

The Design and Validation of a Force and Bending Moment Sensing Device

By

Alexander Hodes

Submitted to the graduate degree program in Bioengineering and the Graduate Faculty
of the University of Kansas in partial fulfillment of the requirements for the degree of
Master of Science

Sarah L. Kieweg, Chair

Sara E. Wilson, Co-chair

Carl P. Weiner, Committee Member

Date defended: August 17, 2015

The Thesis Committee for Alexander Hodes
certifies that this is the approved version of the following thesis:

The Development and Validation of a Force and Moment Sensing Device

Sarah L. Kieweg, Chair

Date approved: August 18, 2015

Abstract

Shoulder dystocia (SD) is a serious obstetric emergency in which an infant's shoulder becomes blocked by the mother's pubic symphysis and the infant is unable to be delivered. In these instances, the infant must be delivered hastily, yet this hurried extraction may lead to an increase in clinician-applied forces. Injury risk is as high as 10% of all SD cases as it is shown that excessive force applied on the head and neck of the infant can lead to birthing injuries, specifically brachial plexus injuries (BPI) [2-4]. Several groups speculate that the direction of force applied to an infant head and neck is important in determining the amount of brachial plexus stretch of an infant [1, 5]. In this study we discuss the development and validation of a force and bending moment sensing glove that will be used to investigate clinician-applied forces.

The developed glove uses an Arduino microcontroller, pressure sensors, inertial sensors, and magnetic sensors to determine orientation and forces sensed by the device. In order to ensure the glove system performed correctly, our device was validated against external systems to assess accuracy of the developed glove. The system was first validated for orientation by placing electromagnetic sensors from an external motion detection system (accuracy $<0.5^\circ$) on a hand mannequin along with the inertial and magnetic sensors (IMUs) of the developed device. Inertial and electromagnetic sensors were placed on the back of each fingertip as well as the hand. After comparing 14 different hand orientations, the average root-mean-square error (RMSE) between calculated Euler angles from the glove device and electromagnetic sensors was less than 16° in any direction.

Force and moment vectors were then calculated by the glove device and validated by applying forces from the glove on a Styrofoam sphere attached to a straightedge clamped to a force plate (*Bertec* 4060-NC) with a force range of 0 - 2,500 N and resolution of ± 0.5 N. Nine different orientations were tested and the force and moment vectors were calculated from each system. The force and moment vectors from the glove were ordered in magnitude. The glove primary force vector component (F_1) and primary moment component (M_1) were compared to their corresponding force plate direction components. Comparisons were made using a root mean squared value (RMSE) and average error (AVGE) over each validation test. The glove was capable of measuring the bending moment component (M_{o1}) and force component with the largest magnitude (F_1). The average percent error for primary force component was found to be 5.85%. F_1 retains this accuracy in any direction the force is applied. Improvements can be made in the glove design to increase accuracy and ease of use.

Acknowledgements

I first would like to thank Dr. Kieweg and Dr. Wilson for their continued guidance, suggestion of ideas, and support. Without their great ideas and continued support I would not have been able to complete this project. I would also like to thank Dr. Weiner for the time he has taken to make suggestions and improvements on the project. I would also like to thank my laboratory members for their continuous help and positivity with anything that came up during the project. My lab members were extremely helpful and were always willing to answer questions and help me in any way. I also would like to thank Dr. Huazhen Fang for his patience and offering help whenever I needed. I would also like to thank Al Syvongsay who helped solder the device and was extremely helpful in turning a schematic into a prototype. I acknowledge the members of my examination committee, Dr. Sarah Kieweg, Dr. Sara Wilson, and Dr. Carl Weiner for their review of my work. I would also like to thank my family and friends who have continuously provided me with advice and support throughout my academic career.

Table of Contents

| | |
|--|-----|
| Abstract | ii |
| Acknowledgements | iii |
| List of Figures..... | vii |
| List of Tables..... | ix |
| Chapter 1: Introduction..... | 1 |
| Thesis Structure | 3 |
| Chapter 2: Background and Significance | 4 |
| A. Background..... | 4 |
| B. Preliminary Studies | 10 |
| C. Concurrent Studies | 13 |
| D. Significance | 14 |
| E. Prior Art..... | 16 |
| F. Innovation | 20 |
| G. Summary of Aims | 22 |
| Chapter 3: Development and Validation..... | 23 |
| A. Introduction | 23 |
| B. Design and Validation Methods | 25 |
| B.1 Instrumentation Design | 25 |
| B.2 Sensor and Data Analysis..... | 28 |
| B.3 Experimental Validation | 40 |
| C. Results..... | 46 |
| C.1 Motion Validation | 46 |

| | |
|---|-----|
| C.2 Force Validation | 51 |
| D. Discussion | 55 |
| E. Conclusion | 61 |
| Chapter 4: Design Process | 63 |
| A. Introduction | 63 |
| B. Previous Glove Design | 64 |
| C. Functioning Environment | 65 |
| D. Qualitative Needs | 67 |
| E. Design Specifications..... | 67 |
| F. Glove Design Choices..... | 68 |
| G. New Glove Design..... | 70 |
| G.1 Sensors and Hardware..... | 70 |
| H. Challenges..... | 79 |
| I. Final Glove Design | 89 |
| I.1 Electrical Schematic | 89 |
| I.2 Data Acquisition and Analysis | 96 |
| I.3 Final Design Specifications | 113 |
| Chapter 5: Conclusion..... | 114 |
| References..... | 119 |
| Appendix | 123 |

List of Figures

| | |
|---|----|
| Figure 2.1: Shoulder dystocia maternal/fetal diagram | 4 |
| Figure 2.2: Brachial plexus anatomy | 5 |
| Figure 2.3: Previously developed gloves | 11 |
| Figure 3.1: Sensor placement | 26 |
| Figure 3.2: Gravity vector projection..... | 29 |
| Figure 3.3: Filter algorithm flowchart | 37 |
| Figure 3.4: Force testing apparatus | 39 |
| Figure 3.5: Glove sensors and wiring | 41 |
| Figure 3.6: Motion testing orientations | 43 |
| Figure 3.7: Force testing orientations | 44 |
| Figure 3.8: Linear regression electromagnetic sensors vs glove sensor motion results | 50 |
| Figure 3.9: Linear regression force magnitudes | 52 |
| Figure 3.10: Linear regression F_1 | 53 |
| Figure 3.11: Linear regression F_2 | 54 |
| Figure 3.12: Linear regression Mo_1 | 55 |
| Figure 4.1: Pressure sensitive film | 65 |
| Figure 4.2: Sensor placement on hand | 69 |
| Figure 4.3: Euler angle coordinate plane..... | 70 |
| Figure 4.4: Drift from gyroscope | 72 |
| Figure 4.5: Flexiforce voltage circuit..... | 73 |
| Figure 4.6: Accelerometer calibration..... | 76 |
| Figure 4.7: Pressure sensor calibration curve | 78 |

| | |
|---|-----|
| Figure 4.8: Motion capture testing apparatus | 83 |
| Figure 4.9: Electromagnetic sensor and IMU placement..... | 85 |
| Figure 4.10: Force testing apparatus..... | 87 |
| Figure 4.11: Force application diagram | 88 |
| Figure 4.12: Electrical schematic | 89 |
| Figure 4.13: Multiplexer channel selection | 90 |
| Figure 4.14: Multiplexer schematic..... | 92 |
| Figure 4.15: mpu6050 schematic | 93 |
| Figure 4.16: sensorstick schematic | 94 |
| Figure 4.17: Flexiforce pressure sensor | 95 |
| Figure 4.18: Flexiforce schematic | 95 |
| Figure 4.19: Arduino script flowchart..... | 98 |
| Figure 4.20: LabVIEW block diagram..... | 100 |
| Figure 4.21: Force testing coordinate plane | 107 |
| Figure 4.22: Time phases of tests | 109 |
| Figure 4.23: MATLAB script flowchart..... | 112 |
| Figure A.1: Euler angle offset calculations | 172 |
| Figure A.2: Scale factor calculations | 173 |
| Figure A.3: Electromagnetic and glove output data..... | 174 |

List of Tables

| | |
|---|----|
| Table 3.1: Sensor Specifications Used in Design..... | 27 |
| Table 3.2: Yaw Angle Offset Ratios for Each Sensor Used in Algorithm | 36 |
| Table 3.3: Force Scale Factors for Each Direction..... | 40 |
| Table 3.4: Euler Angle Offset Ratios for Each Sensor Used in Algorithm | 42 |
| Table 3.5: Error Comparison for Euler Angles Tested..... | 47 |
| Table 3.6: Error Comparison Among Different Time Phases | 47 |
| Table 3.7: Error Metrics Among Sensors and Euler Angles | 49 |
| Table 3.8: AVGE Two-way ANOVA Test Results..... | 49 |
| Table 3.9: Percent Error Results of Components Measured | 53 |

Chapter 1: Introduction

The objective of this project is to develop gloves that can measure forces and moments applied by the hand in six dimensions ($F_x, F_y, F_z, M_{ox}, M_{oy}, M_{oz}$). These gloves are designed to be worn by obstetricians and other clinicians while delivering infants in live births or with infant mannequins in simulated births. Shoulder dystocia (SD) is a potentially life threatening obstetrical emergency that occurs, typically when the anterior shoulder of the fetus is caught behind the symphysis pubis of the mother after delivery of the head. Injury can occur in up to 10% of all SD deliveries [3, 4]. Prior studies have identified risk factors for SD, but the majority of instances remain unpredictable, thus proper and gentle management of SD cases is crucial to prevention of neonatal injuries. It has been shown that SD simulation training is the most effective way of managing SD [4, 6]. Minimization of forces and moments experienced by the infant are critical in preventing injuries [7]. Other groups have quantified forces applied by a clinician during delivery but have neglected to record bending moments which have been suggested to result in brachial plexus stretch [1, 8-10]. Measurement of forces and moments applied by clinicians during birthing simulations provide quantitative values to assess training effectiveness, to provide feedback for simulation training, and allow us to further examine the biomechanics of brachial plexus injury (BPI).

Pressure sensing gloves have been previously used by our research group and were successfully used in birthing simulations as well as live deliveries [11]. However these gloves only measured the applied pressure and did not fully assess the applied force. Motion sensing capabilities allow us to calculate direction of pressure application and total force. Because the amount of BP stretch is thought to be dependent upon the

direction of forces applied to the head, it is vital to have the ability to measure directions of forces and moments to quantitatively assess of birthing simulations and live birth delivery. Thus, the overall objective is to develop a glove device that is capable of measuring forces and moments applied by the wearer. In order to accomplish our objective we followed these specific aims:

Aim 1. Design a microprocessor based device worn on a hand that can obtain applied force and orientation data. Pressure sensors, magnetometers, gyroscopes, and accelerometers will be wired to a microcontroller. Data needs to be read into a computer to analyze sensor data.

Aim 2. Develop the methodology of hand orientation calculation from the sensors in the glove and assess hand orientation measurement against an external motion capture system. Raw sensor data will be converted into rotational angle orientation data using MATLAB. We will compare motion data to that of trakSTAR motion sensing system to assess the microcontroller system's accuracy. ***Hypothesis: Motion sensing data will be accurate when compared to a known accurate motion sensing device in predicting hand posture.***

Aim 3. Develop the methodology of force calculation from hand orientation and pressure sensors and assess this calculation against an external force measurement. Using orientation data we will calculate force and moment vectors. We will then compare and validate the integrated force data to that of an externally mounted

load cell to confirm the accuracy of data. ***Hypothesis: Force and moment vector data will be accurate when compared to that of an external load cell.***

Thesis Structure

The thesis first describes the importance and significance of brachial plexus injury (BPI) occurrence in infant deliveries. Background information and previous work related to the motion sensing glove are first discussed in chapter 2. The third chapter is a manuscript that describes the design and validation of the gloves. This thesis describes the sensors used in development of the glove, algorithms and theory behind sensor fusion, as well as a validation of the gloves capabilities. Chapter 4 describes the design process of the glove including the final electrical schematic and design that was completed, as well as challenges faced in the design process. Chapter 5 is the conclusion and discusses what we can draw from our studies. The last chapter also discusses future studies and potential alterations to the current glove design. Limitations are also discussed in the current glove development.

Chapter 2: Background and Significance

A. Background

Shoulder dystocia (SD) is a potentially life threatening obstetrical emergency that typically occurs when the anterior shoulder of the fetus is caught behind the symphysis pubis of the mother after delivery of the head [3] (Fig. 2.1). Oxygen loss and potential death may result from prolonged entrapment of the fetus, however hasty extraction may lead to a brachial plexus injury (BPI). Thus, the delivery must be controlled to avoid either excess total force, or incorrect application of normal force [3]. BPI is a nerve praxis that if severe may permanently compromise shoulder and arm muscle control. BPI is as high as 10% of all SD deliveries in some hospitals and other relatively minor neonatal birthing injuries such as a clavicle break are even more common [4].

