui. The peak means for the left and right legs were compared between the control condition 1u and each of these conditions to ascertain whether subjects were capable of changing gait in response to the provided feedback. As seen in table 7, in the 2bi condition only five of the nine subjects had significant condition*side interaction, and those that did had very low effect sizes averaging 0.037±0.019 when significant. They did, however, exhibit a robust increase in anterior acceleration during the

2bi condition (average $\eta_p^2 = 0.38 \pm 0.20$). The 2bd condition similarly had only three significant results with relatively low effect sizes averaging only 0.058±0.053. 2bd mean comparisons had significant effect sizes in all cases but C7, however, participants did not consistently reduce anterior acceleration. In fact, three participants actually increased their anterior acceleration, despite feedback and encouragement to decrease the acceleration. The 2ui condition had notably larger effects, with eight of the 9 subjects having significant effect sizes averaging 0.5625±0.211. 2ui was found to have similar problems to 1b where leg changes were inconsistent with both between subjects and between acceleration and force. These changes also had no clear pattern between 1u and 1b, as seen in table 8. A typical set of left to right comparisons for the control and feedback condition can be seen in figure 9, in which the conditions have relatively similar behavior between sides aside from 1ui. The atypical case where the interaction effect was not significant can be seen in figure 10, where the right leg has high variance rather than a clear pattern of difference from the left.

Table 7, Control Condition vs Feedback Conditions. 2bi and 2bd data significant to $p < 0.05$ where significant, and 2ui data significant to $p < 0.001$ where significant.

Subject	C1	C2	C3	C4	C5	C6	C7	C8	C9	Subj Avg	Subj Stdv
1u L Peak Mean	0.356	0.302	0.271	0.31	0.296	0.247	0.278	0.25	0.274	0.287	0.034
1u L Peak Stdv	0.04	0.028	0.03	0.027	0.04	0.038	0.026	0.044	0.027	0.033	0.007
1u R Peak Mean	0.302	0.292	0.239	0.308	0.305	0.285	0.277	0.254	0.269	0.281	0.024
1u R Peak Stdv	0.049	0.038	0.035	0.024	0.04	0.048	0.026	0.043	0.042	0.038	0.009
2bi L Peak Mean	0.481	0.384	0.415	0.333	0.367	0.397	0.368	0.332	0.566	0.405	0.076
2bi L Peak Stdv	0.117	0.059	0.094	0.064	0.069	0.074	0.082	0.105	0.116	0.087	0.022
2bi R Peak Mean	0.494	0.357	0.385	0.35	0.377	0.485	0.31	0.333	0.491	0.398	0.073
2bi R Peak Stdv	0.117	0.062	0.098	0.072	0.059	0.085	0.067	0.103	0.118	0.087	0.023
2bi/1u Condition * L_R Partial Eta Sqd	0.037	0.008	not sig	not sig	not sig	0.037	0.06	not sig	0.044	0.037	0.019
2bi-1u Mean Diff	0.158	0.073	0.145	0.033	0.071	0.175	0.062	0.082	0.257	0.117	0.071
2bi/1u Partial Eta Sqd	0.452	0.353	0.503	0.092	0.317	0.596	0.208	0.205	0.693	0.38	0.198
2bd L Peak Mean	0.288	0.317	0.346	0.262	0.136	0.222	0.216	0.276	0.273	0.26	0.062
2bd L Peak Stdv	0.05	0.032	0.064	0.037	0.031	0.041	0.037	0.063	0.052	0.045	0.013
2bd R Peak Mean	0.306	0.31	0.325	0.261	0.134	0.218	0.217	0.28	0.27	0.258	0.06
2bd R Peak Stdv	0.055	0.036	0.075	0.037	0.03	0.048	0.035	0.06	0.052	0.048	0.014
2bd Condition * L_R Partial Eta Sqd	0.112	not sig	not sig	not sig	0.007	0.055	not sig	not sig	not sig	0.058	0.053
2bd-1u Mean Diff	-0.03	0.016	0.08	-0.05	-0.166	-0.05	-0.06	0.026	not sig	-0.03	0.073
2bd/1u Partial Eta Sqd	0.084	0.053	0.313	0.344	0.846	0.055	0.472	0.058	not sig	0.278	0.281
2ui L Peak Mean	0.293	0.432	0.522	0.21	0.167	0.265	0.529	0.334	0.687	0.382	0.172
2ui L Peak Stdv	0.054	0.061	0.105	0.05	0.053	0.078	0.078	0.06	0.103	0.071	0.021
2ui R Peak Mean	0.609	0.283	0.56	0.492	0.469	0.573	0.159	0.248	0.229	0.402	0.172
2ui R Peak Stdv	0.088	0.048	0.155	0.102	0.062	0.102	0.036	0.045	0.074	0.079	0.037
2ui/1u Condition * L_R Partial Eta Sqd	0.704	0.373	not sig	0.574	0.689	0.472	0.791	0.177	0.74	0.565	0.211

Table 8, 2ui Leg Differences Compared to 1b

Subject	C1	C2	C3	C4	C5	C6	C7	C8	C9	Subj Avg
2ui acc L/R	0.481	1.526	0.933	0.427	0.356	0.462	3.329	1.346	2.994	1.317
2ui F L/R	1.568	0.835	1.055	1.244	0.551	0.952	0.637	1.087	0.37	0.922
1b acc L/R	0.84	0.85	1.12	0.86	0.839	0.88	0.91	1.29	1.18	0.97
1b F L/R	0.62	1.33	0.73	0.62	0.87	0.7	0.56	0.94	0.71	0.79

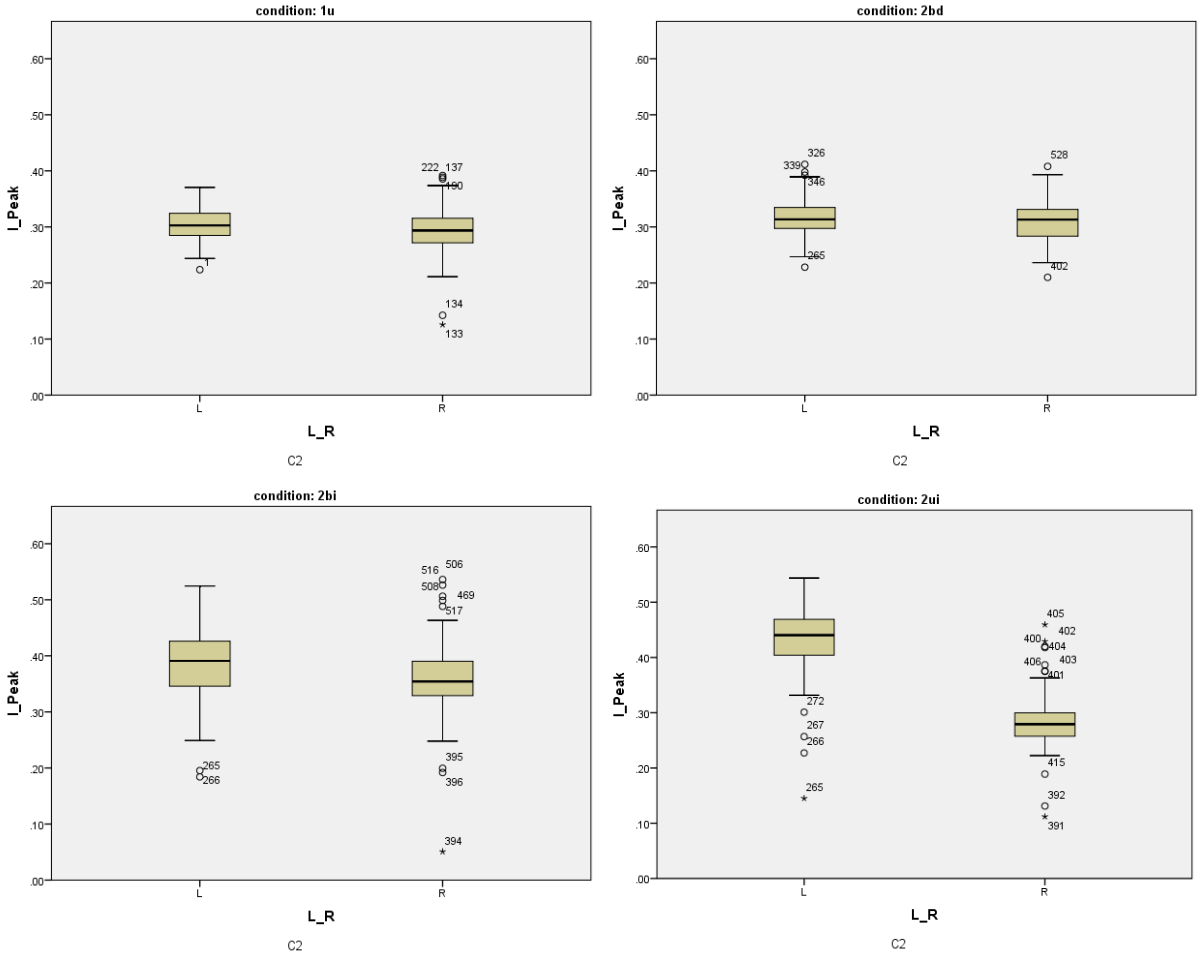

Figure 9, Example of Typical Condition Comparisons. The legs are seen to be significantly different in 2ui while they are not in 1u, 2bd, or 2bi.

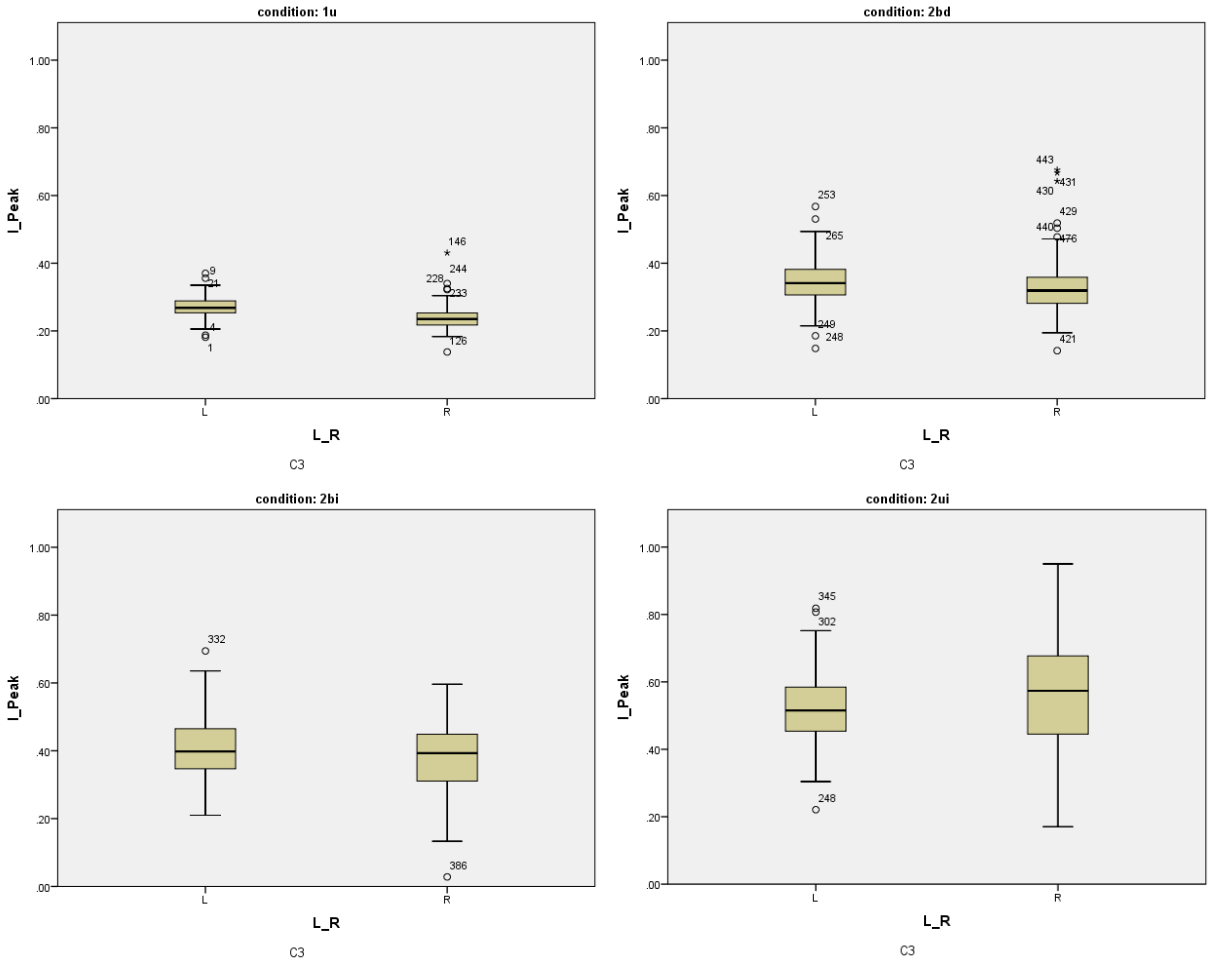

Figure 10, Comparisons for Atypical C3 Results. The legs are not seen to be significantly different in 2ui, possibly owing to increased variance.

Discussion:

The hypothesis that the IMU's measurements would correlate to the motion capture's acceleration and the force plates' propulsive GRF was partially supported by the data. The IMU measurements had strong correlations with the subject's COM acceleration as measured by motion capture, but exhibited weaker correlations with the GRF, particularly in comparing peak values. Secondly, the hypothesis that the IMU would detect a difference between limbs in the presence of an induced asymmetry was partially supported by the data. Left vs right peak accelerations were consistently found to be different, though which side was higher was not always consistent with the GRF measurements. Finally, the hypothesis that subjects would be able to modify their gait in response to COM visual feedback was also partially supported by the data. Peak accelerations for both legs was consistently increased on instruction, though responses were inconsistent in response to instructions to decrease the peaks, and instructions to increase the peak acceleration of one leg resulted in consistently different peak accelerations but with inconsistency as to which side was increased. In general, these data suggest that the IMU is a potential tool for measuring, and providing feedback regarding, anterior acceleration in hemiparetic gait comparable to motion capture. This has important implications for being able to accurately measure gait outside of a motion capture volume.

Acceleration and Ground Reaction Force Correlations

Importantly, this study found high to moderate correlations between the IMU's anterior COM acceleration and the motion capture's anterior acceleration, although only moderate to low correlations were observed between the IMU's acceleration and the force plates' propulsive GRF. In all cases the correlation of peak values was lower than the correlations of the entire data streams. The relatively lower correlations between the IMU acceleration and GRF values were

likely due in part to force and corresponding acceleration happening at different times in the gait cycle, causing the curves to exhibit a relative phase shift (as indicated in figure 4). It is also possible that the integral of propulsive force over each step, rather than the peak force used here, would relate more closely to acceleration. The strength of the correlation with motion capture acceleration suggests that the IMU represents a viable alternative to motion capture for gait analysis in the context of asymmetric gait as well as symmetric gait. This extends the work of others who have demonstrated the utility of an IMU to measure the vertical COM in individuals with Parkinson's patients [21]. IMU motion tracking has the potential to not only be a cheaper alternative to motion capture but can also be used without a large and immobile sensor configuration. The similarity of treadmill to overground gait [30] suggests that this work performed on a treadmill could be seamlessly translated to an overground environment.

Induced Left-Right Impairment Detection

The IMU consistently detected a difference between the brace-induced impairment on subjects' left legs, however the decrease in force by the braced limb was not always matched by a corresponding decrease in anterior acceleration from that limb. While the cause of this inconsistency requires investigation, it should be noted that the IMU's acceleration data continued correlating fairly well with the motion capture correlation. This implies that the disagreement between force and acceleration is not a problem specific to the IMU. Instead, the inconsistent behavior between subjects appears 'real' rather than a measurement error. The ability to reliably detect asymmetry in general does itself have potential uses however as seen in previous studies linking asymmetry to inefficient walking [19].

Gait Change in Response to Feedback

In response to real-time feedback of anterior acceleration via the IMU, it was found that subjects could consistently increase their COM anterior acceleration peaks, but decreasing these peaks was more difficult. Additionally, instructions to increase the peak for only one leg consistently produced a difference between legs but were inconsistent in terms of which leg was increased. Participants' success at increasing anterior acceleration bilaterally suggests that the feedback provided was capable of eliciting a potentially beneficial change. Whereas decreasing the anterior acceleration bilaterally proved more difficult, it is also the furthest from what actual treatment for hemiparetic gait would consist of. Furthermore, decreasing COM movement is likely an ineffective strategy for improving gait [23] and thus subjects were likely resistant to adopt such a strategy.

Of particular relevance to individuals with hemiparesis, this study attempted to have unimpaired individuals increase their anterior acceleration with only one limb. These unilateral increase effects did not always increase the intended leg's contribution to acceleration, though it is likely that this could be improved on. After subject testing was completed, it was found during analysis that the method of determining left or right step was unreliable when validated against the GRF-calculated heel strike data. Specifically, the algorithm used during subject testing checked whether the average frontal plane acceleration in the middle of positive anterior acceleration was positive or negative, but this was not found to be an accurate enough method. A more robust method for left vs right step determination with the IMU was developed after data collection and was based on a comparison of the average frontal plane acceleration between the current and previous areas of positive anterior acceleration. Not having this algorithm for data collections may have contributed to the observation that the target limb was not always the limb

that produced increased anterior acceleration. Due to time constraints all subject recordings had already been done, but the revised algorithm was validated in post-processing by accurately determining left or right peaks of the already collected data. Regardless, the data clearly demonstrated that participants were able to substantially increase the anterior acceleration associated with just one limb as well as increase the propulsive forces of a single limb. The ability to increase propulsion in only one leg could be directly applicable to hemiparetic rehabilitation by tapping into patients' propulsive reserve for their hemiparetic legs.

Others have used propulsive feedback to demonstrate that individuals post-stroke [25] and healthy older adults [14] have the ability to increase propulsive forces. Whereas these prior studies were done on force-instrumented treadmills, the work in this study suggests that appropriate feedback is possible during walking overground. The clinical utility of this approach increases generalizability and allows individuals to practice walking in environments that are more convenient or familiar to them.

Limitations

Although this study showed partial support for its hypotheses, there are two unresolved questions with the study's result. The first is the disagreement between acceleration and force data, particularly in the asymmetric gait conditions. Peak force in this study was found as the peak of the anterior ground reaction force in the leg performing push off for that step. A potential solution to this issue is looking at the combined A/P force for both legs rather than for individual legs, which would change the timing and height of the peaks somewhat. It is also possible that measuring only anterior force from the IMU's perspective is insufficient for accurately determining force due to rotation of the pelvis causing the IMU's anterior to be off-axis from the body's anterior. The second of these concerns is the inconsistent leg behavior

between patients for the unilateral increase feedback. As mentioned previously, a possible reason for this is the flawed left/right step classification technique that was used for subject feedback before it could be validated against motion capture data. A further limitation of this study is the fact that it induced an impairment using a leg brace instead of testing in post-stroke individuals with hemiparesis. While this has been used in previous literature [26], [27], [31] and avoided confounding issues associated with a disordered nervous system, it is likely that the simulated gait pattern was not the same as from stroke patients. The abnormal movement associated with an actual paretic leg would likely be less subtle than the effect of a brace however, so the fact that this study could detect the braced asymmetry suggests that it could also detect hemiparetic asymmetry. Another factor that might have affected the result is the use of a single pelvis marker to approximate center of mass, though this has been established as a relatively close estimate [28], [29].

Future Work

The force correlation results seen in this study might be improved through methods discussed above such as measuring force integrals rather than peaks for steps or combining the curves of both legs. Use of the improved left/right classification algorithm has potential to improve feedback results, as does more detailed analysis of the body's center of mass behavior. In addition to addressing these concerns, the IMU measurement demonstrated in this study can be applied to over ground studies. Using a similar sensor system, fully portable feedback could be provided using audio feedback or visual feedback via augmented reality glasses. An example of such a potential feedback method would be indicating the amount steps are above or below a target by playing one of two sounds at a volume based on distance from the target.

Conclusions

This study found strong to moderate correlations between COM anterior acceleration as measured by the IMU with anterior acceleration as measured by a motion capture system, but only moderate to weak correlation with anterior propulsive force as measured by force plates in an instrumented treadmill. Induced asymmetry between legs was reliably detected, though the behavior of which had decreased acceleration measurements did not match the unilateral decrease in force caused by the leg brace. Visual feedback was used to successfully produce increased acceleration peaks and asymmetric gait but was unsuccessful at decreased acceleration peaks and at determining which leg increased in the asymmetric case. The findings of this study could be used to create IMU based feedback systems for the detection of gait asymmetry or therapeutic training programs usable outside of a clinical environment.

REFERENCES

[1] K. S. Bruno A, "Motor Recovery In Stroke," *eMedicine*, 09-Dec-2004. [Online]. Available: https://emedicine.medscape.com/article/324386-overview.

[2] N. CDC, "Underlying Cause of Death 1999-2016," *cdc.gov*, 20-Dec-2017. [Online]. Available: https://wonder.cdc.gov/wonder/help/ucd.html.

[3] D. Mozaffarian *et al.*, "Heart disease and stroke statistics--2015 update: a report from the American Heart Association," *Circulation*, vol. 131, no. 4, pp. e29-322, Jan. 2015.

[4] V. L. Roger *et al.*, "Heart Disease and Stroke Statistics—2012 Update: A Report From the American Heart Association," *Circulation*, p. CIR.0b013e31823ac046, Jan. 2011.

[5] E. G. Gonzalez and P. J. Corcoran, "Chapter 16 - Energy Expenditure During Ambulation," in *The Physiological Basis of Rehabilitation Medicine (Second Edition)*, Butterworth-Heinemann, 1994, pp. 413–446.

[6] R. L. Waters and S. Mulroy, "The energy expenditure of normal and pathologic gait," *Gait Posture*, vol. 9, no. 3, pp. 207–231, Jul. 1999.

[7] A. Hudakova and A. Hornakova, "Mobility and quality of life in elderly and geriatric patients," *Int J Nurs Midwifery*, vol. 3, Jan. 2011.

[8] H. Shimada *et al.*, "Predictive validity of the classification schema for functional mobility tests in instrumental activities of daily living decline among older adults," *Arch. Phys. Med. Rehabil.*, vol. 91, no. 2, pp. 241–246, Feb. 2010.

[9] S. J. Page, D. R. Gater, and P. Bach-Y-Rita, "Reconsidering the motor recovery plateau in stroke rehabilitation," *Arch. Phys. Med. Rehabil.*, vol. 85, no. 8, pp. 1377–1381, Aug. 2004.

[10] R. F. Macko, G. V. Smith, C. L. Dobrovolny, J. D. Sorkin, A. P. Goldberg, and K. H. Silver, "Treadmill training improves fitness reserve in chronic stroke patients," *Arch. Phys. Med. Rehabil.*, vol. 82, no. 7, pp. 879–884, Jul. 2001.

[11] R. F. Macko *et al.*, "Treadmill exercise rehabilitation improves ambulatory function and cardiovascular fitness in patients with chronic stroke: a randomized, controlled trial," *Stroke*, vol. 36, no. 10, pp. 2206–2211, Oct. 2005.

[12] A. Middleton, C. H. Braun, M. D. Lewek, and S. L. Fritz, "Balance impairment limits ability to increase walking speed in individuals with chronic stroke," *Disabil. Rehabil.*, vol. 39, no. 5, pp. 497–502, Feb. 2017.

[13] D. S. Reisman, R. Wityk, K. Silver, and A. J. Bastian, "Split-Belt Treadmill Adaptation Transfers to Overground Walking in Persons Poststroke," *Neurorehabil. Neural Repair*, vol. 23, no. 7, pp. 735–744, Sep. 2009.

[14] J. R. Franz, M. Maletis, and R. Kram, “Real-time feedback enhances forward propulsion during walking in old adults,” *Clin. Biomech.*, vol. 29, no. 1, pp. 68–74, Jan. 2014.

[15] J. Wang, C. P. Hurt, C. E. Capo-Lugo, and D. A. Brown, “Characteristics of horizontal force generation for individuals post-stroke walking against progressive resistive forces,” *Clin. Biomech.*, vol. 30, no. 1, pp. 40–45, Jan. 2015.

[16] C. K. Balasubramanian, M. G. Bowden, R. R. Neptune, and S. A. Kautz, “Relationship between step length asymmetry and walking performance in subjects with chronic hemiparesis,” *Arch. Phys. Med. Rehabil.*, vol. 88, no. 1, pp. 43–49, Jan. 2007.

[17] R. G. Ellis, K. C. Howard, and R. Kram, “The metabolic and mechanical costs of step time asymmetry in walking,” *Proc. R. Soc. B Biol. Sci.*, vol. 280, no. 1756, Apr. 2013.

[18] D. J. Farris, A. Hampton, M. D. Lewek, and G. S. Sawicki, “Revisiting the mechanics and energetics of walking in individuals with chronic hemiparesis following stroke: from individual limbs to lower limb joints,” *J. NeuroEngineering Rehabil.*, vol. 12, p. 24, Feb. 2015.

[19] L. N. Awad, J. A. Palmer, R. T. Pohlig, S. A. Binder-Macleod, and D. S. Reisman, “Walking Speed and Step Length Asymmetry Modify the Energy Cost of Walking After Stroke,” *Neurorehabil. Neural Repair*, vol. 29, no. 5, pp. 416–423, Jun. 2015.

[20] K. E. Zelik and P. G. Adamczyk, “A unified perspective on ankle push-off in human walking,” *J. Exp. Biol.*, vol. 219, no. Pt 23, pp. 3676–3683, 01 2016.

[21] P. Esser, H. Dawes, J. Collett, M. G. Feltham, and K. Howells, “Validity and inter-rater reliability of inertial gait measurements in Parkinson’s disease: A pilot study,” *J. Neurosci. Methods*, vol. 205, no. 1, pp. 177–181, Mar. 2012.

[22] F. Massaad, T. M. Lejeune, and C. Detrembleur, “Reducing the energy cost of hemiparetic gait using center of mass feedback: a pilot study,” *Neurorehabil. Neural Repair*, vol. 24, no. 4, pp. 338–347, May 2010.

[23] K. E. Gordon, D. P. Ferris, and A. D. Kuo, “Metabolic and mechanical energy costs of reducing vertical center of mass movement during gait,” *Arch. Phys. Med. Rehabil.*, vol. 90, no. 1, pp. 136–144, Jan. 2009.

[24] M. G. Bowden, C. K. Balasubramanian, R. R. Neptune, and S. A. Kautz, “Anterior-posterior ground reaction forces as a measure of paretic leg contribution in hemiparetic walking,” *Stroke*, vol. 37, no. 3, pp. 872–876, Mar. 2006.

[25] K. Genthe, C. Schenck, S. Eicholtz, L. Zajac-Cox, S. Wolf, and T. M. Kesar, “Effects of real-time gait biofeedback on paretic propulsion and gait biomechanics in individuals post-stroke,” *Top. Stroke Rehabil.*, pp. 1–8, Feb. 2018.

[26] M. D. Lewek, A. J. Osborn, and C. J. Wutzke, “The influence of mechanically and physiologically imposed stiff-knee gait patterns on the energy cost of walking,” *Arch. Phys. Med. Rehabil.*, vol. 93, no. 1, pp. 123–128, Jan. 2012.

[27] C. J. Wutzke, G. S. Sawicki, and M. D. Lewek, “The influence of a unilateral fixed ankle on metabolic and mechanical demands during walking in unimpaired young adults,” *J. Biomech.*, vol. 45, no. 14, pp. 2405–2410, Sep. 2012.

[28] F. Yang and Y.-C. Pai, “Can sacral marker approximate center of mass during gait and slip-fall recovery among community-dwelling older adults?,” *J. Biomech.*, vol. 47, no. 16, pp. 3807–3812, Dec. 2014.

[29] M. J. Floor-Westerdijk, H. M. Schepers, P. H. Veltink, E. H. F. van Asseldonk, and J. H. Buurke, “Use of inertial sensors for ambulatory assessment of center-of-mass displacements during walking,” *IEEE Trans. Biomed. Eng.*, vol. 59, no. 7, pp. 2080–2084, Jul. 2012.

[30] P. O. Riley, G. Paolini, U. D. Croce, K. W. Paylo, and D. C. Kerrigan, “A kinematic and kinetic comparison of overground and treadmill walking in healthy subjects,” *Gait Posture*, vol. 26, no. 1, pp. 17–24, Jun. 2007.

[31] J. A. Nessler, V. Gutierrez, J. Werner, and A. Punsalan, “Side by side treadmill walking reduces gait asymmetry induced by unilateral ankle weight,” *Hum. Mov. Sci.*, vol. 41, pp. 32–45, Jun. 2015.