The brachial plexus (BP) is a network of nerves that travels through the neck and

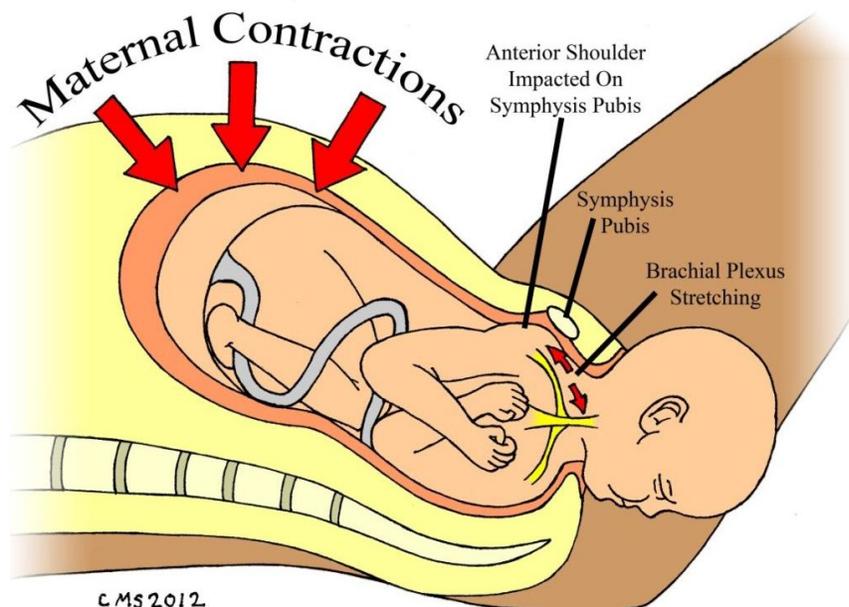


Fig. 2.1 Shoulder dystocia is an emergency when the shoulder of the infant is wedged behind the pubic bone as seen in the above picture. Brachial plexus nerves are labeled in the infant and may become stretched during delivery (Illustration reproduced with permission [1])

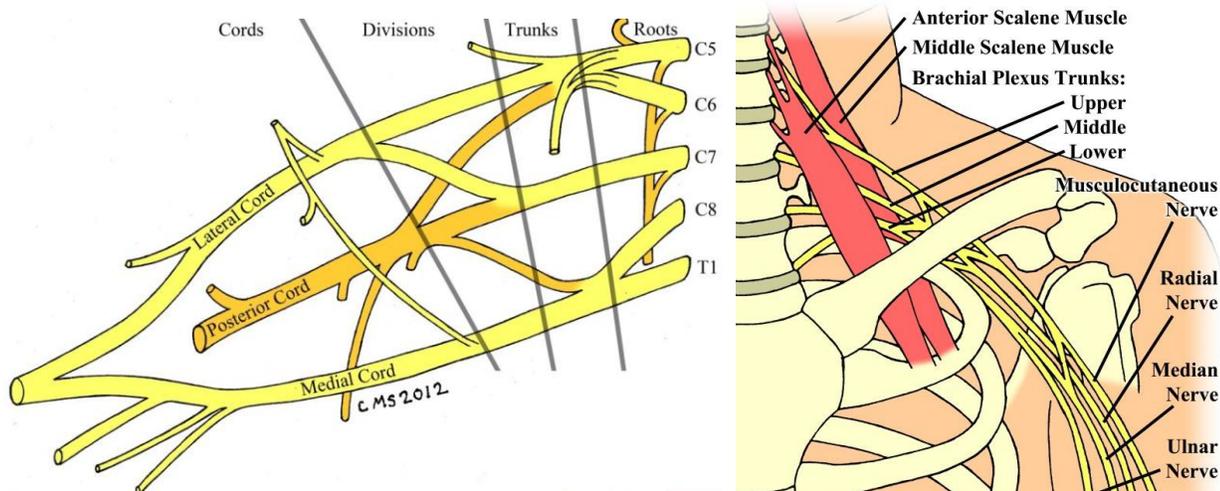


Fig. 2.2 The brachial plexus is a group of nerves that originate at the spine from C5, C6, C7, C8 and T1. The nerves are attached in trunks as seen above and eventually divide into three separate cords that control radial, medial, and ulnar musculature. The image on the right displays the location of the brachial plexus in the body. (Illustration reproduced with permission [1])

into the arm, extensive stretching of any of these nerves may result in injury [12]. The nerves comprising the BP originate at the C5, C6, C7, C8, and T1 sections of the spinal cord (Fig. 2.2). These are the roots of the nerves which come together in three separate trunks; superior, medial, and inferior trunks. These then divide into separate into three cords called the lateral, posterior, and medial cords which terminate in branches that each are responsible for muscular control. Elbow flexion, deltoid muscle contraction, digital extension, and hand control are the main functions of the BP [13]. The most common locations of injury are at C5 and C6 roots stretching by the head and neck being bent toward the opposite shoulder. The magnitude of stretch that occurs in the nerve is important as well as the rate at which the nerve is stretched at [14]. Injuries

may vary from a stretching of axons to complete avulsion (tearing) of nerves which may be permanent even after surgery [15].

Transient injury occurs in about 10% of all of SD cases, and the injury is permanent in about 1% of those (1.5 out of 1000 live births) [4]. Prior studies have identified risk factors for SD including: maternal obesity, diabetes, postdated pregnancy, advanced maternal age and a history of shoulder dystocia [15-18]. Additionally, abnormalities in maternal anatomy such as short stature and abnormal pelvic anatomy can increase risk [17]. Several studies have shown that the risk of complication from shoulder dystocia is correlated to the infant's birth weight [15, 19, 20]. Although there are risk factors for SD the majority of cases are unpredictable. Clinical management of SD cases is of utmost importance because of the unpredictability of SD cases [7, 10]. In order to manage SD deliveries, maneuvers have been developed to attempt to alleviate the emergency [21]. A series of maneuvers commonly used are the McRoberts maneuver, suprapubic pressure, delivery of the posterior arm, the Gaskin maneuver, and rotational maneuvers including Rubin II and Woods screw maneuver [21]. The McRoberts maneuver is one of the most commonly used in which the mother's legs are flexed tightly to the woman's abdomen to provide more room for the shoulder to be released. The best maneuver or the ideal sequence of maneuvers is unknown and is not always successful in resolving the dystocia [6].

Several studies have concluded that hands-on training using SD simulation based techniques is the most efficient way to teach care givers how to manage SD while posing minimal risk to the newborn and mother [7, 8]. SD simulation techniques use a rudimentary mannequin representing a mother's pelvis and an infant mannequin

[1, 7, 10]. Hands on training has been proven to reduce the amount of BPI in practice but this training neglects to include quantitative force assessment involved in mock deliveries or live deliveries [10]. As BPI is related to the amount of nerve stretching experienced and force on the head and neck can increase this stretching, it is important to determine forces experienced by the infant in mock deliveries [1, 5, 9, 11, 22].

Forces experienced by infants during delivery, specifically SD cases, have been explored by many research groups [2, 5, 8, 9, 11, 22-24]. Computer models have been developed to simulate forces exerted by maternal and clinicians during a routine and SD delivery and observe brachial plexus stretch [2, 5]. Grimm et al. used a computer model to examine brachial plexus stretch in SD scenarios and has included the use of maneuvers. Computer modeling does provide important findings, these findings are based upon models that are developed using assumptions that are not always exact; using caprine (goat) model stiffness of a pediatric neck and scaling the adult shoulder stiffness down to a pediatric model [2, 5]. Birthing simulators such as the PROMPT (*Laerdal*) and other simulations have been used to examine clinician applied forces and maternal forces [1, 7, 8, 11, 23, 24]. Though these simulations have positive effects on training outcomes, these still provide inconsistencies with a true live delivery [1, 7, 8, 11, 23, 24]. One inconsistency is that the resistance of delivery is determined by a variable human force holding an infant mannequin in the maternal pelvis and the infant mannequin does not have the same material properties of a live infant [1, 24]. Designing a tool that could improve birthing simulations would provide a better way to examine true forces experienced by an infant in SD and routine vaginal deliveries [1, 9, 22]. It is

important that the application of force must be examined during training for SD scenarios [24].

Several studies have quantified obstetrician hand contact pressure data in live deliveries [1, 9, 22]. Studies aimed to quantify peak force and peak impulse force which are important in determining injury [1, 9, 22]. However, these studies have neglected to include direction and bending moments [1, 9, 22]. Bending moments have been speculated to cause a large amount of brachial plexus stretch indicating the importance of the described parameter [1, 8, 23]. Directionality of force application has been speculated to be an important factor in determining brachial plexus stretch and injury, as an ear to shoulder bend provides most brachial plexus stretch [1, 5, 8, 23]. In order to obtain a force and bending moment directional vector, motion capture technology must be utilized in a device that can be worn by clinicians [1, 25, 26].

Human hand motion has been captured using several different sensors or systems; bending resistors, optic motion capture systems, inertial, and magnetic sensors [25, 26]. New devices such as the *Leap Motion Controller* use optic sensors such as infrared cameras to capture motion of the hands. However, these optic-based systems rely upon an external transmitter being able to track hands without any obstructions between the transmitter's line-of-sight [25, 26]. A glove used to track hand motion was developed using bending resistors at each finger joint [26]. This device has the capabilities of measuring finger bending but is unable to capture hand rotations and orientation which are important in determining the direction of force applied [26]. The *Acceleglove* (Meta Motion) is a device using accelerometers to obtain orientation of the hand of the glove wearer [25]. Though this glove is capable of capturing hand gestures

and orientation, it does not capture rotations in the *yaw* direction or around the *z-axis* of the hand which corresponds to radial and ulnar deviation [25]. The use of only accelerometers makes this device unable to capture a rotation where the gravity vector does not change and is also susceptible to error that is innate in accelerometers [25, 27-30]. Inertial sensors do hold an advantage over optic-based sensors and are the correct choice for developing a glove device to be used in live births by a clinician. However, there must be additional sensors integrated onto the glove to improve its capabilities [25, 27-30].

Accelerometers can be aided with gyroscopes and magnetometers to improve their motion tracking abilities, these are sometimes combined together to form inertial measurement units (IMUs) [25, 27-30]. An accelerometer alone has shortcomings, one is the inability to capture *yaw* rotations and its susceptibility to transient accelerations [25, 27-30]. Gyroscopes aid accelerometers as they measure angular velocity around each axis, including the *z-axis*, however these sensors are susceptible to drift while at rest [27-30]. Magnetometers are used in addition to accelerometers and gyroscopes to improve *yaw* rotations as magnetometers measure the earth's magnetic field [27-30]. Magnetometers do contain errors and these are due to external magnetic sources [27-30]. Several filters have been designed to fuse data from accelerometers, gyroscopes, and magnetometer to obtain rotation data in the form of quaternions, Euler angles, or rotation matrices [27-30]. The use of these three separate sensors produces an accurate way to measure hand orientation and direction of a hand [27-30]. With the addition of IMUs to a pressure sensing glove, calculating force and bending moments may be possible.

Minimization of forces and moments applied are likely critical in preventing injuries, but it is unknown whether maternal- or clinician-applied forces, or some combination of both, may cause injury [23]. Clinician-applied force is an important parameter to measure as studies demonstrate that significant risk to the infant can be minimized if the applied traction force is kept below 22.5 pounds or 100 Newtons [22, 24]. Traction force is defined by the force applied by pulling the head and neck outward [22]. Previous studies have quantified forces applied by a clinician during delivery but have neglected to record bending moments which are speculated to cause a large amount of brachial plexus stretch [5, 8, 22, 23]. Measurement of the forces and moments applied by clinicians during delivery could offer a quantitative tool to assess training effectiveness. Combined with maternal generated forces, such a tool would allow for the first time, the measurement of the total work required for delivery. Pressure sensing gloves have been previously developed by our research group and were successfully used in birthing simulations as well as live deliveries [1]. However, these developed gloves did not provide force measurement in three dimensions and it has been shown that the amount of BP stretch is dependent upon the direction of force the head experiences [23]. Because the amount of BP stretch is dependent upon the direction of forces applied to the head, it is vital to have the ability to measure directions of forces and moments during mock or live deliveries [23].

B. Preliminary Studies

Studies have concluded that hands-on training using shoulder dystocia (SD) simulation based techniques are an efficient way to learn how to manage SD cases [7,

10]. Our collaborators, Weiner et al., have implemented PRactical Obstetric Multidisciplinary Training (PROMPT) at the University of Kansas Hospital in labor and delivery [10]. PROMPT is a UK birthing simulation program made to improve obstetric outcomes by teaching proper management techniques during SD and other birthing difficulties [10]. Weiner et al. examined the incidence of Cesarean section delivery (C/S), SD, perinatal hypoxic ischemic encephalopathy (HIE), as well as brachial plexus injury (BPI) and the effect PROMPT training had on these measurements. Mandatory PROMPT simulation training was implemented at the University of Kansas Hospital and, over the course of five years, significantly decreased the rates of BPI/SD and C/S rate [10]. With appropriate simulation training, BPI/SD can be reduced during complicated deliveries but other quantifiable performance metrics to measure training course effectiveness have not yet been explored [10].

To address the need for quantitative assessment of forces applied during birthing delivery, our research group has previously used pressure sensing gloves that can be worn underneath sterile surgical gloves [1, 9]. Our research group used these gloves to measure clinician-applied forces directly and to provide a quantifiable measure during



Fig. 2.3 Previously developed gloves contained two sensors along each finger and contained two extra sensors along the lateral side of the hand (i.e. pinky side)

training simulations and live births [1, 9]. Pressure mapping was also performed during mock deliveries to determine location and concentration, on the hand, of the clinician-applied forces [1]. This initial pressure mapping was performed by placing Fujifilm Pressure Measurement Film Prescale, pressure sensitive film, cut in the shape of a hand and worn underneath a sterile glove [1]. Mock deliveries using a maternal and fetal mannequin were then performed to simulate what forces one would experience in a typical delivery [1]. It was concluded that the ring, middle, and pointer finger exerted significantly larger forces than the palm or the thumb [1]. The developed pressure sensing gloves had sensors along the length of each finger but neglected to include the palm, as it was shown that pressures exerted by the palm were negligible [1]. Pressure sensing gloves developed were worn underneath sterile latex gloves and were successfully used in mock deliveries as well as live deliveries [1, 9].

In these previous studies, the pressure sensing gloves were worn in simulations and it was shown that the sensors on the distal ends of each fingertip provided the largest amount of pressure [1]. Mock deliveries were useful to simulate live birth scenarios, including SD cases [1]. However, there were limitations to this training simulation as the investigator provided the movement and resistance of the mannequin may have differed from those experienced in a live delivery. The most important use of these developed gloves was in actual live deliveries which were performed in vaginal deliveries as well as C/S cases [9]. Results of these live deliveries suggested that SD cases do result in higher clinician-applied pressures [9]. This finding suggests that clinician-applied forces, specifically in SD cases must be further examined to see if they are a culprit in causing BPI or other neonatal injuries.

There were several problems when using the glove in mock and live deliveries; one was that the gloves had a different fit on different individuals causing slightly different measurements during each trial [1]. More problematic was that the pressure sensing glove only measured pressure applied and, because there was no data on direction or pressure application, could not be used to determine total force on the head of the fetus [1]. Direction of moments and forces applied to an infant's neck are important to include when exploring clinician-applied forces as brachial plexus stretch is associated with bending of the neck [5, 23]. The developed gloves provide 6 degrees of freedom: 3 dimensions in force and moment, providing more quantitative analysis for how clinician-applied forces may affect BPI in delivery. The developed gloves can also be used as a training tool for providing quantitative assessment during mock deliveries for clinicians partaking in the PROMPT training course.

C. Concurrent Studies

Our lab is currently developing a mathematical model that will quantify the relationship between forces applied to the fetal head and brachial plexus stretch, specifically to identify the threshold limits of the force and moments applied on the fetal head that will be below the nerve stretch required for a BPI. Using published experimental, mechanical, and geometric properties of the cervical spine, brachial plexus, and neck musculature a computational model will be developed using *Adams*, *OpenSim*, or *Simpleware* software. This is to be used in accordance with the results from trials of the developed gloves to further understand the affect clinician applied forces have on infant brachial plexus stretch and injury. An infant mannequin will be

developed that will have a head and neck that is instrumented with strain gauges. This mannequin will be used in the PROMPT simulation training course and can be used to further examine the effects of, not only clinician applied forces but maternal forces as well. This instrumented mannequin will include an instrumented brachial plexus nerve. These advancements to the mannequin will improve the accuracy of birthing simulations and will provide more quantitative data to assess these simulations.