APPENDICES

Appendix A

Live Feedback and Recording Code

```
global versionNumber
versionNumber = "No Bluetooth v1.5" # v 1.4 was used for I_C1-I_C9

import serial
import threading
import numpy as np
import Queue
import Tkinter as tk
import tkMessageBox
import time, datetime
import os.path
import winsound
import operator

import threespace as ts_api
##import threespace_api as ts_api
import matplotlib
import matplotlib.pyplot as plot

global runButtonOn
runButtonOn = False
##global doTareNextStep # Declared elsewhere do to order of operations
##doTareNextStep = True
global tareInfo
global tareInfoAngle
global peakCalibValue
peakCalibValue = 0.01
global programIsClosing
programIsClosing = False
global doUpdateTargetLines
doUpdateTargetLines = False
global segIndicator
segIndicator = "green"
global writeFileNameBase
writeFileNameBase = "threeSpaceOutput"
useStepSounds = False # Plays a different sound on left vs right steps
usePlotCurrentMaxMinLines = False # current and previous max/min lines
useSidePlot = False
useSideBars = False # side to side bars

class SerialThread(threading.Thread):
    ### Step input definition
    def __init__(self, peakQueue, peakTimeQueue, currentQueue, lastStepValsList, dataStoreT):
        threading.Thread.__init__(self)
        self.ISCONNECTED = 0
        self.peakQueue = peakQueue
```

```
        self.peakTimeQueue = peakTimeQueue
        self.currentQueue = currentQueue
        self.lastStepValsList = lastStepValsList
        self.dataStoreT = dataStoreT

        #Set the BaudRate
        self.baudrate = 115200
        #set the Duration
        # zero = infinite.
        self.usrDur = 0
        #Set the Interval, in Microseconds (1 e -6 sec)
        self.usrInt = 1e-2 * 1e6 # 10000

        self.connectDevice()

    def connectDevice(self):
        device_list = ts_api.getComPorts(filter=ts_api.TSS_FIND_BT)
        print(device_list)
        self.bt_device = 0
        if len(device_list) > 0:
            com_port = device_list[0]
            #create a Bluetooth device, and communicate with it on com port 'com_port'
            self.bt_device = ts_api.TSBTSensor(com_port=com_port)
            self.bt_device.stopStreaming()
            self.bt_device.stopRecordingData()
            self.bt_device.updateCurrentTimestamp(time.clock())
            self.bt_device.setStreamingTiming(interval=self.usrInt, duration=self.usrDur, delay=0)
            self.bt_device.setStreamingSlots(slot0='getCorrectedAccelerometerVector')
            self.bt_device.setFilterMode(mode=2)
            self.bt_device.baudrate = self.baudrate
            self.bt_device.commitSettings()
            self.ISCONNECTED = 1
            print("Device connected.")

            #dt = datetime.datetime
            #dt = dt.now()
        else:
            print("There were no sensors detected.")
            self.ISCONNECTED = 0

    def disconnectDevice(self):
        self.bt_device.stopStreaming()
        self.bt_device.stopRecordingData()
        #Close the sensor communication
        self.bt_device.close()
        self.ISCONNECTED = 0
        print("Device Disconnected.")

    def run(self): # Called without being visibly called from somewhere
        if(self.bt_device != 0):
            self.readDevice()
```

```
    else:
        print("Sensor not found.")

def readDevice(self):

    #Tare the sensor here.
    self.bt_device.tareWithCurrentOrientation()

    self.bt_device.startStreaming()
    self.bt_device.startRecordingData()
    #Setup data arrays
    tStart = 0 # Start time
    tPrevious = 0
    tAll = [] # All the time points
    yAll = [] # All the forward points
    yAllRaw = [] # Raw y points for smoothing
    xAllRaw = []
    ySinceZero = [] # The forward points since the last zero cross
    xSumSinceZero = 0 # Sum of x values since last 0
    xListSinceZero = [] # List of x values since last 0 ## Part of the outdated R/L system
    # yMax = 0 # y max (temporary var used in loop)
    # yMin = 0 # y min (temporary var used in loop)
    prevXAvgForYMax = 0
    prevYPeakWasMax = False # If the previous peak was a y maximum
    prevYPeakWasMin = False # If the previous peak was a y minimum
    barAvgLength = 3 # How many steps (from current back) the left average / right peak averages go
    pastLeftPeaks = [0] * barAvgLength
    pastRightPeaks = [0] * barAvgLength
    prevYForMean = []
    prevYForMeanLenMax = 2000
    thisXCompareSum = 0 # New X side finder
    thisXCompareCount = 0
    prevXComapreSum = 0
    prevXCompareCount = 0
    thisXCompareSide = ""
    xAvg = 0
    lastSideYAvg = 0
    #print(dir(self.bt_device)) # Print all functions of self.bt_device
    while self.ISCONNECTED and programIsClosing == False:
        global doTareNextStep
        if doTareNextStep:
            tareInfo = self.bt_device.getLatestStreamData(timeout=10)
            doTareNextStep = False
            print('-Device Zeroed-')

        if runButtonOn:
            sensInfo = self.bt_device.getLatestStreamData(timeout=10)
            sensInfoAngle = self.bt_device.getTaredOrientationAsEulerAngles()

            if sensInfo is not None:
                #Gets stream data from the sensor and prints out HH:MM:SS:MS, X, Y
```

```
##                 diagnosticOutStr = str(datetime.datetime.now().hour) +":"+
str(datetime.datetime.now().minute)
##                 diagnosticOutStr = diagnosticOutStr + ":"+str(datetime.datetime.now().second)+"."+
str(sensInfo[0])
##                 diagnosticPrint(1, diagnosticOutStr)
              timeNow = matplotlib.dates.date2num(datetime.datetime.now())

              if tStart == 0:
                 tStart = timeNow
              timeNow = timeNow - tStart

              if len(yAll) > 0:  #after the first time through
                 xAcc = sensInfo[1][0] - tareInfo[1][0] # Acceleration info
                 yAcc = -(sensInfo[1][1] - tareInfo[1][1])
                 zAcc = sensInfo[1][2] - tareInfo[1][2]
##                  xAcc = sensInfoAngle[0] # Orientation info (not currently working)
##                  yAcc = sensInfoAngle[1]
##                  zAcc = sensInfoAngle[2]

                 smoothThetaY = 20
                 backPointCountY = 10

                 smoothThetaX = 100
                 backPointCountX = 30

                 # Note: y is front to back
                 # x is side to side
                 plotXAsY = False
##                  plotXAsY = True
                 if(plotXAsY == True):
                    yAcc = xAcc
                    smoothThetaY = smoothThetaX
                    backPointCountY = backPointCountX
##                  print(plotXAsY)

                 yAccRaw = yAcc
                 yAllRaw.append(yAcc) # This is used in smoothing
                 xAllRaw.append(xAcc)

                 tAll.append(timeNow)

                 ### Y Smooothing
                 backPointWeightSum = 0
                 backPointWeightedValueSum = 0
                 currentPointInc = len(yAllRaw) -1
                 if(currentPointInc >= backPointCountY):
                    for backPointOffset in range(backPointCountY +1): # +1 because it starts at 0 counting
up
                       backPointInc = currentPointInc - backPointOffset
                       timeDiff = timeNow - tAll[backPointInc]
                       backPointWeight = np.exp(smoothThetaY * timeDiff)
```

```
                    backPointWeightSum = backPointWeightSum + backPointWeight
                    backPointWeightedValueSum = backPointWeightedValueSum + backPointWeight *
yAllRaw[backPointInc]
                yAcc = backPointWeightedValueSum / backPointWeightSum

            ### X Smooothing
            backPointWeightSum = 0
            backPointWeightedValueSum = 0
            currentPointInc = len(xAllRaw) -1
            if(currentPointInc >= backPointCountX):
                for backPointOffset in range(backPointCountX +1): # +1 because it starts at 0 counting
up
                    backPointInc = currentPointInc - backPointOffset
                    timeDiff = timeNow - tAll[backPointInc]
                    backPointWeight = np.exp(smoothThetaX * timeDiff)
                    backPointWeightSum = backPointWeightSum + backPointWeight
                    backPointWeightedValueSum = backPointWeightedValueSum + backPointWeight *
xAllRaw[backPointInc]
                xAcc = backPointWeightedValueSum / backPointWeightSum

            prevYForMean.append(yAcc) # Subtracts long running y average
            if(len(prevYForMean) > prevYForMeanLenMax):
                prevYForMean.pop(0)
            yAccDemeaned = yAcc - sum(prevYForMean)/len(prevYForMean)

            yAll.append(yAccDemeaned)
            ySinceZero.append(yAccDemeaned)

##              xSumSinceZero = xSumSinceZero + xAcc
##              xListSinceZero.append(xAcc)

            if(yAccDemeaned < 0):
                thisXCompareSum = thisXCompareSum + xAcc
                thisXCompareCount = thisXCompareCount + 1
            else:
                if(thisXCompareCount > 0):
                    if(prevXCompareCount > 0):
                        xAvg = thisXCompareSum / float(thisXCompareCount)
                        if(xAvg > prevXComapreSum / float(prevXCompareCount)):
                            thisXCompareSide = 2 # 2 for right
                        else:
                            thisXCompareSide = 1 # 1 for left
                    prevXComapreSum = thisXCompareSum
                    prevXCompareCount = thisXCompareCount
                    thisXCompareSum = 0
                    thisXCompareCount = 0

            self.lastStepValsList.append(timeNow)
            self.lastStepValsList.append([yAccDemeaned, xAcc])
            if(len(self.lastStepValsList) >= 20): # This is done to bundle the output, since queue
getting is kind of slow
```

```
                    self.currentQueue.put(self.lastStepValsList)
                    self.lastStepValsList = []
##                  self.currentQueue.put([timeNow, yAccDemeaned])
##                  self.dataStoreT.put([timeNow, xAcc, -yAccDemeaned, zAcc])
                y_if_Peak = 0 # set in peak detection then made part of dataStoreT
                L_or_R_peak = ""
                # Having this be after peaks

                if (yAll[len(yAll) - 2] < 0 and yAccDemeaned > 0) or (yAll[len(yAll) - 2] > 0 and
yAccDemeaned < 0):
                    #if the point crosses 0:
##                      self.currentQueue.put(self.lastStepValsList)
##                      self.lastStepValsList = []

                    if (len(ySinceZero)>5): # If y since crossing is long enough
                        if (yAll[len(yAll) - 2] < 0 and yAccDemeaned > 0): # crosses from neg to pos
(looking for a min)
                            peakYInd, peakY = min(enumerate(ySinceZero), key=operator.itemgetter(1))
                        else:                                   # crosses from pos to neg (looking for a max)
                            peakYInd, peakY = max(enumerate(ySinceZero), key=operator.itemgetter(1))
                        global peakCalibValue
                        if(abs(peakY) > peakCalibValue * 0.5):

                            leftOrRightVal = 0
                            lastSideYAvg = 0
                            if(peakY < 0): # True when forward
                                leftOrRightVal = thisXCompareSide
                                if(leftOrRightVal == 1): # 1 for left
                                    pastLeftPeaks.append(peakY)
                                    del pastLeftPeaks[0]
                                    if(pastLeftPeaks[0] != 0):
                                        lastSideYAvg = sum(pastLeftPeaks) / float(len(pastLeftPeaks))
                                elif(leftOrRightVal == 2): # 2 for right
                                    pastRightPeaks.append(peakY)
                                    del pastRightPeaks[0]
                                    if(pastRightPeaks[0] != 0):
                                        lastSideYAvg = sum(pastRightPeaks) / float(len(pastRightPeaks))
##                                  lastSideYAvg = peakY # Simple y output for testing
                            self.peakQueue.put([peakY, xAvg, leftOrRightVal, lastSideYAvg, 0])
                            self.peakTimeQueue.put(timeNow)
                            diagnosticPrint(2, "y Min/Max added")

                    else:  # If y since crossing isn't long enough
                        diagnosticPrint(2, "nothing here")
                    del ySinceZero[:]
                self.dataStoreT.put([timeNow, xAcc, -yAcc, zAcc]) # Y Inverted due to axis direction
issues
##                  else:#did not cross 0, so this is remaining either positive or negative
##                      tAll.append(timeNow)
            else: #first time through:
                xAcc = sensInfo[1][0] - tareInfo[1][0]
```

```
                yAcc = sensInfo[1][1] - tareInfo[1][1]
                zAcc = sensInfo[1][2] - tareInfo[1][2]
                yAll.append(yAcc)
                tAll.append(timeNow)
                ySinceZero.append(yAcc)
##                  tTemp.append(timeNow)
                #print(ySinceZero)
        self.disconnectDevice()

class App(tk.Tk):
    def __init__(appRoot):
        tk.Tk.__init__(appRoot)
        global versionNumber
        appRoot.title("IMU recorder "+ versionNumber)
##      appRoot.geometry("1360x750")
##      appRoot.geometry("680x700")
        appRoot.oscWidth = 600
        appRoot.barCanvasWidth = 250

##      print(str(appRoot.oscWidth + appRoot.barCanvasWidth))
        appRoot.geometry(str(appRoot.oscWidth + appRoot.barCanvasWidth) + "x700")
        gridRow = 0

        #frameLabel = tk.Frame(appRoot, padx=400, pady =400)
        #appRoot.text = tk.Text(frameLabel, wrap='word', font='TimesNewRoman 37',
                    # bg=appRoot.cget('bg'), relief='flat')
        #frameLabel.pack()
        #gridRow += 1
        #appRoot.text.pack()
        #gridRow += 1

        appRoot.yScaleFactor = 300
        appRoot.yScaleStringVar = tk.StringVar()
        appRoot.yScaleStringVar.set(str(appRoot.yScaleFactor))

        appRoot.leftTargetVal = 0
        appRoot.leftTargetStringVar = tk.StringVar()
        appRoot.leftTargetStringVar.set(str(appRoot.leftTargetVal))
        appRoot.rightTargetVal = 0
        appRoot.rightTargetStringVar = tk.StringVar()
        appRoot.rightTargetStringVar.set(str(appRoot.rightTargetVal))

##      appRoot.yPeakCalibFactor = 0
        global peakCalibValue
##      peakCalibValue = 0
        appRoot.yPeakCalibStringVar = tk.StringVar()
##      appRoot.yPeakCalibStringVar.set(str(appRoot.yPeakCalibFactor))
        appRoot.yPeakCalibStringVar.set(str(peakCalibValue))

        global writeFileNameBase
        appRoot.yWriteFileNameStringVar = tk.StringVar()
```

```
        appRoot.yWriteFileNameStringVar.set(str(writeFileNameBase))

        appRoot.oscHeight = 450 # oscWidth declared earlier
        appRoot.oscTimeToCross = 5
        appRoot.oscCanvas = tk.Canvas(appRoot, width=appRoot.oscWidth, height=appRoot.oscHeight,
bg='black')
        appRoot.oscCanvas.grid(row=gridRow, columnspan=2)
        appRoot.oscCanvas.create_line(0, appRoot.oscHeight/2, appRoot.oscWidth, appRoot.oscHeight/2,
fill="green", dash=(7,1))

        appRoot.barWidth = 50
        appRoot.barYBase = appRoot.oscHeight/2
        appRoot.xMin0 = appRoot.barCanvasWidth * 1/4 - appRoot.barWidth/2
##        appRoot.xMin1 = appRoot.xMin0 + appRoot.barWidth
        appRoot.xMax0 = appRoot.barCanvasWidth * 3/4 - appRoot.barWidth/2
##        appRoot.xMax1 = appRoot.xMax0 + appRoot.barWidth
        appRoot.xCurrent0 = appRoot.xMax0 + appRoot.barWidth
        appRoot.xCurrent1 = appRoot.xCurrent0 + appRoot.barWidth
        appRoot.barCanvas = tk.Canvas(appRoot, width=appRoot.barCanvasWidth,
height=appRoot.oscHeight, bg='white')
        appRoot.barCanvas.grid(row=gridRow, column=3, columnspan=2)
        gridRow += 1

        appRoot.startButton = tk.Button(appRoot, text="-Init-", command=appRoot.startButtonCallback)
        appRoot.startButton.grid(row=gridRow, column=0, columnspan=2)
        gridRow += 1

        appRoot.tareButton = tk.Button(appRoot, text="Tare", command=appRoot.tareButtonCallback)
        appRoot.tareButton.grid(row=gridRow, column=0, columnspan=2)
        gridRow += 1

        appRoot.scaleText = tk.Label(appRoot, text="Y Display Scaling:")
        appRoot.scaleText.grid(row=gridRow, column=0, sticky='E')
        appRoot.scaleEntry = tk.Entry(appRoot, width=10, textvariable=appRoot.yScaleStringVar)
        appRoot.scaleEntry.bind('<Return>', appRoot.scaleEntryCallback)
        appRoot.scaleEntry.grid(row=gridRow, column=1, sticky='W')
        gridRow += 1

        appRoot.leftTargetText = tk.Label(appRoot, text="Left Target:") # Target line input boxes
        appRoot.leftTargetText.grid(row=gridRow -1, column=3, sticky='E')
        appRoot.leftTargetEntry = tk.Entry(appRoot, width=10, textvariable=appRoot.leftTargetStringVar)
        appRoot.leftTargetEntry.bind('<Return>', appRoot.leftTargetEntryCallback)
        appRoot.leftTargetEntry.grid(row=gridRow, column=3, sticky='E')
        appRoot.rightTargetText = tk.Label(appRoot, text="Right Target:")
        appRoot.rightTargetText.grid(row=gridRow -1, column=4, sticky='W')
        appRoot.rightTargetEntry = tk.Entry(appRoot, width=10,
textvariable=appRoot.rightTargetStringVar)
        appRoot.rightTargetEntry.bind('<Return>', appRoot.rightTargetEntryCallback)
        appRoot.rightTargetEntry.grid(row=gridRow, column=4, sticky='W')
```

```
        appRoot.scaleText = tk.Label(appRoot, text="Calibration Peak Height (ignotes peaks with abs below
50% of):")
        appRoot.scaleText.grid(row=gridRow, column=0, sticky='E')
        appRoot.scaleEntry = tk.Entry(appRoot, width=10, textvariable=appRoot.yPeakCalibStringVar)
        appRoot.scaleEntry.bind('<Return>', appRoot.peakCalibEntryCallback)
        appRoot.scaleEntry.grid(row=gridRow, column=1, sticky='W')
        gridRow += 1

        appRoot.fileNameText = tk.Label(appRoot, text="Output file name:")
        appRoot.fileNameText.grid(row=gridRow, column=0, sticky='E')
        appRoot.fileNameEntry = tk.Entry(appRoot, width=10,
textvariable=appRoot.yWriteFileNameStringVar)
        appRoot.fileNameEntry.bind('<Return>', appRoot.writeFileNameEntryCallback)
        appRoot.fileNameEntry.grid(row=gridRow, column=1, sticky='W')
        gridRow += 1

        appRoot.saveButton = tk.Button(appRoot, text="Save", command=appRoot.saveButtonCallback)
        appRoot.saveButton.grid(row=gridRow, column=0, columnspan=2)
        gridRow += 1

        appRoot.clearButton = tk.Button(appRoot, text="Clear Data",
command=appRoot.clearButtonCallback)
        appRoot.clearButton.grid(row=gridRow, column=0, columnspan=2)
        gridRow += 1

        appRoot.timeOutputHeaderLabel = tk.Label(appRoot, text="Time:")
        appRoot.timeOutputHeaderLabel.grid(row=gridRow, column=0, columnspan=2)
        gridRow += 1
        appRoot.timeOutputLabel = tk.Label(appRoot, text="-")
        appRoot.timeOutputLabel.grid(row=gridRow, column=0, columnspan=2)
        gridRow += 1

        appRoot.peakQueue = Queue.Queue()
        appRoot.peakTimeQueue = Queue.Queue()
        appRoot.currentQueue =  Queue.Queue()
        appRoot.lastStepValsList =  []
        appRoot.dataStoreT = Queue.Queue()
        ### Step input value setting
        thread = SerialThread(appRoot.peakQueue, appRoot.peakTimeQueue, appRoot.currentQueue,
appRoot.lastStepValsList, appRoot.dataStoreT)
        thread.start()

        # Initialize properties to pass between serial loops
        appRoot.previousTime = 0
        appRoot.startTime = 0
##          appRoot.currentLines = []
        appRoot.currentLineTimes = [];
        appRoot.currentPeakLineTimes = [];
        appRoot.previousCurrentHeight = appRoot.oscHeight / 2
        appRoot.previousCurrentSideHeight = appRoot.oscHeight / 2
```

```
        appRoot.process_serial() # Start loop
##      appRoot.height = 0

        global runButtonOn
        runButtonOn = False

        global doTareNextStep
        doTareNextStep = True

    def startButtonCallback(appRoot):
        global runButtonOn
        if runButtonOn:
            print('---Recording Off---')
            runButtonOn = False
            appRoot.startButton.config(text='Start Recording')
        else:
            print('---Recording On---')
            appRoot.startButton.config(text='Stop Recording')
            runButtonOn = True

    def tareButtonCallback(appRoot):
        global doTareNextStep
        doTareNextStep = True

    def scaleEntryCallback(appRoot, enterEvent):
        eventVal = enterEvent.widget.get()
        try:
            eventVal = float(eventVal)
        except:
            eventVal = 0
        if eventVal > 0:
            print("Scale set to "+ appRoot.yScaleStringVar.get())
            appRoot.yScaleFactor = eventVal
        else:
            print("Invalid Entry, scale not set")
            appRoot.yScaleStringVar.set(appRoot.yScaleFactor)

    def leftTargetEntryCallback(appRoot, enterEvent):
        eventVal = enterEvent.widget.get()
        try:
            eventVal = float(eventVal)
        except:
            eventVal = 0
        if eventVal != 0:
            print("Left target set to "+ appRoot.leftTargetStringVar.get())
            appRoot.leftTargetVal = eventVal
            global doUpdateTargetLines
        else:
            print("Invalid Entry; clearing line")
            appRoot.leftTargetStringVar.set(0)
```

```
        doUpdateTargetLines = True
    def rightTargetEntryCallback(appRoot, enterEvent):
        eventVal = enterEvent.widget.get()
        try:
            eventVal = float(eventVal)
        except:
            eventVal = 0
        if eventVal != 0:
            print("Right target set to "+ appRoot.rightTargetStringVar.get())
            appRoot.rightTargetVal = eventVal
            global doUpdateTargetLines
        else:
            print("Invalid Entry, right target not set")
            appRoot.rightTargetStringVar.set(0)
        doUpdateTargetLines = True

    def peakCalibEntryCallback(appRoot, enterEvent):
        eventVal = enterEvent.widget.get()
        try:
            eventVal = float(eventVal)
        except:
            eventVal = 0
        global peakCalibValue
        if eventVal > 0:
            print("Scale peak calibration to "+ appRoot.yPeakCalibStringVar.get())
##          appRoot.yPeakCalibFactor = eventVal
            peakCalibValue = eventVal
        else:
            print("Invalid Entry, peak calibration not set")
##          appRoot.yPeakCalibStringVar.set(appRoot.yPeakCalibFactor)
            appRoot.yPeakCalibStringVar.set(peakCalibValue)

    def writeFileNameEntryCallback(appRoot, enterEvent):
        eventVal = enterEvent.widget.get()
        try:
            eventVal = str(eventVal)
        except:
            eventVal = ""
        global writeFileNameBase
        if len(eventVal) > 0:
            print("Output file name set to "+ appRoot.yWriteFileNameStringVar.get())
            writeFileNameBase = eventVal
        else:
            print("Invalid Entry, output file name not set")
            appRoot.yWriteFileNameStringVar.set(writeFileNameBase)

    def saveButtonCallback(appRoot):
        global writeFileNameBase
        writeFileName = './'+ writeFileNameBase +'.csv'
        saveOptionDialog(appRoot, writeFileName)
```

```
    def clearButtonCallback(appRoot):
        if tkMessageBox.askokcancel("Clear Data", "Clear data log?"):
            appRoot.dataStoreT.queue.clear()

    def process_serial(appRoot):
        #appRoot.text.delete(1.0, 'end')
##          global runButtonOn
##          if runButtonOn == False and (appRoot.leftPeakCount > 0 or appRoot.rightPeakCount > 0):
##              print(runButtonOn)
##              if appRoot.leftPeakCount > 0:
##                  leftPeakAvg = appRoot.leftPeakSum / appRoot.leftPeakCount
##              else:
##                  leftPeakAvg = 0
##              if appRoot.rightPeakCount > 0:
##                  rightPeakAvg = appRoot.rightPeakSum / appRoot.rightPeakCount
##              else:
##                  rightPeakAvg = 0
##              print(leftPeakAvg, rightPeakAvg)

        global doUpdateTargetLines
        if doUpdateTargetLines: # Draws the target lines
            targetColor = "purple"
            leftTargetHeight = appRoot.barYBase - appRoot.leftTargetVal * appRoot.yScaleFactor
            rightTargetHeight = appRoot.barYBase - appRoot.rightTargetVal * appRoot.yScaleFactor
            appRoot.barCanvas.delete(appRoot.barCanvas.find_withtag("leftTargetLine")) # Bar plot target
lines
            if(appRoot.leftTargetVal != 0):
                appRoot.barCanvas.itemconfig(appRoot.barCanvas.find_withtag("leftTargetLine"), dash=(4,
4), tags="leftTargetLine")
                appRoot.barCanvas.create_line(0, leftTargetHeight, appRoot.barCanvasWidth/2,
leftTargetHeight, fill=targetColor, tags="leftTargetLine")
            appRoot.barCanvas.delete(appRoot.barCanvas.find_withtag("rightTargetLine"))
            if(appRoot.rightTargetVal != 0):
                appRoot.barCanvas.itemconfig(appRoot.barCanvas.find_withtag("rightTargetLine"), dash=(4,
4), tags="rightTargetLine")
                appRoot.barCanvas.create_line(appRoot.barCanvasWidth/2, rightTargetHeight,
appRoot.barCanvasWidth, rightTargetHeight, fill=targetColor, tags="rightTargetLine")
            doUpdateTargetLines = False

##          oscWidth = int(appRoot.oscCanvas.cget('width'))
        if appRoot.currentQueue.qsize():
            try:
##                  print(appRoot.currentQueue.qsize())
                currentPosAndTime = appRoot.currentQueue.get()
##                  print(appRoot.currentQueue.qsize())
##                  print("**********")
##                  print(currentPosAndTime)
            except Queue.Empty:
                print("Time Queue Error")
                print(Queue.Empty)
                pass
```

```
##          print(len(currentPosAndTime))

        currentTime = currentPosAndTime[0]
        currentHeight = currentPosAndTime[1][0]
        currentSidePos = currentPosAndTime[1][1] # aaa
##          currentTimeRaw = currentTime
        if appRoot.startTime == 0:
            appRoot.startTime = currentTime
##              print(currentTime - appRoot.startTime)
        currentTime = (currentTime - appRoot.startTime) * 100000
        if appRoot.peakTimeQueue.qsize():
            lastPeakTime = appRoot.peakTimeQueue.get()
            lastPeakTime = (lastPeakTime - appRoot.startTime) * 100000
            appRoot.peakTimeQueue.queue.clear()
        else:
            lastPeakTime = currentTime

        if appRoot.peakQueue.qsize(): # This triggers when a peak in forward velocity is found
            try:
                peakDataObj = appRoot.peakQueue.get() # [height, x average, (0 if min, 1 if left max, 2 if
right max)]
                peakHeight = peakDataObj[0]
                peakBarHeight = appRoot.barYBase + peakHeight * appRoot.yScaleFactor

                if(peakHeight > 0): # Peak is a Max (forward) / Red
                    peakColor = "red"
                    minOrMaxRect = "maxRect"
                    peakBarX0 = appRoot.xMax0
                    peakLine0Tag = "maxLine0"
                    peakLine1Tag = "maxLine1"
                else: ###################### Peak is a Min (backward) / Blue
                    peakColor = "blue"
                    minOrMaxRect = "minRect"
                    peakBarX0 = appRoot.xMin0
                    peakLine0Tag = "minLine0"
                    peakLine1Tag = "minLine1"
                peakBarX1 = peakBarX0 + appRoot.barWidth

##                  appRoot.barCanvas.delete(appRoot.barCanvas.find_withtag(minOrMaxRect)) # Min/Max
height bars
##                  appRoot.barCanvas.create_rectangle(peakBarX0, appRoot.barYBase, peakBarX1,
peakBarHeight, fill=peakColor, tags=minOrMaxRect)

                if(usePlotCurrentMaxMinLines):
                    appRoot.oscCanvas.delete(appRoot.oscCanvas.find_withtag(peakLine1Tag)) # Peak lines
for the oscilloscope plot
                    appRoot.oscCanvas.itemconfig(appRoot.oscCanvas.find_withtag(peakLine0Tag), dash=(4,
4), tags=peakLine1Tag)
                    appRoot.oscCanvas.create_line(0, peakBarHeight, appRoot.oscWidth, peakBarHeight,
fill=peakColor, tags=peakLine0Tag)
```

```
                offsetX = appRoot.oscWidth - (currentTime - lastPeakTime) * appRoot.oscWidth /
appRoot.oscTimeToCross
                currentLine = appRoot.oscCanvas.create_line(offsetX, appRoot.barYBase, offsetX,
peakBarHeight, fill=peakColor, tags="oscLineHorizontal")
                appRoot.currentPeakLineTimes.append(lastPeakTime)

            lastSideYAvg = peakDataObj[3] # average of the last n peaks based on number of previous
steps
            if(lastSideYAvg != 0):
                avgBarHeight = appRoot.barYBase + lastSideYAvg * appRoot.yScaleFactor
            else:
                avgBarHeight = 0
##              avgBarHeight = peakBarHeight

            xAvgColor = peakColor
            if(peakDataObj[2] > 0):
                if(peakDataObj[2] == 1): # Makes a different noise on max if left/right was positive or
negative
                    if(useStepSounds):
                        winsound.PlaySound("SystemExclamation", winsound.SND_ASYNC) # Left
                    xAvgColor = "Orange"
##                      minMaxBarHeight = appRoot.barYBase - peakHeight * appRoot.yScaleFactor # -
instead of +
                    minMaxBarHeight = avgBarHeight
                    leftOrRightRect = "leftRect"
                    peakBarX0 = appRoot.xMin0
                    peakLine0Tag = "leftLine0"
                    peakLine1Tag = "leftLine1"
                    peakTextTag = "leftText"
                else:
                    if(useStepSounds):
                        winsound.PlaySound("SystemHand", winsound.SND_ASYNC) # Right
                    xAvgColor = "Grey"
                    minMaxBarHeight = avgBarHeight
                    leftOrRightRect = "rightRect"
                    peakBarX0 = appRoot.xMax0
                    peakLine0Tag = "rightLine0"
                    peakLine1Tag = "rightLine1"
                    peakTextTag = "rightText"
                peakBarX1 = peakBarX0 + appRoot.barWidth

                if(lastSideYAvg != 0):
                    appRoot.barCanvas.delete(appRoot.barCanvas.find_withtag(leftOrRightRect)) # avg
height bars
                    appRoot.barCanvas.create_rectangle(peakBarX0, appRoot.barYBase, peakBarX1,
minMaxBarHeight, fill=xAvgColor, tags=leftOrRightRect)

                appRoot.barCanvas.delete(appRoot.barCanvas.find_withtag(peakLine1Tag)) # Peak lines
for the bar plot
                appRoot.barCanvas.itemconfig(appRoot.barCanvas.find_withtag(peakLine0Tag), dash=(4,
4), tags=peakLine1Tag)
```

```
                appRoot.barCanvas.create_line(0, minMaxBarHeight, appRoot.oscWidth,
minMaxBarHeight, fill=xAvgColor, tags=peakLine0Tag)

                barText = str(round(-lastSideYAvg, 2))
                textX = peakBarX0 + appRoot.barWidth/2
                appRoot.barCanvas.delete(appRoot.barCanvas.find_withtag(peakTextTag)) # Text for
average magnitude
                appRoot.barCanvas.create_text(textX, minMaxBarHeight-12, fill="black", font="Times 20
bold", text=barText, tags=peakTextTag)

                # y peak height horizontal line color coded to side (uses peak height instead of average
height)
                offsetX = 5 + appRoot.oscWidth - (currentTime - lastPeakTime) * appRoot.oscWidth /
appRoot.oscTimeToCross
                currentLine = appRoot.oscCanvas.create_line(offsetX + 20, peakBarHeight, offsetX - 20,
peakBarHeight, fill=xAvgColor, tags="oscLineHorizontal")
                appRoot.currentPeakLineTimes.append(lastPeakTime)

             if(useSideBars == True): # This used x avg
                sideBarHeight = appRoot.barYBase + peakDataObj[1] * appRoot.yScaleFactor #
horizontal line for x avg
                offsetX = 5 + appRoot.oscWidth - (currentTime - lastPeakTime) * appRoot.oscWidth /
appRoot.oscTimeToCross
                currentLine = appRoot.oscCanvas.create_line(offsetX + 20, sideBarHeight, offsetX - 20,
sideBarHeight, fill="purple", tags="oscLineHorizontal")
                appRoot.currentPeakLineTimes.append(lastPeakTime)

   ##           peakBarHeightInv = appRoot.barYBase - peakHeight * appRoot.yScaleFactor
   ##           currentLine = appRoot.oscCanvas.create_line(offsetX, appRoot.barYBase, offsetX,
peakBarHeightInv, fill="yellow", tags="oscLineHorizontal")
   ##           appRoot.currentPeakLineTimes.append(lastPeakTime)

             appRoot.peakQueue.queue.clear()
             appRoot.peakTimeQueue.queue.clear()
          except Queue.Empty:
             pass

##        currentBarHeight = appRoot.barYBase + currentHeight * appRoot.yScaleFactor

       currentLineLength = (currentTime - appRoot.previousTime) * appRoot.oscWidth /
appRoot.oscTimeToCross
       oscLineList = [appRoot.oscWidth, appRoot.previousCurrentHeight]
       oscLineList2 = [appRoot.oscWidth, appRoot.previousCurrentSideHeight]

       lineCumulativeLength = -currentLineLength
       lineIncPreviousTime = appRoot.previousTime

       for lineInc in range(len(currentPosAndTime)/2):#len(currentPosAndTime)/2
          lineIncTime = currentPosAndTime[0 + lineInc*2]
          lineIncTime = (lineIncTime - appRoot.startTime) * 100000
          lineIncHeight = currentPosAndTime[1 + lineInc*2][0]
```

```
            lineIncBarHeight = appRoot.barYBase + lineIncHeight * appRoot.yScaleFactor
            lineCumulativeLength += (lineIncTime - lineIncPreviousTime) * appRoot.oscWidth /
appRoot.oscTimeToCross
            oscLineList.append(appRoot.oscWidth + lineCumulativeLength)
            oscLineList.append(lineIncBarHeight)
            appRoot.previousCurrentHeight = lineIncBarHeight
            if(useSidePlot == True):
               lineIncSideHeight = currentPosAndTime[1 + lineInc*2][1]
               lineIncBarSideHeight = appRoot.barYBase + lineIncSideHeight * appRoot.yScaleFactor
               oscLineList2.append(appRoot.oscWidth + lineCumulativeLength)
               oscLineList2.append(lineIncBarSideHeight)
               appRoot.previousCurrentSideHeight = lineIncBarSideHeight
            lineIncPreviousTime = lineIncTime
##          print(oscLineList)
         global segIndicator
##          if(segIndicator == "green"):
##             segIndicator = "yellow"
##          else:
##             segIndicator = "green"
         segIndicator = "green"
         currentLine = appRoot.oscCanvas.create_line(oscLineList, fill=segIndicator, tags="oscLine")
         if(useSidePlot == True):
            currentLine2 = appRoot.oscCanvas.create_line(oscLineList2, fill="purple", tags="oscLine2")

##          print(len(appRoot.oscCanvas.find_withtag("oscLine")) - 1)
         linesPopped = 0
         for lineInc in range(len(appRoot.oscCanvas.find_withtag("oscLine")) - 1):
            lineInc = lineInc - linesPopped
            thisLine = appRoot.oscCanvas.find_withtag("oscLine")[lineInc]
            appRoot.oscCanvas.move(thisLine, -currentLineLength, 0)
            if(useSidePlot == True):
               thisLine2 = appRoot.oscCanvas.find_withtag("oscLine2")[lineInc]
               appRoot.oscCanvas.move(thisLine2, -currentLineLength, 0)
            appRoot.oscCanvas.update()
            if currentTime - appRoot.currentLineTimes[lineInc -1] > appRoot.oscTimeToCross:# * .9
               appRoot.oscCanvas.delete(thisLine)
               if(useSidePlot == True):
                  appRoot.oscCanvas.delete(thisLine2)
##                   appRoot.currentLines.pop(0)
               appRoot.currentLineTimes.pop(0)
               linesPopped += 1

         for lineInc in range(len(appRoot.oscCanvas.find_withtag("oscLineHorizontal"))):
            thisLine = appRoot.oscCanvas.find_withtag("oscLineHorizontal")[lineInc]
            appRoot.oscCanvas.move(thisLine, -currentLineLength, 0)
            appRoot.oscCanvas.update()

         linesPopped = 0
         if(len(appRoot.oscCanvas.find_withtag("oscLineHorizontal")) > 1):
            for lineInc in range(len(appRoot.oscCanvas.find_withtag("oscLineHorizontal"))):
               lineInc = lineInc - linesPopped
```

```
                thisLine = appRoot.oscCanvas.find_withtag("oscLineHorizontal")[lineInc]
                if currentTime - appRoot.currentPeakLineTimes[lineInc] > appRoot.oscTimeToCross:# * .9
                    appRoot.oscCanvas.delete(thisLine)
                    appRoot.currentPeakLineTimes.pop(0)
                    linesPopped += 1

            appRoot.previousTime = currentTime
##              appRoot.currentLines.append(currentLine)
            appRoot.currentLineTimes.append(currentTime)
##            appRoot.previousCurrentHeight = currentBarHeight
            ##################
            appRoot.currentQueue.queue.clear()

        appRoot.after(10, appRoot.process_serial)       # milliseconds

def diagnosticPrint(optLvl, inpStr): # A version of prints where each use has a tag, to easily turn on or off sets of outputs
    # Currently, continuous message outputs are level 1 and peak messages are level 2
    if(optLvl >= 3):
        print(inpStr)

class saveOptionDialog():
    def __init__(self, appRoot, writeFileName):
        top = self.top = tk.Toplevel(appRoot)
        top.title("Save?")
        infoLabelString = "Writing "+ str(appRoot.dataStoreT.qsize()) +" entries to "+ writeFileName
        top.infoLabel = tk.Label(top, text=infoLabelString).grid(row=0, columnspan=2, sticky='W',
pady=(5,0))
        if os.path.isfile(writeFileName):
            top.fileStatusLabel = tk.Label(top, text=writeFileName + " already exists. Overwrite or
Append?")
            top.fileStatusLabel.grid(row=1, columnspan=2, sticky='W')
            top.overwriteButton = tk.Button(top, text="Overwrite", command=lambda:
self.overWriteOption(appRoot, writeFileName))
            top.overwriteButton.grid(row=2, column=0, padx=(0,5))
            top.appendButton = tk.Button(top, text="Append", command=lambda:
self.appendOption(appRoot, writeFileName))
            top.appendButton.grid(row=3, column=0, padx=(0,5))
####            top.writeNewButton = tk.Button(top, text="Write New", command=lambda:
self.writeNewOption(appRoot, writeFileName))
####            top.writeNewButton.grid(row=4, column=0,padx=(0,5))
####            top.writeNewTextEntry = tk.Entry(top, width=25).grid(row=4, column=1,  sticky='W')
        else:
            top.fileStatusLabel = tk.Label(top, text=writeFileName + " does not exist. Make new file?")
            top.fileStatusLabel.grid(row=1, sticky='W')
            top.writeNewButton = tk.Button(top, text="Write File", command=lambda:
self.writeNewOption(appRoot, writeFileName))
            top.writeNewButton.grid(row=2, column=0, padx=(0,5))

        cancelButton = tk.Button(top, text="Cancel", command=self.cancelSaveOption)
        cancelButton.grid(row=5, column=0, padx=(0,5))
```

```
    def overWriteOption(self, appRoot, writeFileName):
        newOrAppend = 'w'
        storedDataToFile(appRoot, newOrAppend, writeFileName)
        print('Overwrote '+ writeFileName)
        self.top.destroy()
    def appendOption(self, appRoot, writeFileName):
        newOrAppend = 'a'
        storedDataToFile(appRoot, newOrAppend, writeFileName)
        print('Appended to '+ writeFileName)
        self.top.destroy()
    def writeNewOption(self, appRoot, writeFileName):
        newOrAppend = 'w'
        storedDataToFile(appRoot, newOrAppend, writeFileName)
        print('Wrote to '+ writeFileName)
        self.top.destroy()
    def cancelSaveOption(self):
        self.top.destroy()

def storedDataToFile(appRoot, newOrAppend, writeFileName):
    ##print(appRoot.dataStoreT.get() - appRoot.startTime) # .get() takes element and removes it
    with open(writeFileName, newOrAppend) as openFile:
        openFile.write("t, x, y, z\n")
        for timeInc in range(len(appRoot.dataStoreT.queue)):
            incList = appRoot.dataStoreT.queue[timeInc]
            incList[0] = incList[0] - appRoot.startTime
            strToWrite = str(incList)[1:-1]
            openFile.write(strToWrite + "\n")
##      appRoot.dataStoreT.queue.clear()

app = App()
##
def on_closing():
    if tkMessageBox.askokcancel("Quit", "Do you want to quit?"):
        global programIsClosing
        programIsClosing = True # Note: This is used to end the measurement loop and close the connection
        app.destroy()
app.protocol("WM_DELETE_WINDOW", on_closing)
##
app.mainloop()

##closeInput = raw_input("Press ENTER to exit") # Makes program not auto close window
```

Appendix B

Post-Processing Code

```
version = "synchViconIMU v5.1"

subjectId = "C1"
##subjectId = "C2"
##subjectId = "C3"
##subjectId = "C4"
##subjectId = "C5"
##subjectId = "C6"
##subjectId = "C7"
##subjectId = "C8"
##subjectId = "C9"
#
trialNumber = "1b"
##trialNumber = "1u"
##trialNumber = "2bd"
##trialNumber = "2bi"
##trialNumber = "2ui"

if(subjectId == "C4"):
   print("Processing special case: C4 had accidentally demeaned IMU data, so the IMU data for it will not be demeaned again")
   specialSubjectIfAny = "C4"
elif(subjectId == "C5" and trialNumber == "2bd"):
   print("Processing special case: C5_2bd had a different time ratio between IMU and vicon")
   specialSubjectIfAny = "C5_2bd"
elif(subjectId == "C9" and trialNumber == "2ui"):
   print("Processing special case: C9_2ui has numerous very low peaks, so demean is turned off to keep them from going below 0")
   specialSubjectIfAny = "C9_2ui"
else:
   specialSubjectIfAny = "none"

from xlrd import open_workbook
import Queue
import operator
import Tkinter as tk
import csv
import os
from bisect import bisect_left
import numpy

##saveOutputFile = True
saveOutputFile = False
emptyLRPeakValsAsZeros = True
componentToOutputOn = "max"
##componentToOutputOn = "min" # Outputting on min obsolete and probably would not work
```

```
doDemean = True
if(specialSubjectIfAny == "C4"):
   doDemean = False
elif(specialSubjectIfAny == "C9_2ui"):
   doDemean = False
##doNotDemeanOutput = True # If demeaning, Only uses demeaning to detect peaks
doNotDemeanOutput = False
disregardNegativePeaks = True
plotXAcc = False
XAccScale = 3 # Scale for making x acc visible plotted alongside y acc

global plotType # Obsolete at this point; consider getting rid of
##plotType = "Peak"
##plotType = "AllAcc"
plotType = "PeakAndAllAcc"

displayStartTime = 0
displayTimeSpan = 10
IMUTimeScaleDefault = 86251.815 * 1.00160 # Approximated IMU time scale; to be refined on
calibration
if(specialSubjectIfAny == "C5_2bd"):
   IMUTimeScaleDefault = 86251.815 * 1.00160 * 0.999 # Subject 5 had a notably different time scale
than the others
timeClickWindow = 0.24 # Window around click to check in for max
synchStopTime = 20 # When getting synch input, shows time from 0 to this
outputFolder = "SynchOutput"

displayStopTime = displayStartTime + displayTimeSpan

trialId = "I_"+ subjectId +"_"+ trialNumber
thisDir = os.path.dirname(__file__)
writeFileName = "I_"+ subjectId +"_"+ trialNumber +"_synchOutput.csv"
writeFilePath = os.path.join(thisDir, outputFolder, writeFileName)
peakCalibValue = 0.01

storedIMUDelay = ""
storedViconDelay = ""
if(subjectId == "C1"): # Values determined through testing and saved for quick reference
   if(trialNumber == "1b"):
      storedIMUDelay = 24.8370807414
      storedViconDelay = 22.058332
   elif(trialNumber == "1u"):
      storedIMUDelay = 16.4350689229
      storedViconDelay = 15.233334
   elif(trialNumber == "2bd"):
      storedIMUDelay = 13.5843944233
      storedViconDelay = 11.316667
   elif(trialNumber == "2bi"):
      storedIMUDelay = 19.377715588
      storedViconDelay = 17.308332
   elif(trialNumber == "2ui"):
```

```
        storedIMUDelay = 19.7736736112
        storedViconDelay = 14.666667
elif(subjectId == "C2"):
    if(trialNumber == "1b"):
        storedIMUDelay = 13.7733773603
        storedViconDelay = 10.133333
    elif(trialNumber == "1u"):
        storedIMUDelay = 17.2079770533
        storedViconDelay = 12.725
    elif(trialNumber == "2bd"):
        storedIMUDelay = 15.2182102955
        storedViconDelay = 9.875
    elif(trialNumber == "2bi"):
        storedIMUDelay = 13.5963925417
        storedViconDelay = 11.275
    elif(trialNumber == "2ui"):
        storedIMUDelay = 14.3873028784
        storedViconDelay = 12.641666
elif(subjectId == "C3"):
    if(trialNumber == "1b"):
        storedIMUDelay = 15.707146194
        storedViconDelay = 13.816667
    elif(trialNumber == "1u"):
        storedIMUDelay = 21.1485091411
        storedViconDelay = 18.775
    elif(trialNumber == "2bd"):
        storedIMUDelay = 13.2514340648
        storedViconDelay = 10.841666
    elif(trialNumber == "2bi"):
        storedIMUDelay = 14.9672354036
        storedViconDelay = 12.883333
    elif(trialNumber == "2ui"):
        storedIMUDelay = 10.8657193409
        storedViconDelay = 9.016666
elif(subjectId == "C4"):
    if(trialNumber == "1b"):
        storedIMUDelay = 18.7047951304
        storedViconDelay = 17.683332
    elif(trialNumber == "1u"):
        storedIMUDelay = 22.3773558299
        storedViconDelay = 20.741667
    elif(trialNumber == "2bd"):
        storedIMUDelay = 18.7707898105
        storedViconDelay = 17.358334
    elif(trialNumber == "2bi"):
        storedIMUDelay = 18.6568026566
        storedViconDelay = 17.291666
    elif(trialNumber == "2ui"):
        storedIMUDelay = 18.6238053166
        storedViconDelay = 17.016666
elif(subjectId == "C5"):
```

```
    if(trialNumber == "1b"):
        storedIMUDelay = 17.9478878437
        storedViconDelay = 16.366667
    elif(trialNumber == "1u"):
        storedIMUDelay = 17.019989769
        storedViconDelay = 15.566667
    elif(trialNumber == "2bd"):
        storedIMUDelay = 18.9817647153
        storedViconDelay = 17.133333
    elif(trialNumber == "2bi"):
        storedIMUDelay = 21.5504511379
        storedViconDelay = 20.5
    elif(trialNumber == "2ui"):
        storedIMUDelay = 18.0488812263
        storedViconDelay = 16.633333
elif(subjectId == "C6"):
    if(trialNumber == "1b"):
        storedIMUDelay = 20.1166206685
        storedViconDelay = 18.483334
    elif(trialNumber == "1u"):
        storedIMUDelay = 18.2278573504
        storedViconDelay = 15.0
    elif(trialNumber == "2bd"):
        storedIMUDelay = 19.3187307055
        storedViconDelay = 17.208334
    elif(trialNumber == "2bi"):
        storedIMUDelay = 20.8495417754
        storedViconDelay = 19.333334
    elif(trialNumber == "2ui"):
        storedIMUDelay = 19.6816813316
        storedViconDelay = 18.491667
elif(subjectId == "C7"):
    if(trialNumber == "1b"):
        storedIMUDelay = 18.0438727913
        storedViconDelay = 16.799999
    elif(trialNumber == "1u"):
        storedIMUDelay = 18.7187946115
        storedViconDelay = 16.958334
    elif(trialNumber == "2bd"):
        storedIMUDelay = 20.7965509236
        storedViconDelay = 18.366667
    elif(trialNumber == "2bi"):
        storedIMUDelay = 14.2463224722
        storedViconDelay = 13.0
    elif(trialNumber == "2ui"):
        storedIMUDelay = 17.6179245005
        storedViconDelay = 15.916667
elif(subjectId == "C8"):
    if(trialNumber == "1b"):
        storedIMUDelay = 18.8797785864
        storedViconDelay = 17.616667
```

```
    elif(trialNumber == "1u"):
        storedIMUDelay = 16.6540421276
        storedViconDelay = 15.691667
    elif(trialNumber == "2bd"):
        storedIMUDelay = 19.6176880142
        storedViconDelay = 18.433332
    elif(trialNumber == "2bi"):
        storedIMUDelay = 15.7341394462
        storedViconDelay = 14.6
    elif(trialNumber == "2ui"):
        storedIMUDelay = 20.3815950414
        storedViconDelay = 19.125
elif(subjectId == "C9"):
    if(trialNumber == "1b"):
        storedIMUDelay = 13.940355366
        storedViconDelay = 12.566667
    elif(trialNumber == "1u"):
        storedIMUDelay = 16.0481019457
        storedViconDelay = 13.808333
    elif(trialNumber == "2bd"):
        storedIMUDelay = 14.1493318147
        storedViconDelay = 12.483334
    elif(trialNumber == "2bi"):
        storedIMUDelay = 14.6702694005
        storedViconDelay = 13.008333
    elif(trialNumber == "2ui"):
        storedIMUDelay = 12.4875270375
        storedViconDelay = 10.858334
if(storedIMUDelay != ""):
    print("Using stored IMU calibration time "+ str(storedIMUDelay))
    print("Using stored Vicon calibration time "+ str(storedViconDelay))

class IMUOrViconObj:
    def __init__(self, IMUOrViconInput):
        if(IMUOrViconInput == "IMU"):
            self.storedTimeDelay = storedIMUDelay
            self.fileToOpen = "I_"+ subjectId +"_"+ trialNumber + ".csv"
            if(specialSubjectIfAny == "C4"):
                self.fileToOpen = "I_"+ subjectId +"_"+ trialNumber + "_yDemeaned.csv"
            subFolderIfAny = "IMUData"
##          self.writeFileName = "I_"+ subjectId +"_"+ trialNumber +"_IMUPeaks.csv"
            self.timeColInd = 0 # Leave this and the next line as default values
            self.xColInd = 1 # y col ind will be this +1
            self.readerDelimiter = ","
            self.dataTimeScale = IMUTimeScaleDefault
            self.accScale = 1
        else:
            self.storedTimeDelay = storedViconDelay
            self.fileToOpen = "I_"+ subjectId +"_"+ trialNumber[0] + "_output.tsv"
            subFolderIfAny = "ViconData"
##          self.writeFileName = "I_"+ subjectId +"_"+ trialNumber +"_ViconPeaks.csv"
```

```
            self.timeColInd = 1 # Time cols are identical and timeColInd = 1
            if(trialNumber == "1b" or trialNumber == "2bd"):
                self.xColInd = 2 # y col ind will be this +1
            elif(trialNumber == "1u" or trialNumber == "2bi"):
                self.xColInd = 18 # y col ind will be this +1
            elif(trialNumber == "2ui"):
                self.xColInd = 34 # y col ind will be this +1
            self.fY1ColInd = self.xColInd + 7
            self.fY2ColInd = self.fY1ColInd + 3
            self.LHSColInd = self.xColInd + 12
            self.RHSColInd = self.xColInd + 13
            self.HSErrorColInd = self.xColInd + 14
            self.force1ColInd = self.xColInd + 7
            self.force2ColInd = self.xColInd + 10
            self.readerDelimiter = "\t"
            self.dataTimeScale = 1
            self.accScale = 1/9.81
        self.fileToOpenPath = os.path.join(thisDir, subFolderIfAny, self.fileToOpen)
        self.allAccQueue = Queue.Queue()
        self.peakQueue = Queue.Queue()
        self.heelStrikeQueue = Queue.Queue()
        self.HSErrors = []
        self.yForceQueue = Queue.Queue()
        self.yForcePeaksR = []
        self.yForcePeaksL = []
        self.yForcePeaksAll = []
        if(self.storedTimeDelay == ""):
            self.calibClickCount = 0
            self.timeDelay = 0
        else:
            self.calibClickCount = 1
            self.timeDelay = float(self.storedTimeDelay)
        self.notDoneBothPlot = True
        self.hsPeaks = []
        self.LHSPeakAvg = 0
        self.RHSPeakAvg = 0
        self.LRMismatchCount = 0
        self.LRMismatchPercent = 0

def is_number(inputVal):
    if(inputVal == ""): # This is added since in vicon files there can be a lot of "" rows
        return False
    try:
        float(inputVal)
        return True
    except ValueError:
        return False

def doReadFile(IMUOrViconInput):
    if(IMUOrViconInput == "IMU"):
        global IMUGlobal
```

```
        dataObjInput = IMUGlobal
    else:
        global ViconGlobal
        dataObjInput = ViconGlobal
    with open(dataObjInput.fileToOpenPath, 'rb') as readFile:
        yAll = [] # All the forward points
        ySinceZero = [] # The forward points since the last zero cross
        ySinceZeroForOutput = []
        xSumSinceZero = 0 # Sum of x values since last 0
        xListSinceZero = [] # Part of old system used in trials
        ##prevXAvgForYMax = 0
        oldXCompareAvgForThisStep = 0 # This system is old but leave it here since it was used in trials 1-
9
        oldXCompareAvgForPreviousStep = 0
        YMaxPeakForThisStep = 0
        thisXCompareSum = 0
        thisXCompareCount = 0
        prevXComapreSum = 0
        prevXCompareCount = 0
        thisXCompareSide = ""
        prevComponentWas = "" # Set to max or min
        prevLorR = ""
        prevLorRIsError = ""
        prevYPeak = 0
        LorRErrorList = []
        maxPeakList = []
        maxMinPeakList = []
        barAvgLength = 3
        pastLeftPeaks = [0] * barAvgLength
        pastRightPeaks = [0] * barAvgLength
        prevYForMean = []
        prevYForMeanLenMax = 2000
        minPeakPointCount = 8
        global plotType
        print("Running "+ version +" to read "+ dataObjInput.fileToOpen +" ("+ IMUOrViconInput +" data
trial "+ trialNumber +")")
        demeanOutputString = ""
        if(doDemean):
            if(doNotDemeanOutput):
                demeanOutputString = " but not output values"
            else:
                demeanOutputString = " and output is demeaned"
        print("Demeaning "+ str(doDemean) + demeanOutputString +";    Reading on "+
componentToOutputOn)
        reader = csv.reader(readFile, delimiter = dataObjInput.readerDelimiter)
        yColInd = dataObjInput.xColInd + 1
        rowInc = 0
        HSPrevTime = 0
        for thisRow in reader:
            if(rowInc == 0): # vicon files have a source indicator in header top
                if(IMUOrViconInput == "Vicon"):
```

```
            # like J:\Lewek Lab\Data\Mike\IMU validation\C1\Session 1\I_C1_1u.c3d
            trialFilePath = thisRow[dataObjInput.xColInd]
            trialNameCheck = trialFilePath.index(trialNumber) # Intended to crash if not found
   ##          if(trialNumber
            print("Vicon data source file is: "+ trialFilePath.split("\\")[-1])
         else:
           if(is_number(thisRow[dataObjInput.xColInd]) and is_number(thisRow[yColInd])): # Note this
assumes non-numeric headers
      ##       print(thisRow[dataObjInput.xColInd])
            timeNow = float(thisRow[dataObjInput.timeColInd])
            xAcc = float(thisRow[dataObjInput.xColInd])
            yAcc = -float(thisRow[yColInd])
      ##       zAcc = float(thisRow[dataObjInput.xColInd + 2])

            yAccForOutput = yAcc
            if(doDemean):
               prevYForMean.append(yAcc)
               if(len(prevYForMean) > prevYForMeanLenMax):
                  prevYForMean.pop(0)
               yAcc = yAcc - sum(prevYForMean)/len(prevYForMean)
            if(doNotDemeanOutput == False):
               yAccForOutput = yAcc # Sets to the demeaned yAcc

            yAll.append(yAcc)
            ySinceZero.append(yAcc)
            ySinceZeroForOutput.append(yAccForOutput)
      ##       xSumSinceZero = xSumSinceZero + xAcc
##             xListSinceZero.append(xAcc)

            if(yAcc < 0):
               thisXCompareSum = thisXCompareSum + xAcc
               thisXCompareCount = thisXCompareCount + 1
            else:
               if(thisXCompareCount > 0):
                  if(prevXCompareCount > 0):
                     if((thisXCompareSum / float(thisXCompareCount)) > (prevXComapreSum /
float(prevXCompareCount))):
                        thisXCompareSide = "R"
                     else:
                        thisXCompareSide = "L"
                  prevXComapreSum = thisXCompareSum
                  prevXCompareCount = thisXCompareCount
                  thisXCompareSum = 0
                  thisXCompareCount = 0

            if(plotType == "Peak" or plotType == "PeakAndAllAcc"):
               y_if_Peak = 0 # set in peak detection then made part of dataObjInput.peakQueue
               L_or_R_peak = ""
               if (yAll[len(yAll) - 2] < 0 and yAcc > 0) or (yAll[len(yAll) - 2] > 0 and yAcc < 0):
                  if (len(ySinceZero)>minPeakPointCount): # If y since crossing is long enough
                     if (yAll[len(yAll) - 2] < 0 and yAcc > 0):
```

```
                    #  crosses from neg to pos (looking for a min)
                    peakYInd, peakY = min(enumerate(ySinceZero), key=operator.itemgetter(1))
                    dummyVar, peakYForOutput = min(enumerate(ySinceZeroForOutput),
key=operator.itemgetter(1))
                else:# crosses from pos to negative (looking for a max)
                    peakYInd, peakY = max(enumerate(ySinceZero), key=operator.itemgetter(1))