D. Significance

Shoulder dystocia (SD) is a serious obstetric emergency and must be treated immediately [2, 3]. This emergency complicates up to 2% of all deliveries [15, 31]. This occurs when the fetus' anterior shoulder, relative to the mother, becomes wedged behind the mother's symphysis pubis bone after the infant's head has emerged during childbirth. Oxygen loss and potential death may result if the situation is not resolved quickly, yet hasty extraction may lead to the application of excessive force. The brachial plexus (BP) is a network of nerves that travels through the neck and into the arms. If the BP is injured during childbirth, a child could have total and irreversible paralysis, sensory loss, or loss of motor skills in the shoulder or arm of the affected side [31]. BPI's occur in 10% of all SD cases and neonatal clavicle fractures are even more common [4]. One way to alleviate SD cases is to perform Cesarean section (CS) [31]. It has been estimated that the prevention of one brachial plexus injury due to SD would cost more than \$9 million by performing 3500 unnecessary CS procedures and there are other risks involved in these procedures [3]. Although there are several risk factors indicating a SD case might occur, SD is not entirely predictable [3, 5, 7, 11, 23, 24, 31].

SD must be properly managed because SD is not predictable and the most successful means of teaching SD management is through simulation training [24]. Knowledge of clinician applied forces and moments is important in determining their effect on BPI and provides a quantitative measurement to assess the delivery of SD cases.

Birthing simulators and training have been developed to train clinicians how to manage SD cases as well as routine deliveries [32]. Mitrani et al. states that simulation is vital to train and provide feedback where initial operation by novices is impractical or dangerous [33]. Clinician-applied force is an important parameter to measure as studies demonstrate that significant risk to the infant during delivery can be minimized if the applied traction force is kept below 22.5 pounds or 100 Newtons [22, 24]. Several birthing simulators have force sensing capabilities in the fetal mannequin but neglect to include direction and bending moments applied by clinicians during these simulations [24]. Maternal and fetal mannequins have been developed by companies including Laerdal, Gaumard, and CAE Fidelis Healthcare. These developed mannequins do provide fetal monitoring including ECG, APGAR score and some provide traction force, however, these mannequins neglect to include force and moments applied in three directions during birthing simulations. The developed gloves provide important quantitative information that can be used as an assessment training tool which no other mannequins provide. Gloves provide a possibility of being used in live deliveries with real infants and could provide more realistic and relevant information than any birthing simulator and fetal mannequin could [1].

Research has shown there is a direct relationship between applied force to the infant's neck/head during SD and BP injuries, whether the excessive forces are

maternal or clinician-applied is unknown [2]. Metaizeau et al. showed that a lateral application of 44 pounds of traction to an infant head caused serious nerve damage, although the direction of traction force is not specific [34]. Mollberg et al. establishes that the number of damaged nerve roots in the BP is proportional to the force applied to the infant's head [35]. BP stretch and injury occur most when the head of the infant is maneuvered in an ear to shoulder direction [23]. Because BP stretch is dependent upon direction of head and neck bending, monitoring of clinician-applied forces and moments in three dimensions during live deliveries provides insightful quantitative measurements that can be related to BPI risk [5, 23].

E. Prior Art

Pressure sensing gloves have been described to be used in the obstetric field by a research group, but it does not capture motion tracking as our device does [22]. There are similar products and developments that have not been applied in the obstetric field but have similar capabilities of the glove we have developed. A variety of gloves are already on the market that measure hand and finger orientation and position including: the Cyberglove, P5 Glove™, 5DT Dataglove, Acceleglove, Hand Mentor, and the HandTutor [25, 36-39]. These devices are focused primarily on capturing orientation of the glove wearer to be used in a virtual environment or for hand therapy [37-40]. The Cyberglove uses bending resistors along fingers to determine orientation of fingers, has an on-board processor, and communicates wirelessly to an external system to analyze the data [39]. The company producing the Cyberglove also produces a device called the Cybertouch device which provides tactile feedback through the use of monitoring

tendon movements in the hand [36]. A device called the Cyberforce has also been developed which provides a force feedback to the wearer. This device contains a very obstructive apparatus and does not quantify the force of the user but provides a force feedback from the virtual environment [37]. This device may be the most sophisticated, however the large apparatus limits the application space for the device [37]. Forces are measured by the fingertips and transmitted to the hand through the use of an apparatus attached on the top of the hand using a force applicator and can simulate the grasping of an object in a virtual environment [37]. This device does not quantify force and moment vectors applied by the user as our developed device does.

The Hand Mentor and HandTutor use bending resistors to determine finger orientation and bending to be used in hand rehabilitation. Though these devices are able to measure flexion and extension of the wrist, these are not capable of measuring orientation of each finger individually. The P5 Glove™ is capable of measuring hand orientation and position. This glove uses bending resistors as well as infrared sensors to determine position. These infrared sensors also require an external infrared tower to transmit IR data [40]. A virtual environment or display is also output when this device is in communication with a computer. This device also uses “dead reckoning” methods to determine position, providing six degrees of freedom (x , y , z , $roll$, $pitch$, yaw). Sensors that could be used by this device to determine orientation and position could include: resistive bend sensors, angle measurement sensors, optical fiber sensors, Hall effect sensors, IMU sensors, a radiofrequency (RF) tracking system, and infrared sensors. This device also claims to use capacitive touch sensors on each finger to provide and record pinch data from a thumb touching any of the other fingers. Though this device is

capable of measuring a pinch force it does not aim to measure force and moment vectors applied by glove wearer. Optical sensors can provide accurate results but may limit the application of the device as a line-of-sight is required to be maintained and a small range may be the consequence of this requirement [40].

The 5DT Dataglove is a similar device to the previously mentioned gloves, it uses bending resistors to determine finger flexure as well as abduction between the fingers. Bluetooth technology can also be used on this device to transmit the glove orientation to a computer. This device as the other devices mentioned, does not contain force monitoring capabilities. The Aceleglove is a device designed more similar to our glove design as it uses accelerometers as opposed to bending resistors to determine hand and finger orientation. However, this glove does not capture yaw angles of hand rotations very well as accelerometers are unable to calculate this accurately alone. Though several these products do provide accurate ways of determining orientation of the hand and fingers, the products are unable to sense force and moment vectors applied by the user.

Several patents contain some of the same methodology and electrical circuitry as our group uses but there are significant differences between our technology and the existing devices. There are many commercially available products that solely measure hand and finger orientation, many patents aim to measure forces for feedback in virtual reality settings. A device has been invented that assists the grasping of an object. This uses force sensors as well as electromyography in addition to a microcontroller to measure a grasping force applied by the user [41]. This measurement is then used to assist the grasping of an object by applying a tensile force to a tendon apparatus on the

glove and attached sleeve of the device. Force calculations are tensile forces in this application and differ widely from our device. Hand orientation is also not determined by the patented idea.

Another patent describes a device that studies grasping force though the methodology is vastly different [42]. This patent uses a fingerprint sensor that contains a six axis force and torque sensor. The fingerprint sensor aids the system by measuring the contact orientation of the finger. This device provides torque or moment vector as well as the force vector applied by the fingertips. This device obtains the same data we aim to record however the device application differs and methodology differs. This device is to be placed onto a physical object to which the grasping forces and moments are measured. The torque and force sensor is also significantly more costly than pressure sensitive resistors that were used in our glove design. These sensors also may provide a larger profile if integrated into a glove.

A separate glove device utilizes fiber optic sensors embedded in a glove [43]. The optic signal is attenuated in response to forces applied at the tips of the device where an applicator deforms from forces applied. A gyroscope and accelerometer are also attached to the top of the glove device to measure the rotation of a leg or arm rotation that the glove user is holding onto. Orientation of the glove itself is not determined using the gyroscope and accelerometer device. Direction and magnitude of force is found using the increase or decrease in Bragg grating periods in the fiber optic sensors. Shear forces are also captured using grating sensors. Bending moments are not measured by the device and this device utilizes fiber optic sensors as opposed to IMUs that our group uses. This device is unable to measure the orientation of each

finger and hand at a time in space but measure the forces applied by the wearer. A graphical display of force is included in the patent design.

A similar patent to our glove, describes a device used to measure forces experienced by the glove and is to be used to determine muscle tone of a patient being examined by the glove wearer [44]. This glove measures moment as well as force vectors applied to the glove from a body part the glove is grasping. Use of bending resistors, potentiometers, and optical fibers can calculate joint angles that are used to determine forces vectors applied to the glove device. Infrared distance sensors in the device can measure the distance of a moment arm to calculate a moment vector. This device does differ from ours in that our device measures the hand and finger orientation through the use of IMUs and can determine yaw angle in which the patented device would not be capable of measuring.

F. Innovation

The developed glove is innovative because there are no other gloves that simultaneously measure bending moments and forces applied by the glove user in three dimensions. There are other gloves and force monitoring systems that have been used to monitor gestures and hand direction, but neglect to include pressure sensors in the system [8, 22]. The *Acceleglove (Meta Motion)* is a glove that contains inertial monitoring sensors used to recognize gestures and hand orientation. Several research groups have examined the applied forces of clinicians on the head and neck of an infant [9]. However, previous efforts neglect to incorporate the direction and bending moment,

both of which are important in determining the amount of BP stretch an infant experiences [1, 5, 9, 11].

By using the developed gloves, delivery training programs can integrate the magnitude and direction of applied forces as well as the bending moments around the neck to attain a comprehensive analysis of acceptable and unacceptable force application on the baby. Knowledge of directional force and bending moments applied by an obstetrician in will provide data that has not yet been explored. Using IMU sensors and variable resistor pressure sensors integrated to determine force and moment vectors is innovative. This technology will advance the field of obstetrics by equipping simulation-based training programs with a quantitative knowledge of clinician-applied forces. This device can be used as a tool to assess training course effectiveness and can be used to examine directional forces and moments in live deliveries and SD cases.

Previous research groups have developed ways to sense force during SD cases have neglected to include the direction of force applied [1, 9, 22]. We have taken this idea to the next step by incorporating motion sensing technology by means of magnetometers, gyroscopes and accelerometers. The developed glove is innovative because inertial and magnetic sensors are able to sense direction of the force and moment application while being minimally obstructive. Incorporating the moment, the direction and magnitude of force, delivery training programs can assess trainees during birthing simulations. The developed glove can also be used as a research tool in live births to determine what type of forces and moments observed during live deliveries.

G. Summary of Aims

The design of a microprocessor based device consisted of inertial sensors, magnetic sensors, and analog pressure sensors. Raw sensor data will be converted to orientation angles to be used in motion sensing validation studies. The device will be assessed comparing orientation data in the form of Euler angles to the trakSTAR motion sensing system. Force measurement capabilities will then be assessed by comparing calculated force and moment vector data from the glove to force plate data. Once the accuracy of the microcontroller system is assessed, the system will be able to be integrated into a wearable product. The next steps will be to integrate the device into a wearable glove that must fit underneath a latex glove. A latex glove would need to be worn over this glove in order for this glove to be used in live deliveries. Data received from the developed glove can be used to further investigate the forces and moments applied by clinicians during mock and live deliveries of infants.

Chapter 3: Development and Validation

A. Introduction

Force identification applied by a human hand and hand posture has many different applications in robotic surgery, surgical training, obstetric training, physical therapy training, and control applications [1, 8, 9, 22, 27, 28, 45]. Capturing posture and orientation of a hand can be accomplished through the use of bending resistors, optical sensors, or inertial and magnetic sensors [45]. Inertial and magnetic sensors provide advantages over optical sensors as they do not require an external optical source and they provide more information regarding orientation than bending resistors do [27, 28, 45]. Inertial monitoring units (IMUs) containing a combination of accelerometers, gyroscopes, and magnetometers are commonly used to provide orientation data [27, 28, 45]. Previous developments from our research group developed pressure-sensing gloves but neglected to include orientation [1]. In this project our research group coupled pressure sensors with IMUs to create a microcontroller device that determines both force and bending moment vectors.

Converting IMU data into orientation data can be accomplished in various ways [27, 28, 45]. Data from accelerometers, gyroscopes, and magnetometers must be fused together in an ideal way to determine orientation accurately [27, 28, 45].

Accelerometers, gyroscopes, and magnetometers each are susceptible to specific inherent errors that must be accounted for [27, 28, 45]. Errors occur in accelerometers when transient accelerations are experienced by the sensor, but these sensors are reliable over longer time periods than gyroscopes [27, 28, 45, 46]. Gyroscopes measure rotational velocity and are prone to drift errors due to integration which provide

inaccuracies over a long time period, but are more accurate than accelerometers over short time periods [27, 28, 45, 46]. Magnetometers are used to aid rotations around the z-axis or yaw rotations. Rotations around the z-axis are not measured by the accelerometer as the gravity vector does not change [27, 28]. Magnetometer data may accumulate errors due to external magnetic field sources such as ferrous metals [27, 28, 45, 46]. Fusing data from each of the IMU sensors requires algorithms that take advantage of this knowledge of errors that occur in each individual sensor [27, 28, 45, 46].

This paper uses a combination of vector and direction cosine matrix (DCM) based complementary filters to fuse IMU data and output orientations in the form of Euler angles [45-47]. Euler angles are the amount of rotation around each axis that the rigid body being described has progressed in time [47]. The complementary filter uses the accelerometer and magnetometer vectors to determine current orientation and then uses gyroscope data over short periods of time to calculate transient orientation changes [45-47]. Analog pressure sensors were used to determine forces applied by the device user. Using forces and orientations sensed, force and moment vectors were determined in three dimensions.

Accuracy of orientation was determined by placing the IMUs from the device onto a hand mannequin along with electromagnetic sensors. A bendable arm is attached to the hand mannequin used and can be clamped to a sturdy surface. The hand mannequin was then moved to known orientations and the Euler angles output from the device were compared with the Euler angles from the electromagnetic sensors. Euler angle rotation sequences for both sensor systems were 3-2-1 or ψ - α - ϕ . A second test

was performed to analyze force and moment sensing capabilities of the device. IMUs and pressure sensors were attached to a hand mannequin. A spherical object was firmly attached to a force plate and the pressure sensors on the hand mannequin were placed on the spherical object and forces were applied. The orientation of the hand mannequin was changed and forces normal to the mannequin were applied. This paper explores the development of a microcontroller-based device and the validation of its capabilities.