                    dummyVar, peakYForOutput = max(enumerate(ySinceZeroForOutput),
key=operator.itemgetter(1))
        ##              print("^ "+ str(-peakY) +" "+ str(-peakYForOutput))
                if(abs(peakY) > peakCalibValue * 0.5):
        ##              leftOrRightVal = 0 # Set on min, 1 if left and 2 if right
                    y_if_Peak = -peakYForOutput
                    peakY = -peakY
                    if(peakYInd < 2):
                        peakYInd = 2
##                      xListSinceZeroMiddle = xListSinceZero[peakYInd/2: peakYInd]
##                      xAvg = sum(xListSinceZeroMiddle)/len(xListSinceZeroMiddle)

                    lastAvg = 0
                    LorRisError = ''
                    yMaxPeak = ''
                    yMaxMinPeak = ''
                    LMaxPeak = ''
                    LMaxMinPeak = ''
                    RMaxPeak = ''
                    RMaxMinPeak = ''
                    if(emptyLRPeakValsAsZeros):
                        LMaxPeak = 0
                        LMaxMinPeak = 0
                        RMaxPeak = 0
                        RMaxMinPeak = 0

                    if(peakY > 0): # is a min (checking inside the height filter)
                        thisComponentIs = "min"
                        yMaxPeak = YMaxPeakForThisStep
        ##                  if(prevYPeak > 0):
        ##                      YMaxPeakForThisStep = 0
                        yMaxMinPeak = -(y_if_Peak - YMaxPeakForThisStep) # Does -(min -
prevMax) if on min
                    else:       # is a max (checking inside the height filter)
                        thisComponentIs = "max"
##                          oldXCompareAvgForThisStep = xAvg
                        YMaxPeakForThisStep = y_if_Peak
                        yMaxPeak = y_if_Peak
        ##                  if(prevYPeak < 0):
        ##                      prevYPeak = 0
                        yMaxMinPeak = y_if_Peak - prevYPeak # aaa
                    if(thisComponentIs == componentToOutputOn):
                        if(thisXCompareSide == "L"):
                            LMaxPeak = yMaxPeak
                            LMaxMinPeak = yMaxMinPeak
```

```
                        else:
                            RMaxPeak = yMaxPeak
                            RMaxMinPeak = yMaxMinPeak
                        L_or_R_peak = thisXCompareSide
                        if(L_or_R_peak != ''):
                            if(L_or_R_peak == prevLorR and prevLorRIsError != 1): # Outdated and
inadequate
                                LorRisError = 1
                            else:
                                LorRisError = 0
                            LorRErrorList.append(LorRisError)
                            prevLorRIsError = LorRisError
                            prevLorR = L_or_R_peak
        ##                  print([timeNow, y_if_Peak, xAvg, L_or_R_peak, LorRisError])
##                          dataObjInput.peakQueue.put([timeNow, xAvg, oldXCompareAvgForThisStep,
y_if_Peak, yMaxPeak, yMaxMinPeak, L_or_R_peak, LorRisError, LMaxPeak, RMaxPeak,
LMaxMinPeak, RMaxMinPeak]) # Y Inverted due to axis direction issues
                        dataObjInput.peakQueue.put([timeNow, "", "", y_if_Peak, yMaxPeak,
yMaxMinPeak, L_or_R_peak, LorRisError, LMaxPeak, RMaxPeak, LMaxMinPeak, RMaxMinPeak]) #
Y Inverted due to axis direction issues
                        prevYPeak = y_if_Peak
                    del ySinceZero[:]
                    del ySinceZeroForOutput[:]
        ##              xSumSinceZero = 0
##                      xListSinceZero = []
      ##          else: # else to if plotting peaks
      ##              Placeholder
                dataObjInput.allAccQueue.put([timeNow, xAcc, yAcc]) # Y Inverted due to axis direction
issues
            if(IMUOrViconInput == "Vicon" and (is_number(thisRow[dataObjInput.LHSColInd]) or
is_number(thisRow[dataObjInput.RHSColInd]))):
##                  if(thisRow[dataObjInput.timeColInd] != 0
                if(thisRow[dataObjInput.LHSColInd] != "" and thisRow[dataObjInput.RHSColInd] == ""): #
HS data
                    dataObjInput.heelStrikeQueue.put([thisRow[dataObjInput.LHSColInd], "L"])
                elif(thisRow[dataObjInput.RHSColInd] != "" and thisRow[dataObjInput.LHSColInd] == ""):
                    dataObjInput.heelStrikeQueue.put([thisRow[dataObjInput.RHSColInd], "R"])
                elif(thisRow[dataObjInput.RHSColInd] != "" and thisRow[dataObjInput.LHSColInd] != ""):
                    thisTimeR = float(thisRow[dataObjInput.RHSColInd])
                    thisTimeL = float(thisRow[dataObjInput.LHSColInd])
                    if(thisTimeR < thisTimeL):
                        dataObjInput.heelStrikeQueue.put([thisRow[dataObjInput.RHSColInd], "R"])
                        dataObjInput.heelStrikeQueue.put([thisRow[dataObjInput.LHSColInd], "L"])
                    else:
                        dataObjInput.heelStrikeQueue.put([thisRow[dataObjInput.LHSColInd], "L"])
                        dataObjInput.heelStrikeQueue.put([thisRow[dataObjInput.RHSColInd], "R"])
                if(thisRow[dataObjInput.HSErrorColInd] != ""):
                    dataObjInput.HSErrors.append(float(thisRow[dataObjInput.HSErrorColInd])) # HS error
data
            if(IMUOrViconInput == "Vicon" and is_number(thisRow[dataObjInput.force1ColInd])): #
Force data
```

```
                dataObjInput.yForceQueue.put([thisRow[dataObjInput.force1ColInd],
thisRow[dataObjInput.force2ColInd]])
            rowInc += 1
        if(IMUOrViconInput == "IMU"):
            IMUGlobal = dataObjInput
        else:
            ViconGlobal = dataObjInput

def doInterpolateForOutput(): # Done before doStepFindHS so force data can be on time scale for heel
strike finding
    # Converts all acc values to simple lists for interpolation function ## Note: Vicon timescale = 1 but
include for good practice
    ViconGlobal.allT = [x[0]*ViconGlobal.dataTimeScale - ViconGlobal.timeDelay for x in
ViconGlobal.allAccQueue.queue]
    ViconGlobal.allAcc = [x[2] for x in ViconGlobal.allAccQueue.queue]
    IMUGlobal.allT = [x[0]*IMUGlobal.dataTimeScale - IMUGlobal.timeDelay for x in
IMUGlobal.allAccQueue.queue]
    IMUGlobal.allAcc = [x[2] for x in IMUGlobal.allAccQueue.queue]
    IMUGlobal.sampledAcc = numpy.interp(ViconGlobal.allT, IMUGlobal.allT, IMUGlobal.allAcc)
    # Interpolates the y forces to fit the vicon time scale
    VScaledInc = numpy.linspace(0, 1, len(ViconGlobal.allT))
    FYScaledInc = numpy.linspace(0, 1, len(ViconGlobal.yForceQueue.queue))
    FY1AllVals = [x[0] for x in ViconGlobal.yForceQueue.queue]
    FY2AllVals = [x[1] for x in ViconGlobal.yForceQueue.queue]
    ViconGlobal.FY1SampledVals = numpy.interp(VScaledInc, FYScaledInc, FY1AllVals) # FY 1 is R"
    ViconGlobal.FY2SampledVals = numpy.interp(VScaledInc, FYScaledInc, FY2AllVals) # FY 2 is L"
    for thisHSError in ViconGlobal.HSErrors:
        if(thisHSError != ViconGlobal.HSErrors[0]):
            print("Crossover marker at "+ str(thisHSError*ViconGlobal.dataTimeScale -
ViconGlobal.timeDelay))
    doStepFindHS()

def doStepFindHS():
    global IMUGlobal
    global ViconGlobal
    heelStrikeTimes = []
    heelStrikeSides = []
    for hsInc in range(len(ViconGlobal.heelStrikeQueue.queue)): # Heel strike data is only from Vicon
        thisHs = ViconGlobal.heelStrikeQueue.queue[hsInc]
        heelStrikeTimes.append(float(thisHs[0]) - ViconGlobal.timeDelay)
        heelStrikeSides.append(thisHs[1])
    dataSources = ["IMU", "Vicon"]
    for dataInc in range(len(dataSources)):
        if(dataSources[dataInc] == "IMU"):
            dataObjInput = IMUGlobal
        else:
            dataObjInput = ViconGlobal
        zeroCrossPeakList = dataObjInput.peakQueue.queue # List of peaks determined by zero crosssing
(the live method)
        zeroCrossPeakInc = 0
        LHSPeakMagSum = 0
```

```
        LHSPeakCount = 0
        RHSPeakMagSum = 0
        RHSPeakCount = 0
        stepHsMaxes = []
        thisMaxSide = ""
        thisMaxSideFromX = ""
        thisStepCrossedZero = "Start"
        LRMismatchCount = 0
##      LRMismatchThisHS = 0 # Records this to subtract if in the case of a subpeak that gets dealt with
        nextHsInd = 0
##      print(heelStrikeTimes)
        nextHSTime = heelStrikeTimes[nextHsInd]
        rejectedPeakCount = 0
        zeroCrossPeakTimesThisHS = []
##      zeroCrossPeakAccsThisHS = []
        currentMaxAcc = 0
        currentMaxAccTime = 0
        currentMaxForceR = 0
        currentMaxForceL = 0
        currentMaxForceTimeR = 0
        currentMaxForceTimeL = 0
        hasHadHSAfterZero = False
        hasHadHSAfterZeroR = False
        hasHadHSAfterZeroL = False
        crossoverInd = 1 # Skips the first crossover item since that one is just a test
        hasCrossoverR = False
        hasCrossoverL = False
        for timeInc in range(len(dataObjInput.allAccQueue.queue)):
            incList = dataObjInput.allAccQueue.queue[timeInc]
            thisTime = incList[0]*dataObjInput.dataTimeScale - dataObjInput.timeDelay
            thisZeroCrossPeak =  zeroCrossPeakList[zeroCrossPeakInc]
            if(hasHadHSAfterZeroR or hasHadHSAfterZeroL): # Occurs when Vicon and a HS past 0 has
occurred
                thisFYR = dataObjInput.FY1SampledVals[timeInc]
                if(hasHadHSAfterZeroR and thisFYR > currentMaxForceR):
                    currentMaxForceR = thisFYR
                    currentMaxForceTimeR = thisTime
                thisFYL = dataObjInput.FY2SampledVals[timeInc]
                if(hasHadHSAfterZeroL and thisFYL > currentMaxForceL):
                    currentMaxForceL = thisFYL
                    currentMaxForceTimeL = thisTime
                if(crossoverInd < len(dataObjInput.HSErrors)): # If within bounds of crossover list
                    currentCrossoverTime = dataObjInput.HSErrors[crossoverInd]
                    if(currentCrossoverTime <= incList[0]):
                        hasCrossoverR = True
                        hasCrossoverL = True
                        crossoverInd = crossoverInd + 1
            if(nextHsInd > 0 and heelStrikeTimes[nextHsInd -1] > 0):
                if(hasHadHSAfterZero == False):
                    hasHadHSAfterZero = True
                if(dataSources[dataInc] == "Vicon"): # Only Vicon uses force data which these are used for
```

```
            if(hasHadHSAfterZeroL == False and heelStrikeSides[nextHsInd -1] == "L"):
                print(">> First V L")
                hasHadHSAfterZeroL = True
            if(hasHadHSAfterZeroR == False and heelStrikeSides[nextHsInd -1] == "R"):
                print(">> First V R")
                hasHadHSAfterZeroR = True
        if(zeroCrossPeakInc < len(zeroCrossPeakList) -1 and thisZeroCrossPeak[0] <= incList[0]): #
Times from same source will be comparable before scaling
            LHeight = thisZeroCrossPeak[8]
            RHeight = thisZeroCrossPeak[9]
            if(LHeight != 0 or RHeight != 0):
##                print([thisTime, nextHSTime])
                if(thisMaxSideFromX == ""):
                    thisMaxSideFromX = thisZeroCrossPeak[6]
                else:
                    thisMaxSideFromX = "E" # for error
##                  if(thisMaxSide != "" and thisMaxSideFromX != thisMaxSide):
##                      print("hs "+ thisMaxSide +" / x "+ thisMaxSideFromX)
##                      LRMismatchCount = LRMismatchCount + 1
##                      LRMismatchThisHS = LRMismatchThisHS + 1
                zeroCrossPeakTimesThisHS.append(
round(thisZeroCrossPeak[0]*dataObjInput.dataTimeScale - dataObjInput.timeDelay, 3) )
##                  zeroCrossPeakAccsThisHS.append(incList[2]) ## Approximated as the first acceleration
            zeroCrossPeakInc = zeroCrossPeakInc + 1
        if(thisTime < nextHSTime):
            if(thisStepCrossedZero != "UpAndDown"):
                if(incList[2] > currentMaxAcc):
                    currentMaxAcc = incList[2]
                    currentMaxAccTime = thisTime
            if(thisStepCrossedZero == "Start" and incList[2] > 0):
                thisStepCrossedZero = "Up"
            if(thisStepCrossedZero == "Up" and incList[2] < 0):
##                  print("UpAndDown")
##                  print(currentMaxAcc)
                thisStepCrossedZero = "UpAndDown" # Stops recording for the step on first negative after
positive has occured
        else:
            if(nextHsInd != 0 and currentMaxAcc != 0):
                if(hasHadHSAfterZero): # Waits until at least one HS after calibrated zero
                    maxTime = currentMaxAccTime
                    maxAcc = currentMaxAcc * dataObjInput.accScale
                    if(len(zeroCrossPeakTimesThisHS) < 1 or len(zeroCrossPeakTimesThisHS) > 2):
                        # 2 peaks are allowed since sometimes there is one false peak, in which case the highest
peak is kept
                        if(len(zeroCrossPeakTimesThisHS) < 1 or max(zeroCrossPeakTimesThisHS) > 0):
                            # Rejects the peaks before the time delay but does not give error
                            print("!!! Row rejected due to zero cross peaks of
"+str(zeroCrossPeakTimesThisHS)+" at "+str(maxTime))
                            rejectedPeakCount = rejectedPeakCount + 1
##                          LRMismatchCount = LRMismatchCount - LRMismatchThisHS # Does not count
rejected peak sfor mismatches
```

```
                    stepHsMaxes.append(["", "", thisMaxSide, ""])
                else:
##                      if(len(zeroCrossPeakTimesThisHS) == 2):
##                          LRMismatchCount = LRMismatchCount - LRMismatchThisHS # Does not count
false peaks for mismatches
                    if(thisMaxSide != thisMaxSideFromX):
                        LRMismatchCount = LRMismatchCount + 1
                    if(thisMaxSideFromX == "L"):
                        LHSPeakMagSum = LHSPeakMagSum + maxAcc
                        LHSPeakCount = LHSPeakCount + 1
                    elif(thisMaxSideFromX == "R"):
                        RHSPeakMagSum = RHSPeakMagSum + maxAcc
                        RHSPeakCount = RHSPeakCount + 1
                    stepHsMaxes.append([maxTime, maxAcc, thisMaxSide, thisMaxSideFromX])
            if(dataSources[dataInc] == "Vicon" and (hasHadHSAfterZeroR or hasHadHSAfterZeroL)):
                if(thisMaxSide == "L"): # R force peak is R to R which corresponds with a L acc peak
                    if(hasHadHSAfterZeroR and hasCrossoverR == False):
                        dataObjInput.yForcePeaksR.append([currentMaxForceTimeR, currentMaxForceR])
                        dataObjInput.yForcePeaksAll.append([currentMaxForceTimeR, currentMaxForceR])
                    else:
                        dataObjInput.yForcePeaksR.append(["", ""])
                        dataObjInput.yForcePeaksAll.append(["", ""])
                        if(hasCrossoverR == True):
                            print("FYR peak rejected due to crossover marker at "+ str(currentMaxForceTimeR))
                    currentMaxForceTimeR = 0
                    currentMaxForceR = 0
                    hasCrossoverR = False
                if(thisMaxSide == "R"):
                    if(hasHadHSAfterZeroL and hasCrossoverL == False):
                        dataObjInput.yForcePeaksL.append([currentMaxForceTimeL, currentMaxForceL])
                        dataObjInput.yForcePeaksAll.append([currentMaxForceTimeL, currentMaxForceL])
                    else:
                        dataObjInput.yForcePeaksL.append(["", ""])
                        dataObjInput.yForcePeaksAll.append(["", ""])
                        if(hasCrossoverL == True):