B. Design and Validation Methods

B.1 Instrumentation Design

The device is composed of an Arduino microcontroller (Atmel AT91SAM3X8E), six accelerometers (Analog Devices ADXL345), six gyroscopes (InvenSense ITG-3200), one 3-axis digital magnetometer (Honeywell HMC5883L), and five pressure sensors (Tekscan ESS301). A more in-depth final design is described in Chapter 4. Five sensors contain an accelerometer and gyroscope integrated onto one chip (Ivensesense MPU6050). These five sensors were each placed on the back of the fingertip of the hand mannequin (YCCTEAM Flexible Nail Trainer), labeled as rectangles in Fig. 3.1. The other IMU contains a magnetometer, accelerometer, and gyroscope which were placed on the center of the back of the hand mannequin. Each pressure sensor was placed on the fingertip of the mannequin. Fig. 3.1 illustrates the placement of each sensor on the hand. Sensor specifications are shown in Table 3.1. Pressure sensors are able to measure forces ranging from 0-133 N which is sufficient in our application as

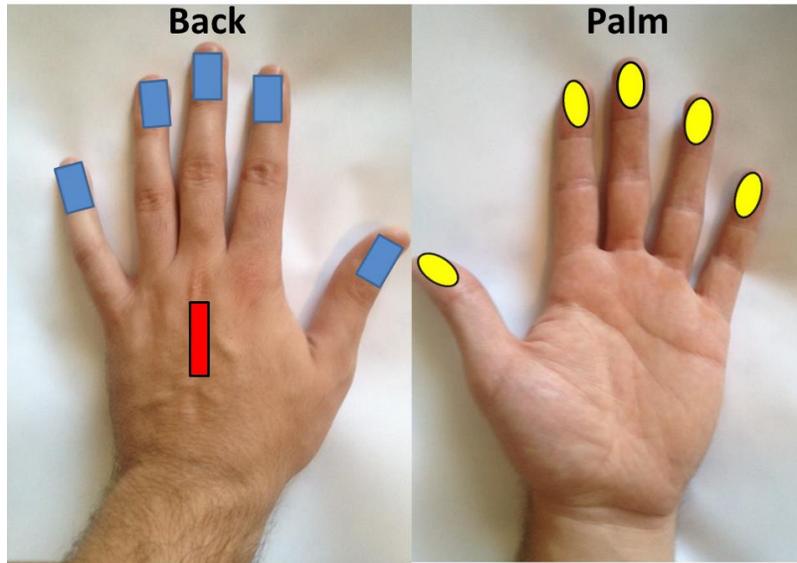


Fig. 3.1 IMU sensors were placed on the back of the hand mannequin (YCCTEAM Flexible Nail Trainer), pressure sensors on the palm side of the hand and the microcontroller and multiplexer were placed on the wrist and were not included in the hand image. The thinner rectangle on the center of the back of the hand is lined up with the middle finger and is the IMU with magnetometer, gyroscope and accelerometer sensors all integrated onto one chip.

previous research groups have identified forces in this application ranging up to 100 N [8, 22, 24]. IMU sensors provide very sensitive results and their ranges can be altered by accessing different registers. Further analysis of sensors and calibration procedures are discussed in more detail in Chapter 4. The IMU sensors use I²C communications and required the use of a multiplexer (Texas Instruments CD74HCT4067) to communicate simultaneously to the microcontroller, as several IMU sensors contained the same address. A multiplexer selects a specific digital or analog signal input and forwards this to a single output line. The complete electrical schematic is shown in Chapter 4, section I.

Table 3.1: Sensor Specifications Used in Design

| | | Range | Sensitivity | Resolution |
|-------------|--------------------------|---------------|-------------|------------|
| | ESS301-Pressure Sensor | 0-133 N | .1596 N | 12bit |
| Sensorstick | ADXL 345 - Accelerometer | ± 2 g | .0039 g | 10bit |
| | ITG-3200 - Gyroscope | ± 500 °/s | .0153 °/s | 16bit |
| | HMC5883L - Magnetometer | ± 1.3 G | .6348 mG | 12bit |
| MPU 6050 | MPU6050 - Accelerometer | ± 2 g | 6.104e-5 g | 16bit |
| | MPU6050 - Gyroscope | ± 500 °/s | .0153 °/s | 16bit |

Table 3.1 All pressure sensors are analog while IMU sensors used are digital. Each IMU contains 3-axes with the same range and sensitivity.

In order to communicate to the five *MPU6050* integrated boards, a serial data line (labeled *SDA*) and a serial clock line (labeled *SCL*) were used for I²C communication. The *SDA* line is connected to the multiplexer signal input. The *SCL* lines of every IMU sensor are tied together. The multiplexer was programmed to shift between various channels of inputs to communicate to each *MPU6050* sensor individually. This means there is a very slight delay in the reading between each IMU located on the fingers but is assumed to be negligible in this application (16 msec). A separate I²C line labeled *SDA1* and *SCL1* were used to connect to the last IMU which was placed on the back of the hand.

Pressure sensors (Tekscan ESS301) were connected to the board through analog pins. These sensors work as simple variable resistors and are connected in a voltage divider circuit. Decoupling capacitors valued 0.1 μ F and 10 μ F were connected in parallel between the *Vcc* and *GND* lines for each of the IMU sensors to remove AC noise in the circuit. Pull up resistors were integrated on each IMU connected to the

device. Each IMU and pressure sensor was calibrated before attaching the device to the hand mannequin (Ch.4. G.1.f.).

B.2 Sensor and Data Analysis

B.2.a) Sensor Fusion Algorithms

In order to obtain orientation information, a sensor fusion algorithm must be implemented on accelerometer, gyroscope, and magnetometer data [27, 28, 45, 46]. For each of the motion sensors on the developed glove the following data fusion techniques were used. A vector based complementary filter was developed to fuse sensor data and was aided by a DCM-based complementary filter [27, 28, 45, 46]. A combination of vector and DCM-based complementary filters described in this paper use methodology by previously developed complementary filters but also combines aspects from several data fusion algorithms [27, 28, 45, 46]. These filters are designed to estimate orientation of a rigid body using accelerometer data, and using other sensors such as gyroscopes or magnetometers to “update” the initial estimation [45, 46].

A first step was to remove biases from each specific sensor found during calibration procedures (Ch.4. G.1.f.), and the accelerometer and magnetometer were run through a first order low-pass filter with a cutoff frequency of half of the sampling rate (10Hz). Accelerometer and magnetometer vectors were then normalized. The first step of the complementary filter was to use the current time point (labeled as i) accelerometer vector to obtain a gravitation force vector labeled \vec{G}_a^i which is a unit vector because of accelerometer data normalization. Gravity vector is the output of the

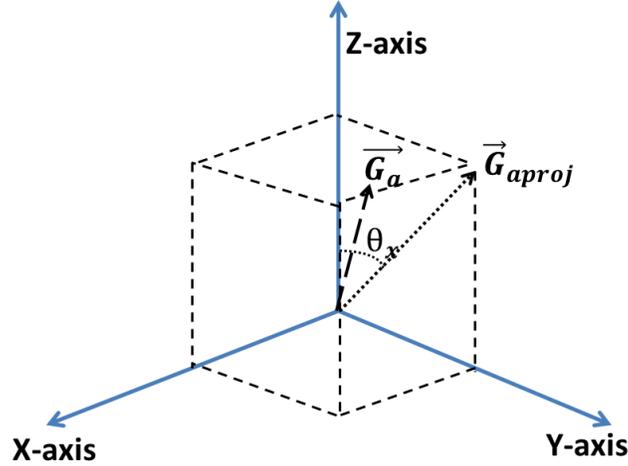


Fig. 3.2 A projection of the gravity vector on the yz -plane is shown in the figure (\vec{G}_{aproj}) and the angle around the x -axis is shown (θ_x). One can see that the way to find the value of the angle around the x -axis is determined through the use of the inverse tangent function.

accelerometer and will be in terms of g forces. $\overrightarrow{d\theta}_g^i = [d\theta_{gx}^i, d\theta_{gy}^i, d\theta_{gz}^i]$ is then calculated by simply multiplying the time instant $dt = 0.1sec$ by the current gyroscope output $\overrightarrow{\omega}_g^i$, $\overrightarrow{d\theta}_g^i = dt \overrightarrow{\omega}_g^i$. $\overrightarrow{d\theta}_g$ is a vector of the angle of rotation around each axis found solely from gyroscope data in %/s. Using the inverse tangent function, the temporary angles around x (θ_x) and y axes (θ_y) can be found using the gravitational force vector observed by the accelerometer as a right triangle is formed by projecting \vec{G}_a onto the yz -plane and xz -plane respectively (Fig.3.2, Eq. 3.1). The next angle of rotation vector obtained from gyroscope is used to find the new angle that the object has progressed by adding the angle change calculated by the gyroscope $\overrightarrow{d\theta}_g$ to the previously calculated temporary angles θ_x and θ_y using:

(3.1)

$$\theta_x^i = atan2(G_{ay}^{i-1}, G_{az}^{i-1}) + d\theta_{gx}^i$$

$$\theta_y^i = \text{atan2}(G_{ax}^{i-1}, G_{az}^{i-1}) + d\theta_{gy}^i$$

where $\vec{G}_a^{i-1} = [G_{ax}^{i-1}, G_{ay}^{i-1}, G_{az}^{i-1}]$. An updated gravitational force vector is calculated using the angles updated from gyroscope data and knowing that gravitational force vector magnitude is still 1. $\vec{G}_g = [G_{gx}, G_{gy}, G_{gz}]$ is found by knowing

$\sqrt{G_{gx}^2 + G_{gy}^2 + G_{gz}^2} = 1$ and following steps 1-3 you can find the equations for all components of \vec{G}_g .

(Step 1) – Divide by 1

$$\frac{G_{gx}}{1} = G_{gx} = \frac{G_{gx}}{\sqrt{G_{gx}^2 + G_{gy}^2 + G_{gz}^2}}$$

(Step 2) – Divide top and bottom by $\sqrt{G_{gx}^2 + G_{gz}^2}$ and numerator is equal to $\sin \theta_y$ (Fig.

3.2)

$$\frac{\frac{G_{gx}}{\sqrt{G_{gx}^2 + G_{gz}^2}}}{\sqrt{G_{gx}^2 + G_{gy}^2 + G_{gz}^2}} = \frac{\sin \theta_y}{\sqrt{1 + G_{gy}^2 / (G_{gx}^2 + G_{gz}^2)}}$$

(Step 3) – Multiply G_{gz}^2 to numerator and denominator under root symbol to obtain G_{gx} .

$$= \frac{\sin \theta_y}{\sqrt{1 + (G_{gy}^2 * G_{gz}^2) / ((G_{gx}^2 + G_{gz}^2) * G_{gz}^2)}} = G_{gx}^i = \frac{\sin \theta_y^i}{\sqrt{1 + (\cos \theta_y^i)^2 * (\tan \theta_x^i)^2}}$$

A similar procedure can be used to find G_{gy} and G_{gz} which simplify to:

$$G_{gy}^i = \frac{\sin \theta_x^i}{\sqrt{1 + (\cos \theta_x^i)^2 * (\tan \theta_y^i)^2}}$$

$$G_{gz}^i = s * \sqrt{1 - (G_{gx}^i)^2 - (G_{gy}^i)^2}$$

Where s is determined by measuring the value of G_{az} . If G_{az} is zero or greater than zero, then s is 1 and if G_{az} is negative s is -1. The next step is to use a weighted average between \vec{G}_a and \vec{G}_g to calculate a final gravity vector \vec{G}_F :

(3.2)

$$\vec{G}_F^i = \frac{\vec{G}_g^i * w_g^i + \vec{G}_a^i}{(1 + w_g^i)}$$

where w_g is weight of the gyroscope (Eq. 3.2). Weighting of \vec{G}_g is determined by calculating a performance metric, prior to normalization, which is the difference in magnitude between consecutive accelerometer vectors \vec{a} :

(3.3)

$$\Delta_a^i = abs \left[\|\vec{a}^i\| - \|\vec{a}^{i-1}\| \right]$$

(Eq. 3.3) [45, 46]. If the performance metric Δ_a is greater than a chosen constant then the weight of gyroscope is increased from 10 to 20. Weights of 5-20 for gyroscope produce acceptable results and the constant can be chosen to any range depending on the application; we chose a constant value of .025 g.

From \vec{G}_F we can calculate final Euler angles around the x -axis (φ) and y -axis (α) using:

(3.4)

$$\varphi^i = -atan2\left(\sqrt{G_{Fy}^i{}^2 + G_{Fz}^i{}^2}, G_{Fx}^i{}^2\right)$$

$$\alpha^i = -atan2\left(\sqrt{G_{Fx}^i{}^2 + G_{Fz}^i{}^2}, G_{Fy}^i{}^2\right)$$

Next we used the magnetometer to calculate rotations around the *z-axis* as a magnetometer contains information necessary to determine *yaw* rotation [48]. Rotation around the *z-axis* or *yaw* (ψ) can be calculated by finding the tangent of $\frac{m_y}{m_x}$ from magnetometer vector $\vec{m} = [m_x, m_y, m_z]$ [48]. Magnetometer values are reported in terms of Gauss and measure the Earth's magnetic field. If the IMU experiences tilting outside of the original *xy-plane* which is detected by a large change in m_z (>.07G) or a change in roll (φ) or pitch (α) value, then a different *yaw* calculation must be used. If tilting is detected, then the data read from the magnetometer must be transposed onto the *xy-plane* using the following:

(3.5)

$$C_{Mx} = (m_x * \cos \alpha) + (m_y * \sin \varphi * \sin \alpha) + (m_z * \cos \varphi * \sin \alpha)$$

$$C_{My} = (m_y * \cos \varphi) - (m_z * \sin \varphi)$$

$$\psi = atan\left(\frac{C_{My}}{C_{Mx}}\right)$$

where C_{Mx} and C_{My} are tilt-compensated magnetometer x and y values (Eq. 3.5) [48].

Euler angles are calculated at every time step and when these are at 75°-105° the vector-based complimentary filter provides inaccurate results because Gimbal lock is a well documented phenomena which occurs when an Euler angle approaches 90°

and a degree of freedom is lost [27, 28]. A direction cosine matrix (DCM) complementary filter is used to calculate orientation when Euler angles approach this inaccurate range. A DCM is a matrix used to convert a vector in the body frame of the sensor or rigid body to the global reference frame [46]. DCM's store the angles from the initial position to the current position in the x, y , and z coordinate planes. This matrix is updated at each time step using output from the accelerometer, gyroscope, and magnetometer. Vectors from each sensor must be normalized to become unit vectors, which is a requirement for a valid DCM to be formed [47].

Assuming the initial positions of a gravity vector and the magnetic north directions of the IMU are initially orthogonal, the DCM can be formed by using accelerometer and magnetometer outputs [46, 47]. The gravity vector is lined up with the z -axis in the negative direction at the initial time point ($i = 0$) and is labeled $\vec{I}^0 = [I_x^0, I_y^0, I_z^0]^T$ which is equal to the negative values of the acceleration vector \vec{a} [45, 47]. The Earth's magnetic north vector is labeled $\vec{J}^0 = [J_x^0, J_y^0, J_z^0]^T$ and is assumed to be orthogonal to the gravity vector and lie in the x -axis. The cross product of these vectors provides us with an orthogonal vector labeled $\vec{K}^0 = [K_x^0, K_y^0, K_z^0]^T$ that lies in the y -axis. The DCM is constructed by combining \vec{I}, \vec{J} , and \vec{K} into a 3X3 matrix,

$$\mathbf{DCM} = [\vec{I}, \vec{J}, \vec{K}].$$

The DCM is then updated at every time point by calculating a $\vec{d\theta}_{DCM}^i$ which is a weighted average of: progressed angle calculated using the accelerometer $\vec{d\theta}_a^i$, angle using the gyroscope $\vec{d\theta}_g^i$, and another using the magnetometer $\vec{d\theta}_m^i$ (Eq. 3.6) [46, 47]. The angular displacements calculated from the accelerometer and magnetometer, $\vec{d\theta}_a^i$

and $\overrightarrow{d\theta}_m^i$ are found by using the cross product of the difference between the present time point and previous time point of their corresponding sensor values and past time point of their corresponding sensor values:

$$\begin{aligned}\overrightarrow{d\theta}_a^i &= \vec{I}^{i-1} \times (\vec{I}^i - \vec{I}^{i-1}) \\ \overrightarrow{d\theta}_m^i &= \vec{J}^{i-1} \times (\vec{J}^i - \vec{J}^{i-1})\end{aligned}\tag{3.6}$$

[46, 47]. $\overrightarrow{d\theta}_g^i$ is calculated as it was in the vector-based complementary filter. A weighted average of the $\overrightarrow{d\theta}_{DCM}^i$ is calculated using each of the three separate values with the aid of the gyroscope weighting described before in (Eq. 3.3) and another performance metric Δ_m to adjust magnetometer weighting:

$$\Delta_m^i = abs \left[\|\overrightarrow{m}^i\| - \|\overrightarrow{m}^{i-1}\| \right]\tag{3.7}$$

where \overrightarrow{m}^i is the magnetic vector at time point i (Eq. 3.7). The performance metric Δ_m quantifies external magnetic interference and when this performance metric passes a chosen constant, in our case .03 G, the weighting of the magnetometer is decreased by half from 10 to 5 [27, 28]. Performance metrics chosen in this paper are used in Sabatini's developed methods to adapt a noise covariance matrix in his Kalman filter [27, 28]. As the Earth's magnetic field should have the same magnitude from time point $i - 1$ to i , a large increase or decrease in magnetic field magnitude indicates an external magnetic interference [27, 28].

(3.8)

$$\vec{d\theta}_{DCM}^i = \frac{(\vec{d\theta}_a^i * w_a^i) + (\vec{d\theta}_m^i * w_m^i) + (\vec{d\theta}_g^i * w_g^i)}{w_a^i + w_m^i + w_g^i}$$

Where w_a is the weight of the accelerometer data and is chosen to be 10. When a weight-adjusted $\vec{d\theta}_{DCM}^i$ has been calculated, each vector in the DCM is updated by adding the cross product of the current angle progressed and current vector to the current DCM vectors (\vec{I}^i, \vec{J}^i):

(3.9)

$$\vec{I}^{i+1} = \vec{I}^i + (\vec{d\theta}_{DCM}^i \times \vec{I}^i)$$

$$\vec{J}^{i+1} = \vec{J}^i + (\vec{d\theta}_{DCM}^i \times \vec{J}^i)$$

$$\vec{K}^{i+1} = \vec{I}^{i+1} \times \vec{J}^{i+1}$$

[46, 47]. The matrix is updated at each small time point dt which is the time between sensor data (0.1sec). The DCM provides angles that the IMU has progressed from the initial position of the device which can be converted into Euler angles. For the DCM to remain valid the vectors must remain orthonormal and have a magnitude of 1 so each vector is normalized at every time step [47]. In order to maintain the orthonormal requirement, a correction metric Err is subtracted from the vectors \vec{I}^{i+1} and \vec{J}^{i+1} , $Err = \frac{(\vec{I}^{i+1} \cdot \vec{J}^{i+1})}{2}$ [46, 47]. This error metric is found by using the triple cross product between \vec{I}^{i+1} and \vec{J}^{i+1} to ensure they are both orthogonal to one another. The DCM is used to compute *roll* (ϕ), *pitch* (α), and *yaw* (ψ) when these Euler angles approach 90° (75° - 105°) as the accuracy of the vector-based algorithm decreases at these values.

A flowchart describing the filtering algorithm is displayed in Fig. 3.3. A new \vec{G}_F vector can be found from the computed Euler angles by assuming an initial $\vec{G}_F^0 = [0,0,1]$

and using angles and rotation matrices to rotate the initial vector into place. One of the six motion sensors on the developed device contains a magnetometer. The *yaw* value for every finger was calculated off of the magnetometer data on the back of the hand. An offset for each of the fingers was discovered by performing a known *yaw* rotation of 90° . This calculation of *yaw* value for separate fingers may cause errors in hand gestures such as finger spreading, however, this assumption will be sufficient for the intended uses of the developed glove. This process is discussed in more detail in Chapter 4; offsets are displayed with respect to the hand *yaw* angle (Table 3.2). Data acquisition was performed with Arduino software and vector and DCM-based complementary filters were implemented in *MATLAB*. Euler angle calculations from device data were made using *MATLAB* software and Euler angle data was imported from the electromagnetic sensors using *MotionMonitor* motion capture software.

Table 3.2 Yaw Angle Offsets for Each Sensor Used in Algorithm

| Pinky | Ring | Middle | Index | Thumb |
|----------------|----------------|------------|-------------|-------------|
| -29.66° | -10.66° | -3° | -19° | -29° |

Table 3.2 These offsets displayed were found by calculating *yaw* angles during a *yaw* rotation around 90° . The difference from electromagnetic sensor *yaw* angle and *yaw* angle found by the hand was found. The offset was added to the *yaw* value found by the hand. ex) $\psi_{index} = \psi_{hand} + offset_{index}$

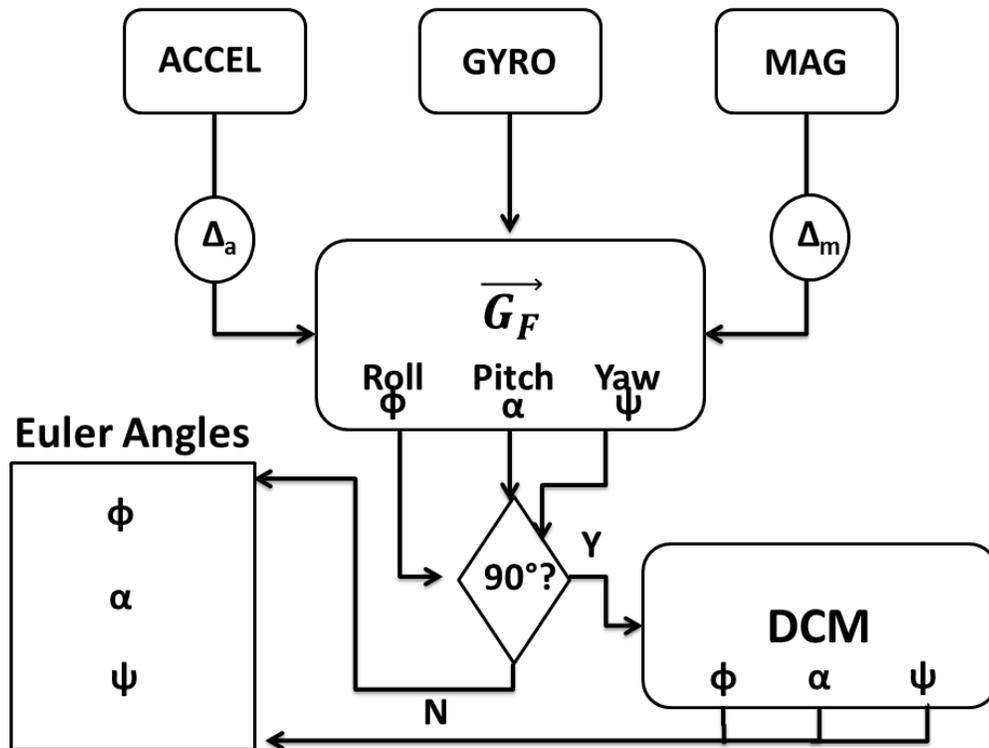


Fig. 3.3 The flowchart describes how the combined complementary filtering algorithm functions. Data is input from each of the sensors; accelerometer data and magnetometer data are compared to performance metrics Δ_a and Δ_m to alter weighting of the filters. A gravity vector is calculated and converted to Euler angles; if the Euler angles are near 75° - 105° then the DCM filter is used to calculate the corresponding Euler angle. The vector based algorithm provides inconsistencies around 90° due to Gimbal lock.

B.2.b) Force and Moment Calculations

Flexible pressure sensors were first calibrated by placing several known weights on the sensor and examining the output voltage change $V(t)$. Data were fit to a trend line for each sensor individually, thus a corresponding voltage reading provides a force output. Pressure sensor calibration is described in greater detail in Chapter 4 section G.1.f. In order to determine the force vector and moment vector, the final gravity vector

was used. The components of the updated gravity vector from the IMU calculations \vec{G}_F , provide coefficients to multiply the scalar force value obtained from each individual pressure sensor to obtain a force vector $f = \|F_x, F_y, F_z\|$

(3.10)

$$[F_x, F_y, F_z] = \left[\frac{G_{Fx}}{\|\vec{G}_F\|} * f, \quad \frac{G_{Fy}}{\|\vec{G}_F\|} * f, \quad \frac{G_{Fz}}{\|\vec{G}_F\|} * f \right]$$

[47]. This force calculation assumes that the only force exerted by the device is in the normal direction, which is not always true in practice [1, 5, 9, 11]. Force components are then summed in each direction for all five pressure sensors to produce a total force vector at each time point $\vec{F}_{tot}^i = [F_{xtot}^i, F_{ytot}^i, F_{ztot}^i]$. Moments were calculated by multiplying the moment arm length, 0.38815 m, of our experimental setup by the corresponding force component that created a bending moment. A force application in the z direction corresponds to a bending moment around the y-axis (M_{O_y}) and a force application in the y direction corresponds to a moment around the z-axis (M_{O_z}) (Fig. 3.4). In order to calculate moment arm length it was assumed that all force was applied in the center of the Styrofoam sphere.

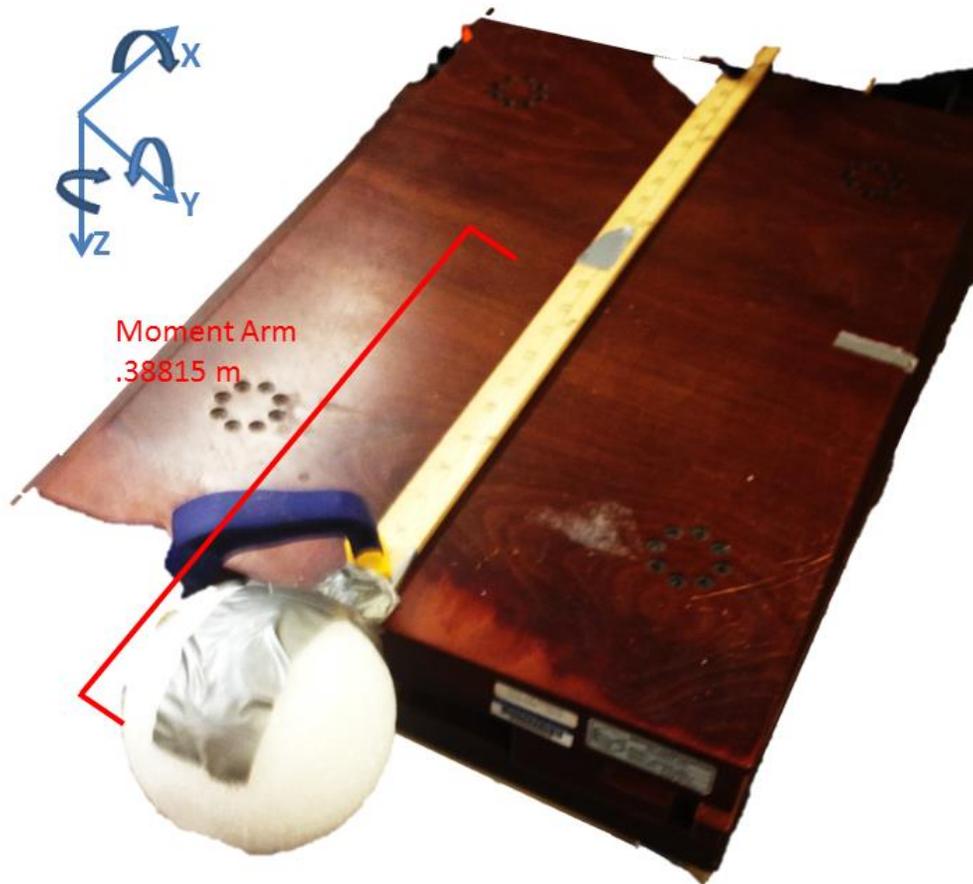


Fig. 3.4 The force testing apparatus is shown with the coordinate plane. Moments are shown in the coordinate plane as arrows around each axis. One can see that a twisting of the Styrofoam would be how a moment around the x axis is induced. A force application in the z direction corresponds to a bending moment around the y -axis (M_{O_y}) and a force application in the y direction corresponds to a moment around the z -axis (M_{O_z})

In the experiment, scale factors were multiplied by the force vector components which were necessary to accommodate for forces applied by the mannequin hand outside of the fingertips or shear forces that may exist (Table 3.3). Scale factors were determined by performing a normal force application in the x, y, and z direction and comparing this result to the force plate reading. A scale factor for each direction was found by dividing the force plate component reading by the component being tested of the glove. This process is described in more detail in Chapter 4, section H.

Table 3.3 Force Scale Factors for Each Direction

| X-Direction | Y-Direction | Z-Direction |
|-------------|-------------|-------------|
| 1.877 | 2.230 | 1.892 |

Table 3.3 The scaling factors are multiplied by the force components calculated in each direction to accommodate for forces experienced outside of the pressure sensing areas.

B.3. Experimental Validation

B.3.a) Motion Validation

This section is to validate the calculations of motion using an experimental procedure utilizing an additional motion tracking system. In order for force vectors to be accurately calculated, the orientation of the device must first be correctly determined. The developed device was placed on the hand mannequin with a set of electromagnetic sensors placed atop of the existing IMU sensors. An IMU and electromagnetic sensor (Ascension trakSTAR) were placed on the back of each fingertip as well as the center of the back of the palm of the hand mannequin. The electromagnetic sensors have an

accuracy of Euler angle under 0.5° [49]. An image displaying the hand mannequin with both electromagnetic and IMU sensors placed is shown in Fig. 3.5. The electromagnetic sensors compared to are accurate within 0.5° for calculated Euler angles [49]. The electromagnetic emitter was placed on a plastic shelving unit, the hand mannequin and bendable arm were clamped to a table. The hand mannequin was within the transmitter's range (0.1-0.9144m) while the bendable arm was outside of range. No ferromagnetic materials were placed within range to ensure that the electromagnetic sensing unit was not interfered with [49].