                            print("FYL peak rejected due to crossover marker at "+ str(currentMaxForceTimeL))
                    currentMaxForceTimeL = 0
                    currentMaxForceL = 0
                    hasCrossoverL = False
            thisMaxSide = heelStrikeSides[nextHsInd]
            thisMaxSideFromX = ""
            thisStepCrossedZero = "Start"
##              LRMismatchThisHS = 0
            nextHsInd = nextHsInd + 1
            zeroCrossPeakTimesThisHS = []
##              zeroCrossPeakAccsThisHS = []
            if(nextHsInd > len(heelStrikeTimes) -1): # Ends at the second to last heel strike
              break
            nextHSTime = heelStrikeTimes[nextHsInd]
            currentMaxAcc = 0
            currentMaxAccTime = 0
```

```
        dataObjInput.hsPeaks = stepHsMaxes
        dataObjInput.LRMismatchCount = LRMismatchCount
        dataObjInput.LRMismatchPercent = (float(LRMismatchCount) /
float(len(IMUGlobal.hsPeaks)))*100
        dataObjInput.LHSPeakAvg = LHSPeakMagSum / float(LHSPeakCount)
        dataObjInput.RHSPeakAvg = RHSPeakMagSum / float(RHSPeakCount)
    if(len(ViconGlobal.hsPeaks) != len(IMUGlobal.hsPeaks)):
        print("!!!!!! Warning: HS peak counts don't match: "+ str(len(ViconGlobal.hsPeaks)) +" / "+
str(len(IMUGlobal.hsPeaks)))
    print(str(len(ViconGlobal.hsPeaks)) +" valid HS max peaks processed")
    print("  "+ str(IMUGlobal.LRMismatchCount) +" hs vs x L/R mismatches in IMU for "+
str(IMUGlobal.LRMismatchPercent)+"% mismatch")
    print("  "+ str(ViconGlobal.LRMismatchCount) +" hs vs x L/R mismatches in Vicon for "+
str(ViconGlobal.LRMismatchPercent)+"% mismatch")
    print("R IMU peak avg / Vicon peak avg "+
str(IMUGlobal.RHSPeakAvg/ViconGlobal.RHSPeakAvg))
    print("L IMU peak avg / Vicon peak avg "+ str(IMUGlobal.LHSPeakAvg/ViconGlobal.LHSPeakAvg))
    doSaveOutputFile()

def doSaveOutputFile():
    if(saveOutputFile):
        global IMUGlobal
        global ViconGlobal
        print("Saving synchronised data to "+ writeFilePath)
        VAllSkipInc = 0
        while(ViconGlobal.allT[VAllSkipInc] < 0):
            VAllSkipInc = VAllSkipInc + 1
        with open(writeFilePath, 'w') as openFile:
##          errorRate = str( round(float(sum(LorRErrorList))/len(LorRErrorList) * 100, 2) ) + "%"
##          maxAvg = str(float(sum(maxPeakList))/len(maxPeakList))
##          maxMinAvg = str(float(sum(maxMinPeakList))/len(maxMinPeakList))
##          stepCount = str(len(dataObjInput.peakQueue.queue)/2)
            openFile.write("Made using "+ version +"\n")
            openFile.write("Subject number "+ subjectId +" Trial part "+ trialNumber +"\n")
            openFile.write("Data from "+ IMUGlobal.fileToOpen +" and "+ ViconGlobal.fileToOpen+"\n")
            openFile.write("Step Count, "+ str(len(ViconGlobal.hsPeaks)) +"\n")

            openFile.write("IMU L/R Mismatch, IMU % Mismatch, Vicon L/R Mismatch, Vicon %
Mismatch\n")
            openFile.write(str(IMUGlobal.LRMismatchCount) +",")
            openFile.write(str(IMUGlobal.LRMismatchPercent) +"%,")
            openFile.write(str(ViconGlobal.LRMismatchCount)+",")
            openFile.write(str(ViconGlobal.LRMismatchPercent) +"%\n")

##          openFile.write("IMU L/R Mismatch, "+ str(IMUGlobal.LRMismatchCount) +"\n")
##          openFile.write("IMU % Mismatch, "+ str(IMUGlobal.LRMismatchPercent) +"%\n")
##          openFile.write("Vicon L/R Mismatch, "+ str(ViconGlobal.LRMismatchCount) +"\n")
##          openFile.write("Vicon % Mismatch, "+ str(ViconGlobal.LRMismatchPercent) +"%\n")

            openFile.write("IMU R avg hs peak, Vicon R avg hs peak,")
            openFile.write("IMU L avg hs peak, Vicon L avg hs peak,")
```

```
        openFile.write("R IMU/Vicon avg hs peak, L IMU/Vicon avg hs peak\n")
        openFile.write(str(IMUGlobal.RHSPeakAvg) +",")
        openFile.write(str(ViconGlobal.RHSPeakAvg) +",")
        openFile.write(str(IMUGlobal.LHSPeakAvg) +",")
        openFile.write(str(ViconGlobal.LHSPeakAvg) +",")
        openFile.write(str(IMUGlobal.RHSPeakAvg/ViconGlobal.RHSPeakAvg) +",")
        openFile.write(str(IMUGlobal.LHSPeakAvg/ViconGlobal.LHSPeakAvg) +"\n")

##          openFile.write("IMU R avg hs peak, "+ str(IMUGlobal.RHSPeakAvg) +"\n")
##          openFile.write("Vicon R avg hs peak, "+ str(ViconGlobal.RHSPeakAvg) +"\n")
##          openFile.write("IMU L avg hs peak, "+ str(IMUGlobal.LHSPeakAvg) +"\n")
##          openFile.write("Vicon L avg hs peak, "+ str(ViconGlobal.LHSPeakAvg) +"\n")
##          openFile.write("IMU/Vicon R avg hs peak, "+
str(IMUGlobal.RHSPeakAvg/ViconGlobal.RHSPeakAvg) +"\n")
##          openFile.write("IMU/Vicon L avg hs peak, "+
str(IMUGlobal.LHSPeakAvg/ViconGlobal.LHSPeakAvg) +"\n")
        openFile.write("IMU synch delay, "+ str(IMUGlobal.timeDelay) +"\n")
        openFile.write("Vicon synch delay, "+ str(ViconGlobal.timeDelay) +"\n")
        openFile.write("\n")
        headerString = ""
        headerString = headerString + "I Peak T" + ","
        headerString = headerString + "V Peak T" + ","
        headerString = headerString + "I Peak Mag" + ","
        headerString = headerString + "V Peak Mag" + ","
        headerString = headerString + "HS L/R" + ","
        headerString = headerString + "I L/R" + ","
        headerString = headerString + "V L/R" + ","
        headerString = headerString + "FYR Peak T" + ","
        headerString = headerString + "FYL Peak T" + ","
        headerString = headerString + "FYR Peak Mag" + ","
        headerString = headerString + "FYL Peak Mag" + ","
        headerString = headerString + "FY All Peak T" + ","
        headerString = headerString + "FY All Peak Mag" + ","
        headerString = headerString + "V All T" + ","
        headerString = headerString + "I Sampled Acc" + ","
        headerString = headerString + "V All Acc" + ","
        headerString = headerString + "FY 1/R" + "," # Force plate 1 is R and 2 is L
        headerString = headerString + "FY 2/L"
##          dataObjInput.yForcePeaksR
        openFile.write(headerString +"\n")
        oInc = 0
        while(oInc + VAllSkipInc < len(ViconGlobal.allT)):
            outRowString = ""
            if(oInc < len(ViconGlobal.hsPeaks)): # Heel strike peak data
                outRowString = outRowString + str(IMUGlobal.hsPeaks[oInc][0]) + "," # I Peak T
                outRowString = outRowString + str(ViconGlobal.hsPeaks[oInc][0]) + "," # V Peak T
                outRowString = outRowString + str(IMUGlobal.hsPeaks[oInc][1]) + "," # I Peak Mag
                outRowString = outRowString + str(ViconGlobal.hsPeaks[oInc][1]) + ","  # V Peak Mag
                outRowString = outRowString + str(ViconGlobal.hsPeaks[oInc][2]) + "," # HS L/R (both are
same so from V)
                outRowString = outRowString + str(IMUGlobal.hsPeaks[oInc][3]) + "," # I L/R
```

```
            outRowString = outRowString + str(ViconGlobal.hsPeaks[oInc][3]) + "," # V L/R
        else:
            outRowString = outRowString +  ",,,,,,,"
        if(oInc < len(ViconGlobal.yForcePeaksR)): # Y Force R Peak Time
            outRowString = outRowString + str(ViconGlobal.yForcePeaksR[oInc][0]) + "," # FY R T"
        else:
            outRowString = outRowString +  ","
        if(oInc < len(ViconGlobal.yForcePeaksL)): # Y Force L Peak Time
            outRowString = outRowString + str(ViconGlobal.yForcePeaksL[oInc][0]) + "," # FY L T"
        else:
            outRowString = outRowString +  ","
        if(oInc < len(ViconGlobal.yForcePeaksR)): # Y Force R Peak Mag
            outRowString = outRowString + str(ViconGlobal.yForcePeaksR[oInc][1]) + "," # FY R Mag
        else:
            outRowString = outRowString +  ","
        if(oInc < len(ViconGlobal.yForcePeaksL)): # Y Force L Peak Mag
            outRowString = outRowString + str(ViconGlobal.yForcePeaksL[oInc][1]) + ","  # FY L Mag
        else:
            outRowString = outRowString +  ","
        if(oInc < len(ViconGlobal.yForcePeaksAll)): # Y Force All Peaks
            outRowString = outRowString + str(ViconGlobal.yForcePeaksAll[oInc][0]) + ","  # FY All T
            outRowString = outRowString + str(ViconGlobal.yForcePeaksAll[oInc][1]) + ","  # FY All
Mag
        else:
            outRowString = outRowString +  ",,"
        outRowString = outRowString + str(ViconGlobal.allT[oInc + VAllSkipInc]) + "," # V All T
        outRowString = outRowString + str(IMUGlobal.sampledAcc[oInc + VAllSkipInc]*
IMUGlobal.accScale) + "," # I Sampled Acc
        outRowString = outRowString + str(ViconGlobal.allAcc[oInc + VAllSkipInc]*
ViconGlobal.accScale) + "," # V All Acc
        outRowString = outRowString + str(ViconGlobal.FY1SampledVals[oInc + VAllSkipInc]) + ","
# FY 1/R"
        outRowString = outRowString + str(ViconGlobal.FY2SampledVals[oInc + VAllSkipInc])
        openFile.write(outRowString +"\n")
        oInc = oInc + 1
    else:
        print("Output saving disabled; saveOutputFile currently False")

class doOutputDataPlots(tk.Tk):
    def __init__(appRoot):
        tk.Tk.__init__(appRoot)
        global IMUGlobal
        global ViconGlobal

        # Canvas and layout
        appRoot.title(version +" reading "+subjectId+"_"+trialNumber)
        appRoot.plotWidth = 1200
        appRoot.plotHeight = 170
        appRoot.geometry(str(appRoot.plotWidth) + "x700")
        gridRow = 0
        appRoot.titleText1 = tk.Label(appRoot,text="Title Placeholder",font=('Helvetica 10 bold'))
```

```
        appRoot.titleText1.grid(row=gridRow, column=0)
        gridRow += 1
        appRoot.plotFrame1 =
tk.Frame(appRoot,width=appRoot.plotWidth,height=appRoot.plotHeight)#bd='1',relief='solid'
        appRoot.plotFrame1.grid(row=gridRow, columnspan=1)
        appRoot.plotCanvas1 = tk.Canvas(appRoot.plotFrame1, width=appRoot.plotWidth,
height=appRoot.plotHeight, bg='white')
        appRoot.plotCanvas1.config(scrollregion=(0,0,appRoot.plotWidth,appRoot.plotHeight))
        appRoot.hbar1=tk.Scrollbar(appRoot.plotFrame1,orient='horizontal')
        appRoot.hbar1.pack(side='bottom',fill='x')
        appRoot.hbar1.config(command=appRoot.plotCanvas1.xview)
        appRoot.plotCanvas1.config(xscrollcommand=appRoot.hbar1.set)
        appRoot.plotCanvas1.config(width=appRoot.plotWidth,height=appRoot.plotHeight)
##      appRoot.vbar1=tk.Scrollbar(appRoot.plotFrame1,orient='vertical')
##      appRoot.vbar1.pack(side='right',fill='y')
##      appRoot.vbar1.config(command=appRoot.plotCanvas1.yview)
##      appRoot.plotCanvas1.config(width=500,height=300)
##      appRoot.plotCanvas1.config(xscrollcommand=appRoot.hbar1.set,
yscrollcommand=appRoot.vbar1.set)
        appRoot.plotCanvas1.pack(side='left',expand=True,fill='both')
        IMUGlobal.titleText = appRoot.titleText1
        IMUGlobal.plotCanvas = appRoot.plotCanvas1
        gridRow += 1
##      appRoot.stepCountText = tk.Label(appRoot, text="Placeholder")
##      appRoot.stepCountText.grid(row=gridRow, column=0)
##      gridRow += 1
        appRoot.titleText2 = tk.Label(appRoot,text="Title Placeholder",font=('Helvetica 10 bold'))
        appRoot.titleText2.grid(row=gridRow, column=0)
        gridRow += 1
        appRoot.plotFrame2 =
tk.Frame(appRoot,width=appRoot.plotWidth,height=appRoot.plotHeight)#bd='1',relief='solid'
        appRoot.plotFrame2.grid(row=gridRow, columnspan=1)
        appRoot.plotCanvas2 = tk.Canvas(appRoot.plotFrame2, width=appRoot.plotWidth,
height=appRoot.plotHeight, bg='white')
        appRoot.plotCanvas2.config(scrollregion=(0,0,appRoot.plotWidth,appRoot.plotHeight))
        appRoot.hbar2=tk.Scrollbar(appRoot.plotFrame2,orient='horizontal')
        appRoot.hbar2.pack(side='bottom',fill='x')
        appRoot.hbar2.config(command=appRoot.plotCanvas2.xview)
        appRoot.plotCanvas2.config(xscrollcommand=appRoot.hbar2.set)
        appRoot.plotCanvas2.config(width=appRoot.plotWidth,height=appRoot.plotHeight)
##      appRoot.vbar2=tk.Scrollbar(appRoot.plotFrame2,orient='vertical')
##      appRoot.vbar2.pack(side='right',fill='y')
##      appRoot.vbar2.config(command=appRoot.plotCanvas2.yview)
##      appRoot.plotCanvas2.config(width=500,height=300)
##      appRoot.plotCanvas2.config(xscrollcommand=appRoot.hbar2.set,
yscrollcommand=appRoot.vbar2.set)
        appRoot.plotCanvas2.pack(side='left',expand=True,fill='both')
        ViconGlobal.titleText = appRoot.titleText2
        ViconGlobal.plotCanvas = appRoot.plotCanvas2
        gridRow += 1
        appRoot.titleText3 = tk.Label(appRoot,text="Title Placeholder",font=('Helvetica 10 bold'))
```

```
        appRoot.titleText3.grid(row=gridRow, column=0)
        gridRow += 1
        appRoot.plotFrame3 =
tk.Frame(appRoot,width=appRoot.plotWidth,height=appRoot.plotHeight)#bd='1',relief='solid'
        appRoot.plotFrame3.grid(row=gridRow, columnspan=1)
        appRoot.plotCanvas3 = tk.Canvas(appRoot.plotFrame3, width=appRoot.plotWidth,
height=appRoot.plotHeight, bg='white')
        appRoot.plotCanvas3.config(scrollregion=(0,0,appRoot.plotWidth,appRoot.plotHeight))
        appRoot.hbar3=tk.Scrollbar(appRoot.plotFrame3,orient='horizontal')
        appRoot.hbar3.pack(side='bottom',fill='x')
        appRoot.hbar3.config(command=appRoot.plotCanvas3.xview)
        appRoot.plotCanvas3.config(xscrollcommand=appRoot.hbar3.set)
        appRoot.plotCanvas3.config(width=appRoot.plotWidth,height=appRoot.plotHeight)
##        appRoot.vbar3=tk.Scrollbar(appRoot.plotFrame3,orient='vertical')
##        appRoot.vbar3.pack(side='right',fill='y')
##        appRoot.vbar3.config(command=appRoot.plotCanvas3.yview)
##        appRoot.plotCanvas3.config(width=500,height=300)
##        appRoot.plotCanvas3.config(xscrollcommand=appRoot.hbar3.set,
yscrollcommand=appRoot.vbar3.set)
        appRoot.plotCanvas3.pack(side='left',expand=True,fill='both')
        gridRow += 1

        doPlotting(appRoot, "IMU", False)
        doPlotting(appRoot, "Vicon", False)

##def getClosestIncFromSortedList(myList, myNumber):
##    """
##    Assumes myList is sorted. Returns closest value to myNumber.
##    If two numbers are equally close, return the smallest number.
##    """
##    pos = bisect_left(myList, myNumber)
##    if pos == 0:
##        return myList[0]
##    if pos == len(myList):
##        return myList[-1]
##    before = myList[pos - 1]
##    after = myList[pos]
##    if after - myNumber < myNumber - before:
##       return pos
##    else:
##       return (pos - 1)

def onCalibClick(event, IMUOrViconInput, appRoot, scaleInfo, windowData):
    global IMUGlobal
    global ViconGlobal
    if(IMUOrViconInput == "IMU"):
        dataObjInput = IMUGlobal
    else:
        dataObjInput = ViconGlobal
    clickMarkColor = "green"
    xBase = scaleInfo[0]
```

```
    xScale = scaleInfo[1]
    yBase = scaleInfo[2]
    yScale = scaleInfo[3]
    clickXUnscrolled = event.widget.canvasx(event.x)
    clickTime = (clickXUnscrolled - xBase)/xScale + dataObjInput.timeDelay
##    print("Clicked "+ str(clickTime))
    clickWindowAcc = []
    clickWindowT = []
    for pointInd in range(len(windowData[0])):
        if(windowData[0][pointInd] > (clickTime - timeClickWindow) and windowData[0][pointInd] <
(clickTime + timeClickWindow)):
            clickWindowAcc.append(windowData[1][pointInd])
            clickWindowT.append(windowData[0][pointInd])
    if((yBase - event.y) >= 0): # If click is above mid line
        clickWindowMaxIndex = max(xrange(len(clickWindowAcc)), key=clickWindowAcc.__getitem__)
    else:
        clickWindowMaxIndex = min(xrange(len(clickWindowAcc)), key=clickWindowAcc.__getitem__)
    clickWindowAccMaxVal = clickWindowAcc[clickWindowMaxIndex]
    clickWindowAccMaxT = clickWindowT[clickWindowMaxIndex]

    if(dataObjInput.calibClickCount > 0):
        clickOutString = IMUOrViconInput + " Peak t = "+ str(clickWindowAccMaxT -
dataObjInput.timeDelay)