Each test began with the hand mannequin at rest in a prone position in the *xy-plane* for 10 sec and the hand was rotated to a known orientation (~ 5 sec) and remained in this orientation to total a 35 sec test. Fourteen known orientations were each tested three times for a total of 54 tests. Each orientation was chosen as to simulate natural hand movements covering a range of $0-180^\circ$ in *roll* angle (hand supination/pronation), $0-90^\circ$ in the *pitch* direction (wrist flexion/extension), and $0-90^\circ$ in the *yaw* direction (radial/ulnar deviation) (Fig.6). In order for IMUs and electromagnetic Euler angles to be

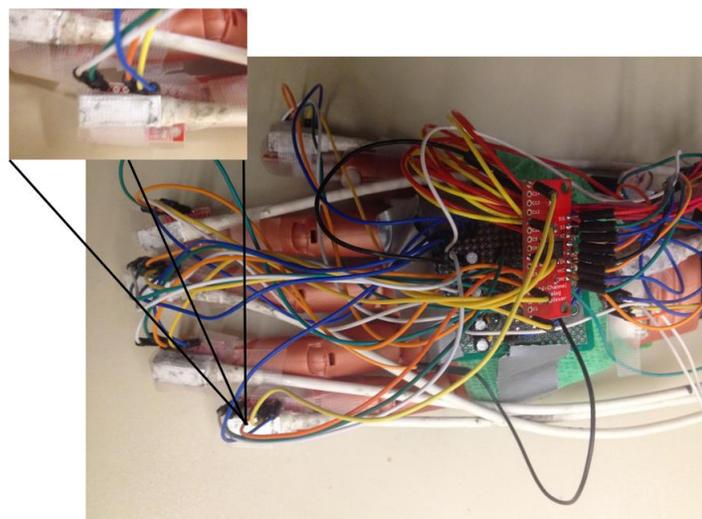


Fig. 3.5 The sensors placed on the back of the hand mannequin along with wiring is shown above. A magnified image of the pinky sensor is shown to display the electromagnetic sensor on top of the IMU sensor.

compared, an offset was determined by three test runs, one around each rotation angle (Ch. 4.1.2.c). This offset was then applied to IMU calculated angles to account for the difference in placement of the electromagnetic and IMU sensors on the hand mannequin that will lead to slight differences in Euler angle rotations solely from placement of sensors (Table 3.4). This process is discussed in more depth in Chapter 4, section 1.2.c.

Table 3.4 Euler Angle Offset for Each Sensor Used in Algorithm

| Euler Angle | Pinky | Ring | Middle | Index | Thumb | Hand |
|--------------------|---------|---------|--------|---------|---------|--------|
| Roll (ϕ) | -5.33° | 23.33° | 16° | 11° | -28.33° | 30° |
| Pitch (α) | 5° | 0° | -7.33° | -2.33° | -25° | 27.66° |
| Yaw (ψ) | -29.66° | -10.66° | -3° | -19.33° | -29.33° | -5.33° |

Table 3.4 These offsets were added to the calculated Euler angle to account for difference in placement between electromagnetic sensors and glove IMU sensors. These offsets are used to enable electromagnetic sensor data and glove sensor data to be compared even with a difference in placement of each sensor.

The glove outputs data at 10 Hz and the electromagnetic sensors output data at 100 Hz. In order to compare the same amount of data points, every tenth point of electromagnetic sensor data was used for analysis. Data was split into three separate time phases: first was the hand at rest (beginning rest), next was the transient movement of the hand, and lastly was the hand at rest at the new orientation (secondary rest). Beginning rest phase consisted of the time between 1-9 seconds, the transient phase was during the 12-20 second time slot, and the secondary rest phase was between the 24-32 second time slot.

Each of the three time phases were analyzed using the same methods. Root-mean-square error (RMSE) and average error (AVGE) were calculated over each time phase for every Euler angle from every IMU sensor. Linear regression tests were also used to compare glove calculated Euler angles and Euler angles obtained from electromagnetic sensors. The fourteen hand orientations tested are shown in Fig. 3.6.

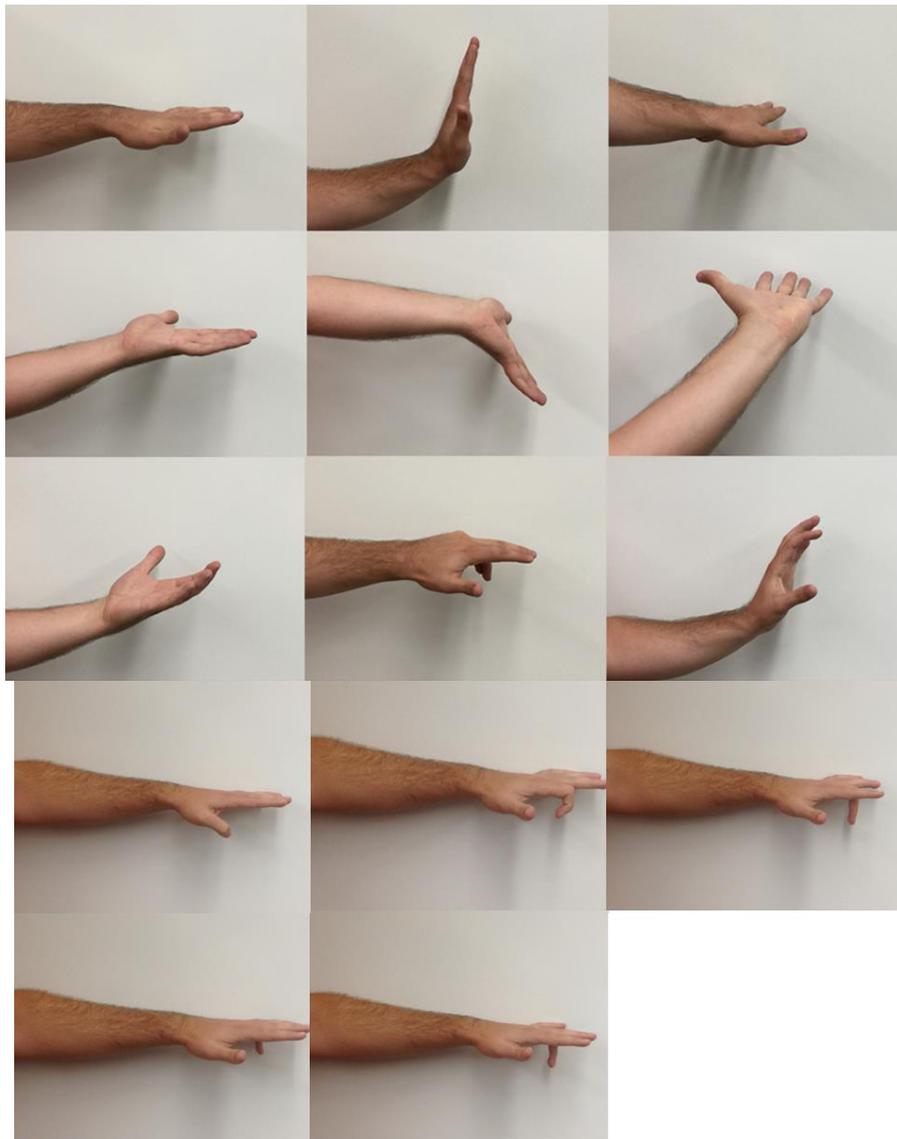


Fig. 3.6 The fourteen different orientations above were the orientations tested in motion validation experiments. The hand mannequin was maneuvered into these positions after the first 10 seconds of resting. Orientations were picked to represent Euler angle rotations in each direction as well as a few complex orientations.

B.3.b) Force Validation

After validation of the motion capturing capabilities, validation of force calculations were performed. A force plate (*Bertec 4060-NC*) with a range of 0-2,500 N and resolution of ± 0.5 N was placed on a table with a long rectangular straightedge clamped to it. The straightedge extends 0.127m off of the force plate and on this end a 0.0635 m-radius Styrofoam spherical object was attached. *Motion Monitor* software was used in accordance with the force plate to calculate forces and moments in each of three dimensions with respect to the center of the plate. However, moments in the x direction (M_{O_x}) were not used in analysis. Each test began with the hand mannequin in a prone position in the xy -plane resting for 10 sec. The hand was then rotated to a stable position and a force was applied by the hand mannequin to the spherical object in a normal direction to the palm of the hand for 20 sec. The middle 10 seconds of the force application were used in RMSE and AVGE calculations when comparing glove data to force plate data. Nine separate orientations were tested and each orientation

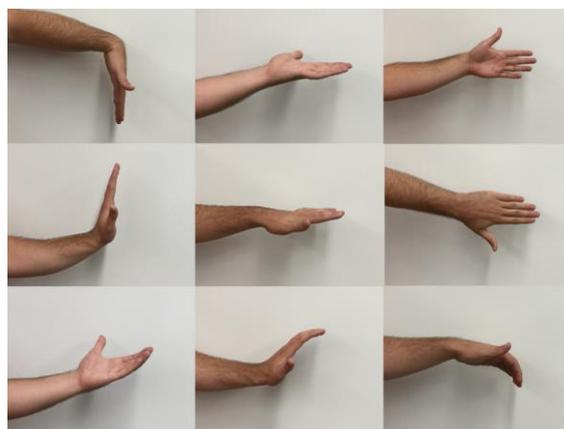


Fig. 3.7 Nine different orientations of the hand were tested by applying a normal force to the direction of the palm. The orientations were chosen to represent positive and negative directions in each of the x , y , and z axes. Three more complex orientations were tested to increase the difficulty of the force determination.

was tested five times (Fig. 3.7). Seven tests were removed due to a sensor error leaving a total of 38 tests; these tests were removed if the sum of all force magnitudes was less than 1 N as forces were not being recorded by the pressure sensors. Force plate data was obtained at 50 Hz so every fifth time point was used in calculations to compare to glove data.

Force plate data provided five quantities: F_x , F_y , F_z , Mo_y , and Mo_z . Glove data also included five directions, neglecting to include Mo_x . A moment around the *x-axis* in the experimental setup was difficult to quantify using the data obtained from the glove as only normal forces are used. A twisting force would have to be quantified in order to obtain Mo_x , which cannot be calculated in the currently developed algorithm. Force results were compared in the two directions of the greatest magnitude by the glove during the 20 sec of force application. The third direction of force application from the glove contained small forces and were unreliable when compared to that of the force plate data. The developed device was designed to measure normal forces applied by it and is incapable of capturing shear forces. Moment of the greatest magnitude from the glove device was compared to the corresponding moment component of the force plate.

The pressure sensors alone underrepresent the total force applied by the mannequin hand because some forces are applied outside of sensor areas on the hand. To correct this, a scale factor was determined in each direction and was applied to the glove pressure data. The scale factors were calculated by performing a normal force application in three directions and comparing these values to that of the force plate data (Table 3.3). This is described in more detail in chapter 4. A scale factor in each direction was multiplied by the resulting force components calculated by the developed glove to

accommodate for forces applied outside of the fingertips [1]. The glove force component with the largest (F_1) and second largest (F_2) magnitudes were compared with force plate data. The moment component with the greatest magnitude (Mo_1) was compared to the corresponding force plate moment component data. Data was compared by calculating RMSE and AVGE between glove data and force plate data over the middle 10 seconds of force application. Average force values over the middle 10 seconds of force application for the three components described (F_1 , F_2 , Mo_1) calculated from glove data and force plate data were compared using a linear regression test.

C. Results

C.1 Motion Validation

Each test was composed of 18 different Euler angle outputs as a function of time; *roll*, *pitch*, and *yaw* for each of the six sensors on the hand mannequin. A root-mean-square error (RMSE) and average error (AVGE) were calculated for each of these 18 Euler angles at the beginning time phase (beginning rest), a transient time phase, and a secondary resting phase (secondary rest) when the hand mannequin was at its final orientation. This calculation was performed for all of the data ($n = 3$). The AVGE was calculated by subtracting the calculated angle of the device from every tenth electromagnetic recorded sensor angles (True Angle) for each time phase of every test. Each time phase was 8 sec in length, the RMSE and AVGE were calculated over the course of each time phase: beginning rest phase, transient phase, and secondary rest phase.

Table 3.5 reports the statistics of both parameters compared among *roll*, *pitch*, and *yaw* values; the average and standard deviation are shown for each angle. Using two separate one-way ANOVA tests comparing RMSE values across Euler angles tested resulted in several significant findings. RMSE values of *roll* and *pitch*, as well as *pitch* and *yaw* significantly differed, both comparisons were found to have *p-values* of $<< .0001$. AVGE significantly differed between all Euler angles ($p << .0001$). All *p-values* calculated used Bonferroni correction to account for multiple comparisons.

Table 3.5 Error Comparison for Euler Angles Tested (°)

| | Roll (ϕ) | Pitch (α) | Yaw (ψ) |
|------|-----------------------------|---------------------------|-------------------------------|
| RMSE | $8.91 \pm 15.24^*$ | $6.37 \pm 10.96^{*\sim}$ | $8.58 \pm 17.07^{\sim}$ |
| AVGE | $-3.87 \pm 13.48^{*\wedge}$ | $-0.48 \pm 11.03^{*\sim}$ | $3.91 \pm 16.84^{\wedge\sim}$ |

$^{*\wedge} p < .05$

Table 3.5 Two one-way ANOVA tests were performed to compare AVGE and RMSE between each Euler angle. Errors did significantly differ among the Euler angles, meaning there is a difference of error between the three angles tested. Pitch differed in RMSE values between both Euler angles and AVGE values significantly differed among all Euler angles. AVGE and RMSE values were over all time phases and all orientations tested.

Roll contained the largest of the RMSE values at 8.91° while *yaw* and *pitch* were at 8.58° and 6.37° respectively. RMSE and AVGE were compared between rest and transient time phases, the rest time phases were characterized as the beginning time phase before rotation as well as the ending pose or secondary rest phase of the hand mannequin time phase.

Table 3.6 Error Comparison Among Different Time Phases (°)

| | Rest | Transient |
|-------|--------------|---------------|
| RMSE* | 5.40 ± 12.84 | 13.06 ± 16.67 |
| AVGE | 0.23 ± 13.79 | -0.87 ± 15.37 |

* $P < .05$

Table 3.6 Two t-test of means was performed comparing AVGE and RMSE values between resting and transient time phases. The errors shown include all Euler angles and all orientations tested. RMSE differed between resting and transient time phases.

Table 3.6 presents the AVGE and RMSE results comparing resting and transient time phases. Two separate two- sample *t-test* of means was performed to compare the AVGE and RMSE distributions between rest and transient time phases across all performed tests and a *p-value* of .08 was found comparing AVGE and a *p-value* of $<< .0001$ was found comparing RMSE ($n_1=252$, $n_2=504$). This indicates RMSE significantly differed between the time phases and AVGE did not. This difference is evident in Table 3.6 which shows that the RMSE and AVGE of rest values are 5.40° and 0.23° respectively while the RMSE and AVGE of transient values are 13.06° and -0.87° respectively.

Error metrics were compared among sensor location. Hand, thumb, index, middle, ring, and pinky RMSE and AVGE average values over all tests are shown in Table 3.7. The largest RMSE value is found from the hand sensor. The pinky AVGE has the greatest magnitude. The smallest RMSE and AVGE values belong to the middle sensor Euler angle values. Two separate one-way ANOVA tests were performed to compare the sensor RMSE and AVGE values between sensor locations. RMSE values did not significantly differ among the different sensor results. AVGE values did significantly differ among sensors.

Table 3.7 Error Metrics Among Sensors (°)

| | RMSE (°) | AVGE (°) |
|---------------|--------------|--------------|
| Hand | 6.69 ± 12.57 | 1.90 ± 11.29 |
| Pinky | 5.61 ± 10.30 | -1.98 ± 9.49 |
| Ring | 6.09 ± 9.98 | -0.62 ± 9.65 |
| Middle | 5.33 ± 8.93 | -0.16 ± 7.86 |
| Index | 5.96 ± 10.31 | 1.16 ± 9.51 |
| Thumb | 5.83 ± 10.41 | 0.61 ± 9.66 |

Table 3.7 RMSE and AVGE values for each sensor are shown. All values are in degrees (°).

Table 3.8 AVGE One-way ANOVA Test Results

| | Hand | Pinky | Ring | Middle | Index | Thumb |
|---------------|------|-------|------|--------|-------|-------|
| Hand | | < | | | | |
| Pinky | | | | | > | > |
| Ring | | | | | | |
| Middle | | | | | | |
| Index | | | | | | |
| Thumb | | | | | | |

Table 3.8 *p-values* below .0033 are highlighted by an > or < comparing AVGE values from different sensor locations. The sensor locations on the vertical axis are those being compared to: the hand has a significantly smaller AVGE value than the pinky. This means the pinky overestimates Euler angle with respect to the electromagnetic sensor system.

Table 3.8 displays the areas of significant p -values ($p < .0033$) for the one-way ANOVA test on AVGE values. Each p -value was again adjusted using Bonferroni correction. Hand sensor AVGE values differed from pinky sensor AVGE values. Pinky sensor AVGE values differed from thumb and index AVGE values as well. Pinky sensor Euler angles were overestimated when compared to hand, index, and thumb sensor angles.

A linear regression between all of the true Euler angles (electromagnetic sensors) and the calculated Euler angles (glove) was performed (Fig. 3.8). The angles were averaged over every time phase tested and for each Euler angle totaling 2268 angles tested. A p -value of $<< .0001$ was found indicating that the slope is not equal to 0 and a linear trend does exist. An R^2 value was calculated to be 0.831 indicating that

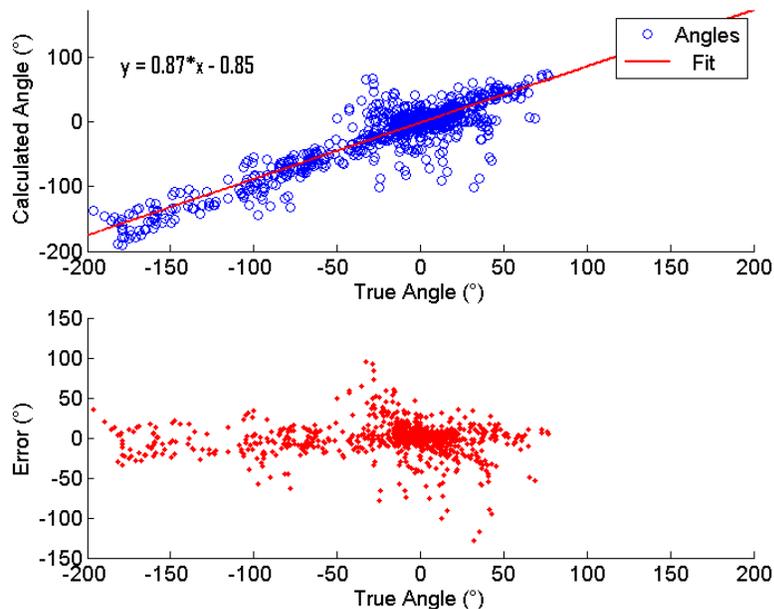


Fig. 3.8 Linear regression comparing true angle (°) and calculated angle (°). The equation of the line is in the top left corner and describes the linear relationship between both parameters. A good fit with an R^2 value of .831 shows that 83% of the variability in calculated angle is described by true angle. Residuals are plotted below to show that a pattern does not exist which would indicate a fit other than a linear one exists.

the linear regression model explains 83% of the variability of the calculated angle.

C.2. Force Validation

A total force vector was calculated by summing the force components determined for each fingertip. Only three components were compared among the force data output from the developed device and data output from the force plate. The components compared were the force component with the largest magnitude averaged over the middle 10 seconds of force application (primary force component F_1), the second largest magnitude (secondary force component F_2), and the moment with the largest magnitude M_1 , all of which were determined from the developed glove. Negative forces correlate to a negative direction.

A linear regression test and a t-test of means was used to determine whether the scalar force magnitude for each test calculated from the force plate and the glove were linearly related. The t-test comparing the force magnitude calculated from the force plate and force magnitude calculated from the glove resulted in a p value of .1165 meaning the two distributions were not proven to be significantly different. A linear regression test was performed between the scalar force obtained from the glove and scalar force from force plate for all force tests (Fig.3.9). An R^2 value of .652 was found and a slope of 0.68 was found. The linear fit is proven significant ($p < .0001$) but contains a small R^2 value.

Force vector components were then compared individually between the two systems. Not all of the components were tested, as the device's force capabilities were limited to recording forces normal to the palm position. The component of force with the

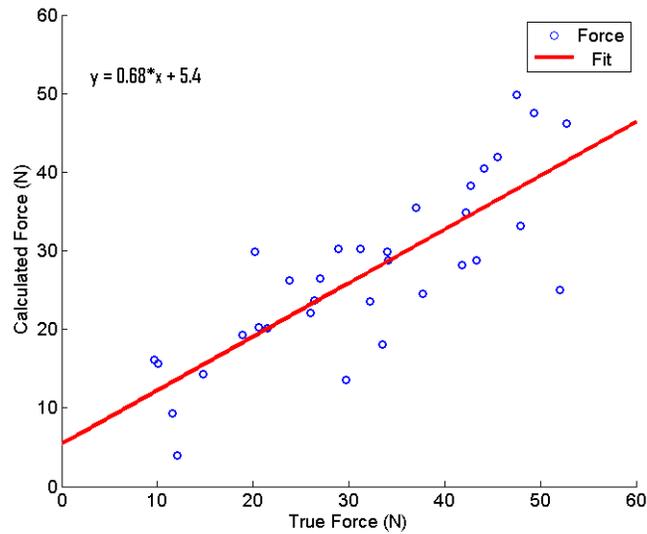


Fig. 3.9 Linear regression between the total force magnitude calculated from the force plate (true force) vs calculated force from the glove device. The linear fit has an R^2 of .652 and a slope of .68. The linear relationship is significant ($p << .0001$).

largest average magnitude from the developed glove was compared to that of its corresponding direction from the force plate data. The average RMSE percent error between the primary force component and corresponding force plate component was found to be 5.85%. This was found by dividing each RMSE by its corresponding force plate scalar force averaged over time for each test. Table 3.9 displays the statistical results for RMSE and AVGE of the primary force and moment components as well as the secondary force component. The results are represented as a percentage of the average force value for each corresponding test. The table displays percent errors. The average RMSE of the secondary force component and primary moment component were extremely large (>50%).

Table 3.9 Percent Error Results of Components Measured

| | F1 | F2 | M1 |
|------|--------|---------|---------|
| RMSE | 5.85% | 105.18% | 58.78% |
| AVGE | -4.20% | 52.03% | -28.72% |

Table 3.9 AVGE and RMSE are error metrics that were calculated for the primary force component, secondary force component and the primary moment component and divided by the force or moment force plate value.

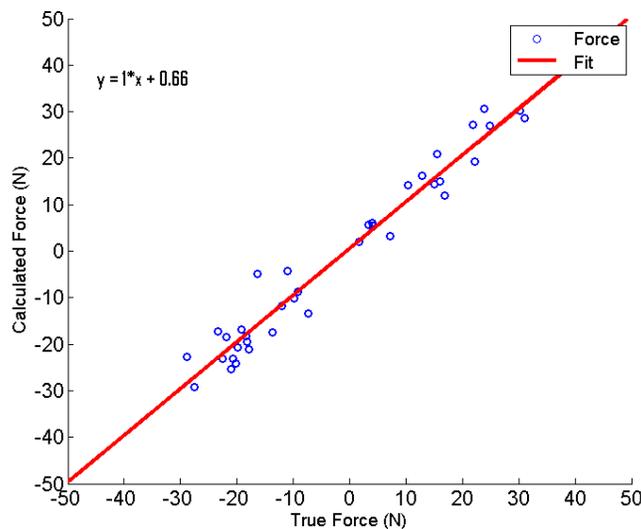


Fig. 3.10 The linear regression test shows the relationship true force (N) to calculated force (N) for primary force component F_1 . The test shows a coefficient of 1 and an R^2 value of .956 meaning that 95% of variability in calculated force is due to the true force. This shows a tight linear fit between the two variables.

A linear regression test comparing the primary force component and the corresponding force component from the force plate was performed. A p -value of $<<.0001$ was found indicating that the linear coefficient does not equal 0. The linear regression model is displayed in Fig. 3.10. The R^2 value of this fit is .956, thus the linear model explains 95% of the variability of the calculated force. The secondary force component was compared to its corresponding force plate data using a linear regression test. Fig. 3.11 displays the linear fit; a p -value of $3.333e-6$ was found

indicating that the linear coefficient between true force and calculated force is not equal to 0. The R^2 value is .456 which indicates the model explains only 45.6% of the variability of the calculated force. The primary moment component was compared to its corresponding force plate data set. A linear regression test was performed, a p -value \ll .0001 was found indicating a significant relationship between true moment and calculated moment (Fig.3.12). An R^2 value of .872 was calculated indicating the model explains 87.2% of the variability in the calculated moment.

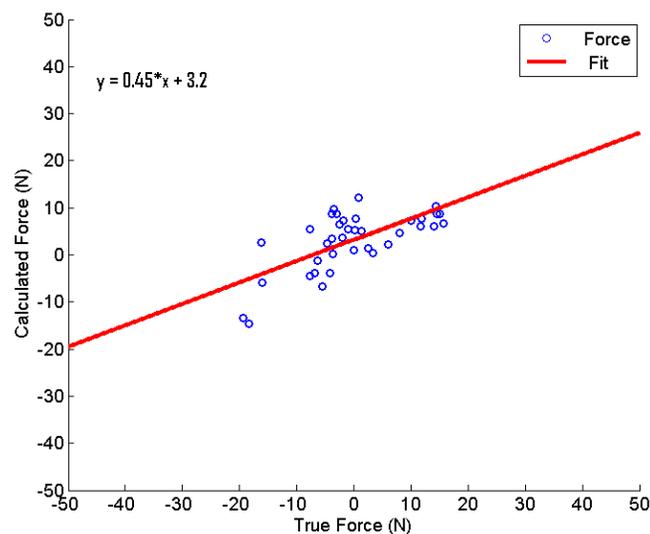


Fig. 3.11 The linear regression test shows the relationship true force (N) to calculated force (N) for secondary force component F_2 . The test shows a coefficient of .45 and a poor fit. This relationship does not contain a coefficient of 1 which is what the relationship should be. The R^2 value is .456 indicating the poor fit, which can be seen in the figure.

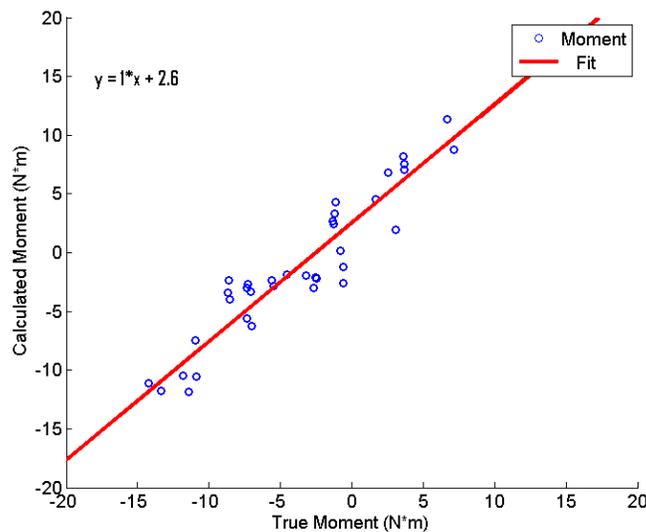


Fig. 3.12 The linear regression test shows the relationship true moment (N•M) to calculated moment (N•m) for primary moment component M_1 . The test shows a coefficient of 1 and an R^2 value of .872 indicating the true moment accounts for 87.2% of the variability in calculated moment. The linear relationship with a coefficient of 1 is shown to be significant and a relatively high R^2 value indicate a good fit, though not as strong as F_1 .

D. Discussion

The developed device proves to be capable of calculating force and moment vectors in three dimensions. The primary force component has an error of 5.85% of total force experienced and the primary moment error is 58.78%. By using the primary force and moment components, a force and moment can be accurately quantified though only one component of the force can be determined accurately at any time point. In the primary moment component, the errors are quite large but we found that a significant linear trend does exist between true moments and calculated moments. This indicates the glove is measuring moments inaccurately but in response to true moments experienced. Improvements in algorithm may improve the error involved.

Motion testing results and contain large RMSE and AVGE values $\sim 10^\circ$ in some cases. An R^2 value of 0.831 and a significantly small p -value indicates that the coefficient of the linear model is not 0 and a trend does indeed occur and explains 83% of variability in calculated angle. A linear trend is a trend that is an ideal trend for the calculated and true angle; however the coefficient of 0.87 is not ideal as a coefficient should be approximately 1.

A comparison between the rest and transient time phases shows that calculated angles in transient time phases have significantly larger AVGE values than angles calculated in rest time phases as shown by t -test results. This increase in inaccuracy could be due to the transient accelerations experienced by the accelerometer. The Δ_α performance metric is designed to accommodate for transient accelerations, however, if the chosen constant is not surpassed, a transient acceleration is experienced and this may induce error in the calculation [27, 28, 46]. Reducing the constant compared to the calculated performance metric makes the algorithm rely upon gyroscope and magnetometer data during smaller transient accelerations. This makes the algorithm more susceptible to problems that occur in gyroscopes and magnetometers that were discussed previously [45, 46, 48]. Gimbal lock issues may arise when using Euler angles and this can be seen when angles are near 90° and these may cause 'jumps' in data which can create significant RMSE and AVGE calculations [27, 28, 46]. Errors also arise because of the difference in placement of electromagnetic sensors and the IMU sensors. The differences of placement were accommodated by offsets calculated before testing but these may differ slightly when complex gestures were made.