        clickOutString = clickOutString + " peak acc = "+ str(clickWindowAccMaxVal) +""
        clickOutString = clickOutString + " [raw t is "+ str(clickWindowAccMaxT) +"]"
        clickOutString = clickOutString + " (click # "+ str(dataObjInput.calibClickCount) +")"
        print(clickOutString)
        windowLX = (clickWindowT[0] - dataObjInput.timeDelay)*xScale + xBase
        windowRX = (clickWindowT[len(clickWindowT)-1] - dataObjInput.timeDelay)*xScale + xBase
        windowMaxPlotHeight = yBase + yScale*clickWindowAccMaxVal
        dataObjInput.plotCanvas.create_line(windowLX, windowMaxPlotHeight, windowRX,
windowMaxPlotHeight, fill=clickMarkColor, width=3)
        windowMaxPlotTime = xBase + xScale*(clickWindowAccMaxT - dataObjInput.timeDelay)
        dataObjInput.plotCanvas.create_line(windowMaxPlotTime, windowMaxPlotHeight + 10,
windowMaxPlotTime, windowMaxPlotHeight - 10, fill=clickMarkColor, width=3)

    dataObjInput.calibClickCount = dataObjInput.calibClickCount + 1
    if(dataObjInput.calibClickCount == 1): # If this is the first calibration click
        print(IMUOrViconInput +" Calibrated start at "+ str(clickWindowAccMaxT))
        dataObjInput.timeDelay = clickWindowAccMaxT
        doPlotting(appRoot, IMUOrViconInput, True) # Remakes the plot
        if(IMUGlobal.timeDelay > 0 and ViconGlobal.timeDelay > 0):
            print("Initial calibrations complete. Ploting comparison and making output file.")
            doBothPlotAndOutput(appRoot)

def doBothPlotAndOutput(appRoot):
    doInterpolateForOutput()
    doPlotting(appRoot, "IMU", True)
    doPlotting(appRoot, "Vicon", True)
    doPlotting(appRoot, "BothPart1", True)
```

```
def doPlotting(appRoot, IMUOrViconInput, clearPrevious):
   if(IMUOrViconInput == "IMU" or IMUOrViconInput == "BothPart1"): # Plots IMU first when
plotting both, then Vicon
      global IMUGlobal
      dataObjInput = IMUGlobal
      plotColor = "black"
      plotColorX = "green"
   else: # Mate to catch "Vicon" or "Both" ("Both" instead of "BothPart2" for title neatness)
      global ViconGlobal
      dataObjInput = ViconGlobal
      plotColor = "purple"
      plotColorX = "green"
   global plotType
   # Plotting Base
##    if(dataObjInput.calibClickCount > 0):
   displayStartTimeShifted = displayStartTime + dataObjInput.timeDelay
   if(dataObjInput.calibClickCount == 0):
      plotStartTime = 0
      displayStartTimeShifted = 0
      displayStopTimeShifted = synchStopTime
   elif(dataObjInput.calibClickCount > 0):
      plotStartTime = displayStartTime
      displayStartTimeShifted = displayStartTime + dataObjInput.timeDelay
      displayStopTimeShifted = displayStopTime + dataObjInput.timeDelay
   if(IMUOrViconInput == "BothPart1" or IMUOrViconInput == "Both"):
      inputTitleObj = appRoot.titleText3
      inputCanvas = appRoot.plotCanvas3
   else:
      inputTitleObj = dataObjInput.titleText
      inputCanvas = dataObjInput.plotCanvas
   if(clearPrevious):
      inputCanvas.delete("all")
   bPad = 4
   bX1 = bPad
   bX2 = appRoot.plotWidth - bPad
   bY1 = bPad
   bY2 = appRoot.plotHeight - bPad
   pointPaddingX = 25
   pointPaddingY = 10
   xBase = bX1 + pointPaddingX
   xSpan = ((appRoot.plotWidth - bPad) - pointPaddingX) - xBase # Max x - min x within a scroll screen

   if((plotType == "AllAcc" or plotType == "PeakAndAllAcc") or dataObjInput.calibClickCount == 0): #
All Points Plot
##       appRoot.stepCountText['text'] = "Plotting all points"
      accTimes = []
      accYs = []
      accXs = []
      for timeInc in range(len(dataObjInput.allAccQueue.queue)):
         incList = dataObjInput.allAccQueue.queue[timeInc]
         thisTime = incList[0]*dataObjInput.dataTimeScale
```

```
        if(thisTime > displayStartTimeShifted):
##          if(thisTime > displayStartTimeShifted and thisTime < displayStopTimeShifted):
##              if(dataObjInput.calibClickCount > 0 or thisTime < displayStopTimeShifted):
            accTimes.append(thisTime)
            accYs.append(incList[2])
            if(plotXAcc):
                accXs.append(incList[1])
    dataObjInput.absMaxPeak = abs(max(accYs))
    effectiveAbsMaxPeak = IMUGlobal.absMaxPeak * IMUGlobal.accScale / dataObjInput.accScale # Displays IMU on same scale as Vicon
##      dataObjInput.accScale = 1/absMaxPeak
    maxTime = displayStopTimeShifted
    pointMaxTime = accTimes[len(accTimes)-1] - dataObjInput.timeDelay
    timeSpan = (maxTime - dataObjInput.timeDelay) - plotStartTime
    yBase = appRoot.plotHeight/2
    ySpan = (bY1 + pointPaddingY) - yBase # How far the highest y point is from the lowest (needs yBase)
    xScale = float(xSpan)/(timeSpan)
    yScale = float(ySpan)/effectiveAbsMaxPeak
    scaleInfo = [xBase, xScale, yBase, yScale]
    if(IMUOrViconInput != "BothPart1" and IMUOrViconInput != "Both"):
        if(dataObjInput.calibClickCount == 0):
            plotColor = "orange"
            windowData = [accTimes, accYs]
            inputCanvas.unbind("<Button 1>") # neccesary to keep previous from lingering windowData = [accTimes, accYs]
            inputCanvas.bind("<Button 1>", lambda event: onCalibClick(event, IMUOrViconInput, appRoot, scaleInfo, windowData), inputCanvas)
        else:
            windowData = [accTimes, accYs]
            inputCanvas.unbind("<Button 1>")
            inputCanvas.bind("<Button 1>", lambda event: onCalibClick(event, IMUOrViconInput, appRoot, scaleInfo, windowData), inputCanvas)
    lineCoordList = []
    for accInd in range(len(accYs)):
        pointX = xBase + xScale*(accTimes[accInd] - dataObjInput.timeDelay - plotStartTime)
        pointY = yBase + yScale*accYs[accInd]
        lineCoordList.append(pointX)
        lineCoordList.append(pointY)
    inputCanvas.create_line(lineCoordList, fill=plotColor)
    lineCoordListX = []
    if(IMUOrViconInput != "BothPart1" and IMUOrViconInput != "Both" and len(accXs) > 0):
        for accInd in range(len(accXs)):
            pointX = xBase + xScale*(accTimes[accInd] - dataObjInput.timeDelay - plotStartTime)
            pointY = yBase + yScale*accXs[accInd] * XAccScale
            lineCoordListX.append(pointX)
            lineCoordListX.append(pointY)
        inputCanvas.create_line(lineCoordListX, fill=plotColorX)
  if((plotType == "Peak" or plotType == "PeakAndAllAcc") and dataObjInput.calibClickCount > 0): # Peak Plot
```

```
      if(IMUOrViconInput != "BothPart1" and IMUOrViconInput != "Both"): # Peaks disables for both
plot
##        if(IMUOrViconInput != "Both"): # Plots IMU but not Vicon peaks when doing both
          # Peak Data formatting
          peakTimes = []
          LPeaks = [] # Include zeroes for when the other is a peak so peakTimes matches length
          RPeaks = []
          errorMaybePeaks = []
   ##       peakTimes.append(dataObjInput.timeDelay) # A starting point is added to "anchor" the plot
start
   ##       LPeaks.append(0)
   ##       RPeaks.append(0)
          peakCount = 0;
          for timeInc in range(len(dataObjInput.peakQueue.queue)):
             incList = dataObjInput.peakQueue.queue[timeInc]
             thisTime = float(incList[0])*dataObjInput.dataTimeScale
   ##          if(thisTime > displayStartTimeShifted and thisTime < displayStopTimeShifted):
             if(thisTime > displayStartTimeShifted):
                LHeight = incList[8]
                RHeight = incList[9]
                isSuspectedPeakError = incList[7]
                if(LHeight != 0 or RHeight != 0): # Filters out how in some options mins are marked with 0s
                   peakTimes.append(thisTime)
                   LPeaks.append(LHeight)
                   RPeaks.append(RHeight)
                   errorMaybePeaks.append(isSuspectedPeakError)
                   peakCount = peakCount + 1
          peakArrays = [LPeaks, RPeaks, errorMaybePeaks]
##          peakArrays = [LPeaks, RPeaks]
          peakArrayColors = ["blue", "red", "orange"]
   ##       appRoot.stepCountText['text']= "Plotting "+ str(peakCount) +" peaks (in window)"
##          if(plotType == "Peak"): # Peak only plot
##             maxPeak = max(max(LPeaks), max(RPeaks))
##             minPeak = min(min(LPeaks), min(RPeaks)) # should be 0 for maxes
##             maxTime = displayStopTimeShifted
##             pointMaxTime = peakTimes[len(peakTimes)-1] - dataObjInput.timeDelay
##             timeSpan = (maxTime - dataObjInput.timeDelay) - plotStartTime
##             absMaxPeak = max(abs(maxPeak), abs(minPeak))
##
##             if(minPeak >= 0 or disregardNegativePeaks):
##                yBase = bY2 - pointPaddingY # For when only positive peaks plotted
##             else:
##                yBase = appRoot.plotHeight/2
##             ySpan = (bY1 + pointPaddingY) - yBase # How far the highest y point is from the lowest
(needs yBase)
##       ##       inputCanvas.create_line(xBase, yBase, xBase + xSpan, yBase + ySpan, fill="green") # test
line from minx miny to maxx maxy
##             xScale = float(xSpan)/(timeSpan)
##             yScale = float(ySpan)/absMaxPeak

          suspectedErrorCount = 0
```

```
            for peakArrayInd in range(len(peakArrays)):
                thisPeakArray = peakArrays[peakArrayInd]
                thisPeakColor = peakArrayColors[peakArrayInd]
                for peakInd in range(len(thisPeakArray)):
                    thisPeakVal = float(thisPeakArray[peakInd])
                    if(thisPeakVal != 0):
                        peakTime = float(peakTimes[peakInd])
                        pXBase = xBase + xScale*(peakTime - dataObjInput.timeDelay - plotStartTime)
                        if(peakArrayInd == 2): # Potential error markers
                            pR = 5
                            pW = 4
                            pYBase = 4
                            suspectedErrorCount = suspectedErrorCount + 1
                        else: # Normal peaks
                            pR = 3
                            pW = 2
                            pYBase = yBase - yScale*thisPeakVal
                        pX0 = pXBase - pR
                        pY0 = pYBase - pR
                        pX1 = pXBase + pR
                        pY1 = pYBase + pR
                        inputCanvas.create_oval(pX0, pY0, pX1, pY1, outline=thisPeakColor, width=pW)
##          if(len(peakArrays) == 3 and ):
##              print(str(suspectedErrorCount) +" suspected min/max mismatches in "+ IMUOrViconInput
+" data")
    if(IMUOrViconInput == "Both"): # HS lines
        hsArrayColors = ["blue", "red", "orange"]
        heelStrikesL = []
        heelStrikesR = []
        for hsInc in range(len(dataObjInput.heelStrikeQueue.queue)):
            thisHs = dataObjInput.heelStrikeQueue.queue[hsInc]
            if(thisHs[0] > displayStartTimeShifted):
                if(thisHs[1] == "L"):
                    heelStrikesL.append(thisHs[0])
                elif(thisHs[1] == "R"):
                    heelStrikesR.append(thisHs[0])
        heelStrikeArrays = [heelStrikesL, heelStrikesR, dataObjInput.HSErrors]
        for hsArrayInd in range(len(heelStrikeArrays)):
            thisHsArray = heelStrikeArrays[hsArrayInd]
            thisHsColor = hsArrayColors[hsArrayInd]
            for hsInd in range(len(thisHsArray)):
                if(hsArrayInd == 2): # Error markers
                    pW = 2
                else:
                    pW = 1
                hsTime = float(thisHsArray[hsInd])
                pXBase = xBase + xScale*(hsTime - dataObjInput.timeDelay - plotStartTime)
                inputCanvas.create_line(pXBase, yBase + ySpan, pXBase, yBase - ySpan, fill=thisHsColor,
width=pW)
##          if(dataObjInput.HSErrors != [] and dataObjInput.calibClickCount != 0): # HS error lines
##              for thisHsError in dataObjInput.HSErrors:
```

```
##             if(thisHsError != dataObjInput.HSErrors[0]):
##                 print("Crossover marker at "+ str(thisHsError))
   if(dataObjInput.hsPeaks != [] and dataObjInput.calibClickCount != 0): # HS based peaks
      for thisHsPeak in dataObjInput.hsPeaks:
         if(thisHsPeak[0] != ""):
            hsPeakTime = thisHsPeak[0] + dataObjInput.timeDelay # Adds this back in since it was
subtracted to output
            pXBase = xBase + xScale*(hsPeakTime - dataObjInput.timeDelay - plotStartTime)
            pYBase = yBase + yScale*thisHsPeak[1] / dataObjInput.accScale # Un-scales since this gets
scaled for output
            if(IMUOrViconInput == "IMU" or IMUOrViconInput == "BothPart1"): # plots IMU plus
thinner
               hsW = 1
            else:
               hsW = 2
            if(thisHsPeak[2] == "L"):
               hsPeakColor = "blue"
            elif(thisHsPeak[2] == "R"):
               hsPeakColor = "red"
            else:
               hsPeakColor = "orange"
            inputCanvas.create_line(pXBase, pYBase + 10, pXBase, pYBase - 10, fill=hsPeakColor,
width=hsW)
            inputCanvas.create_line(pXBase + 10, pYBase, pXBase - 10, pYBase, fill=hsPeakColor,
width=hsW)

   if(dataObjInput.calibClickCount == 0):
      titleString = "Click on the first of the three calibration peaks to calibrate start time"
      plotTimeSpan = synchStopTime - displayStartTime
   else:
##       titleString = componentToOutputOn +" data from "+ dataObjInput.fileToOpen
      if(IMUOrViconInput == "BothPart1" or IMUOrViconInput == "Both"):
         titleString = "Combined IMU/Vicon data from "+ subjectId +"_"+ trialNumber
      else:
         titleString = "Data from "+ dataObjInput.fileToOpen
         titleString = titleString +" ("+ IMUOrViconInput +" data trial "+ trialNumber +")"
      plotTimeSpan = displayTimeSpan
      if(dataObjInput.calibClickCount == 0):
         print(IMUOrViconInput +" Calibrated max point time is "+ str(pointMaxTime) +" (raw is "+
str(pointMaxTime+dataObjInput.timeDelay) +")")
##    inputCanvas.create_text(appRoot.plotWidth/2, bY1 + 2,fill="black", font="Times 14",
text=titleString, anchor="n")
   inputTitleObj['text'] = titleString
   scrollWidth = (appRoot.plotWidth - pointPaddingX*2- bPad*2) * pointMaxTime / plotTimeSpan +
pointPaddingX*2 + bPad*2
   inputCanvas.config(scrollregion=(0,0,scrollWidth,appRoot.plotHeight))
   inputCanvas.create_line(bX1, bY1, scrollWidth-bPad*2, bY1, scrollWidth-bPad*2, bY2, bX1, bY2,
bX1, bY1) # Frame
   inputCanvas.create_line(0, yBase, scrollWidth, yBase, fill="black", dash=(7,1)) # Y base line
   if(dataObjInput.calibClickCount > 0):
##       bottomMarkerFractions = [0, 0.25, 0.5, 0.75, 1]
```

```
        bottomMarkerTime = plotStartTime
##      for bottomMarkerFraction in bottomMarkerFractions:
        while bottomMarkerTime < pointMaxTime:
##          bottomMarkerFraction = bottomMarkerFractions[bottomMarkerInc]
##          bottomMarkerTime = timeSpan*bottomMarkerFraction + plotStartTime
            bottomMarkerTimeRound = str(round(bottomMarkerTime, 2))
            bottomMarkerX = xBase + xScale * (bottomMarkerTime - plotStartTime)
            inputCanvas.create_text(bottomMarkerX, bY2 ,fill="black", font="Times 10",
text=bottomMarkerTimeRound, anchor="s")
            inputCanvas.create_line(bottomMarkerX, bY2 -21, bottomMarkerX, bY2 -21-10, fill="black") #
Marker Tick
            bottomMarkerTime = bottomMarkerTime + 5
    if(IMUOrViconInput == "BothPart1"):
        doPlotting(appRoot, "Both", False)
    if(dataObjInput.notDoneBothPlot and IMUOrViconInput == "IMU" and
dataObjInput.storedTimeDelay != ""):
        dataObjInput.notDoneBothPlot = False
        print("Using stored calibration to plot comparison and make output file.")
        doBothPlotAndOutput(appRoot)

global IMUGlobal
IMUGlobal = IMUOrViconObj("IMU");
global ViconGlobal
ViconGlobal = IMUOrViconObj("Vicon");

doReadFile("IMU")
doReadFile("Vicon")

outGraph = doOutputDataPlots()
outGraph.mainloop()
```