Comparisons among Euler angles as well as sensor locations show several differences that exist in error results. Differences between the *pitch* angle and the other Euler angles exist. Smaller errors are found in the *pitch* angle and could be due to the fact that the *pitch* angle is tested over a smaller range of angles than *roll* and *yaw* angles. The reason this angle was tested over a smaller range of angles was because the range of motion of the wrist flexion/extension is smaller than hand around the wrist (hand supination/pronation). Differences exist in error between Euler angles obtained from hand and pinky sensors. This difference could simply be due to difference in placement of electromagnetic sensors and IMU sensors. Another possibility of difference is the fact that the hand uses a different sensor than sensors used on the fingers. A difference also exists between the thumb and pinky sensors as well as the index and pinky sensors. These differences could be due to the fact that these sets of fingers are farther away from each other than many of the others. This distance could provide a difference in ranges of angles experienced by each of the sensors. Though if distance between fingers was a large reason for difference in error characteristics we would expect to see large errors between ring and thumb sensors which we do not.

Force testing was accomplished using a rectangular straight edge with a Styrofoam sphere attached to the end of it. The force testing apparatus did not allow for a moment to be calculated in the x direction because a moment was not induced by a simple normal force. Seven tests had to be removed due to pressure sensor errors; forces from the developed glove were not recorded in these tests and thus were removed. In order to accommodate for forces applied by the hand mannequin not located on the fingertips, test runs were performed in each direction to determine scale

factors that calculated forces need to be multiplied by. The scale factors are essentially simulating a larger area of force application. Scale factors were all less than 2.

The force tests composed of every possible direction to ensure the developed device could capture force in any direction. Force and moment components were ordered by magnitude for each specific test orientation. The primary and secondary force components as well as the primary moment component were the only components statistically tested. Primary force and moment components show a lower RMSE and AVGE than that of the secondary force component. The primary components are more accurate because the complementary filter algorithm and weighting of the force vector make the normal vector component more heavily weighted than others. Results are accurate in every direction the hand is facing but only one force vector component is reliable at a time point. An accurate measurement from one component of force provides insight that other studies have neglected to include [1, 8, 11, 22].

A linear regression analysis was performed for the primary force component, secondary force component, and the primary moment component. The primary force component of the developed device was compared to the corresponding force plate component using a linear regression test. The resulting linear regression test shows that a linear trend exists between the calculated and true force and the trend accounts for 95% of the variability in calculated force. The calculated coefficient for the primary force component is 1 showing a direct relationship between calculated and true force. A linear regression comparing the secondary force component and its corresponding force plate data shows that a significant trend exists, that the coefficient is not equal to 0. This fit does have a very low R^2 indicating that this linear trend is not a good fit, it does not

account for a large portion of variability in the calculated force data. The moment component linear regression analysis does show that a significant trend does exist and the calculated coefficient is 1. An R^2 value of 0.872 is not sufficiently high but is larger than that of the secondary force linear regression test.

The most reliable force calculations are that of the largest force component in magnitude and that of the moment component with the largest magnitude. These are the most accurate measurements because the algorithms to find force and moment component weigh the normal vector to the hand mannequin higher than the other components. This is most likely what is happening in the tests. Forces captured in other directions could be due to a few different reasons. One reason forces are experienced in other directions that are not captured by the glove is that shear forces may be experienced by the fingertips or hand mannequin. Forces may also be experienced by other places of the hand mannequin, which were attempted to be accounted for using a scaling factor. Forces applied outside of the fingertip may be greater in some tests than others but was controlled using the same placement of the hand mannequin for each repeated test. In our potential future applications forces experienced outside of the fingertip are negligible [1, 9].

The developed glove provides force data in the primary force component direction within 5.85% of the total forces experienced. Moment calculations are inaccurate as they contain high errors of total force experienced (58.78%). These calculations are large but maintain a consistent linear trend with the true moments. Calculation of the moment arm must be calculated prior to calculation of a moment, in applications this may provide challenges that must be accommodated by assumptions

thus increasing inaccuracies. Forces that are not in the normal direction are not able to be captured by the device. Forces applied outside of the fingertips are also not able to be captured. Another limitation is that the device cannot capture a moment in the x direction as this corresponded to a twisting moment on the spherical object. This calculation is not a simple multiplication of force by a moment arm and would need to be calculated using more sophisticated algorithms. Calculation of the bending moment also assumes that all force is applied at the center of the Styrofoam sphere and provides a limitation in the moment calculation. Another limitation that this design has is the use of one magnetometer on the hand mannequin. *Yaw* angles for each finger are calculated from the magnetometer data on the back of the hand. This calculation cannot account for gestures like spreading of the fingers or *yaw* rotations applied to specific fingers and not the hand. The use of one magnetometer in the design was sufficient to capture gestures used in our application and was done so to reduce cost of the device. The amount of test trials and orientations could also be increased to improve the statistical power of these validation studies as well.

Future developments of the glove could use various additions. One improvement would be to use a smaller resistor value in the electrical schematic that would increase the sensitivity of the pressure sensors. The pressure sensors used had a large range from 0-133N and the upper values were never close to being reached. Improvements in pressure sensor calibration and sensitivity may provide more accurate results. Use of shear force sensors in addition to pressure sensors could provide more insightful data of forces applied in non-normal directions. An obvious addition would be to include more pressure sensors outside of the fingertips as was done in our group's previous work [1,

9]. Use of more sophisticated algorithms such as the Kalman filter could improve motion tracking capabilities, however, it would require more time to analyze in the system (Appendix, A.3) [27, 28].

E. Conclusion

The developed motion tracking and force monitoring glove does have the capability to measure a primary force component in any direction within 5.77 N or with 5.85% error. There will be improvements in accuracy based on improvements of the algorithms used to obtain forces and moments. There are many sources of error that have increased inaccuracies of the device. One large source of error could be that calibration of the pressure sensors could be improved. Pressure sensors may need to be affirmed with a force plate by themselves before being integrated into a system to ensure each pressure sensor is properly calibrated. As the values of forces calculated by the glove are derived from these numbers, this source of error is relatively large. Another large source of error is the fact that this study did not control for the total application of force at only the normal direction of the pressure sensors. Human error could have a large contribution to forces experienced outside of pressure sensing area as well as force in non-normal directions which could contribute a large portion of the AVGE by itself. Offsets of yaw value were determined from only one sensor and could potentially induce an error in the system; however, we assume this to only be evident in few of the orientations tested. This device has many potential applications and can be improved or altered depending on its application [1, 8, 22, 27, 28, 45]. Through the use of IMU and pressure sensors our research group was able to develop a device that can

record forces and moments applied in any direction. The developed glove can be used to provide directional force and moment data for the investigation of shoulder dystocia management [1, 8, 9, 11, 23, 24].

Chapter 4: Design Process

A. Introduction

Shoulder dystocia (SD) is a potentially life threatening obstetrical emergency that occurs, typically when the anterior shoulder of the fetus is caught behind the symphysis pubis of the mother after delivery of the head [3]. Oxygen loss and potential death may result if the infant is not delivered. This emergency situation leads to a rapid delivery that can result in dangerous forces experienced by the infant, causing an increase in fetal injuries in SD cases. One review of 285 shoulder dystocia cases found the rate of fetal injury was 24.9% (71/285) including 48 cases of (16.8%) brachial plexus palsy, 27 (9.5%) clavicular fracture, 12 (4.2%) humeral fractures, and one neonatal death due to ischemic encephalopathy [50]. Thus, the delivery must be controlled to avoid either excess total force, or incorrect application of normal force [3]. Brachial plexus injury (BPI) is one of the most severe fetal injuries that occur. BPI is a nerve praxis that if severe may permanently compromise shoulder and arm muscle control. Transient injury occurs in about 10% of all of SD cases, and the injury is permanent in about 1% of those (1.5 out of 1000 live births) [4]. Prior studies have identified risk factors for SD, but the majority of instances remain unpredictable, and because of this, SD management is of utmost importance. There are maneuvers to resolve the impasse, but the best maneuver or ideal sequence of maneuvers is unknown [6] . Several studies have concluded that hands on training using SD simulation based techniques is the most efficient way to teach care givers how to manage SD with minimal risk to the newborn[7, 10]. Minimization of forces and moments applied are likely critical in preventing injuries, but it is unknown whether maternal or clinician applied forces, or some combination of

both may cause injury [23]. Previous studies have quantified forces applied by a clinician during delivery but have neglected to record bending moments which have been shown to cause a large amount of brachial plexus stretch [1, 5, 9, 11]. Measurement of the forces and moments applied by clinicians during delivery could offer a quantitative tool to assess training effectiveness. Combined with measurement of maternal generated forces, such a tool would allow for the first time, the measurement of the total work required for delivery. This force sensing tool could also provide important information in improving birthing simulations as the maternal birthing simulations are relatively crude.

B. Previous Glove Design

Previous work has been performed by our research team investigating shoulder dystocia with the same goals we have today: to quantify clinician-applied forces during live deliveries, specifically shoulder dystocia cases, and in mock deliveries [1, 9]. In order to achieve these goals, our research team previously developed a pressure-sensing glove to be worn by clinicians during mock deliveries as well as live deliveries [1, 9]. The design of these gloves focused only on force sensing and neglected to include directionality [1, 9]. The original glove designed contained pressure sensors (*Vista Medical FSA*) at two positions on each finger; proximal and distal positions [1, 9]. The fifth metacarpal also contained two additional pressure sensors in addition to the proximal and distal positioned sensors.

The previously developed pressure sensing glove was tested first in mock deliveries. Mock deliveries consisted of a maternal mannequin as well as a fetal mannequin. An infant was held in the pelvic region of the maternal mannequin and difficult deliveries were simulated by the holder providing additional resistance to the fetal mannequin being birthed [1, 9, 24]. During these simulations, the clinician wore the pressure sensing gloves and pressure sensitive film (*Fujifilm Prescale*) underneath the



Fig. 4.1 Pressure sensitive film displayed where highest pressure densities were found during the birthing simulations [1].

glove [1]. The addition of pressure sensitive film displayed the pressure distribution and magnitude during the mock delivery (Fig. 4.1). This information is helpful when designing additional iterations of the glove to know where on the hand pressure is highest [1]. However, we must keep in mind that this study was performed in a mock delivery setting and does not exactly mimic forces exhibited in live deliveries [1, 9].

Conclusions from the previous study provide insightful information that has been used in our current study. One conclusion is that the fingertips provide significantly larger pressures than the rest of the sensors; distal portions on the finger as well as the palm provide negligible pressures [1]. Several of the pressure sensors reached maximum pressure values of 20psi or 0.14 MPa, however, this was rare [1]. Several suggestions from this study were to include motion sensing capabilities as pressure data alone cannot capture proper positioning of the clinician hand [1]. Direction of force as well as bending moments are important parameters for a new glove to be able to capture as these may provide more information regarding brachial plexus stretch [5]. Bending of the head from ear to shoulder has been shown to increase brachial plexus stretch [23].

C. Functioning Environment

The developed device must be able to be worn during a mock delivery as well as a live delivery. In order for the glove to be successfully used in these environments there must be knowledge about how it will affect the device functionality. A device will be able to be worn by the clinician delivering the baby while not getting in the way of a successful birth. A hospital bed will be in the middle of the room along with monitoring technology close by. Monitoring technology is used for the fetus as well as monitoring the mother's vital signs and additional medical equipment is nearby for the case of emergencies. In order for the glove to be minimally obstructive, a wireless technology would be ideal in a live delivery room. Another cord would provide more chances for nurses to trip or other medical equipment to get tangled with. In a live delivery room

there is not much room for a laptop and long wires to be and should be placed off to the side of the room as to not affect the labor process. This glove design was a prototypical device and does not have wireless technology, as wireless technology will be a last step in product design.

D. Qualitative Needs

- Must be able to capture force and bending moment in 3D
- Must be usable in clinical environment: live or simulated births
- Must be able to wear the device underneath a latex glove
- Must not require an external source or instrument
- Forces and moment data must be able to be analyzed using software (MATLAB, LabVIEW, etc.)
- Data must be able to be output in real-time (around 30Hz)
- Minimize delay between sensor data

E. Design Specifications

- _ Capture forces ranging from 0-100 Newtons
- _ Minimal profile (thickness/height) < .01m
- _ Euler angle calculation error < 10°
- _ Total Force calculation error < 5%
- _ Data output > 30 Hz
- _ Delay between sensors < .05 sec

F. Glove Design Choices

Our research group has previously developed pressure sensing gloves that were successfully used in birthing simulations and live births. Previous glove design were able to measure pressure successfully but were unable to capture direction of applied force [1, 9]. The current glove is required to have motion sensing capabilities as well as pressure sensing capabilities. Measurement of force and moment vectors in three dimensions is the design goal. A force vector normal to each finger as well as the palm which can be summed to give a total force vector is output by the glove. The glove is designed to be used by clinicians while delivering infants in live births or birthing simulations. The glove must be able to be worn in a clinical environment and should have a small profile, as it must be worn under a sterile latex glove during live deliveries. The glove must be able to measure forces ranging from 1 N all the way to 100 N as this force has been recorded in other simulations [22, 24]. In order for the glove to be useful in the clinical field the device must not rely upon an external light source as there may be line-of-sight issues.

Bending resistors have been used to capture finger bending and positioning, however, these resistors do not provide the extent of data that we must have to compute an accurate force vector [26]. There are many optical-based sensor systems including the *Leap Motion Controller (LEAP Motion)* that could provide data usable in our system. However, the clinical environment provides obstacles that would present a line-of-sight issue during delivery (legs of mother, hospital bed, gown etc.). The use of an external transmitter would be impractical to be used in a clinical setting so the optical-based sensor systems were avoided.

Inertial sensors and magnetometer sensors combine together to form inertial monitoring units (IMUs). These sensors are chosen because of their capabilities of capturing motion without an external light source or transmitter. IMUs also can provide more information than other sensors that is important in determining the device orientation [27, 28]. The developed glove contains 11 sensors: 5 accelerometer-gyroscope sensors (Ivensense MPU6050), 1 sensor composed of an accelerometer (Analog Devices ADXL345), a gyroscope (InvenSense ITG-3200), and a magnetometer (Honeywell HMC5883L), as well as 5 analog pressure sensors (Tekscan ESS301). On the back of each fingertip the *MPU6050* sensor was placed to determine orientation of each finger. Centered on the back of each hand is a 9 degrees of freedom *Sensorstick* which is an accelerometer, gyroscope, and magnetometer sensor combined on one board. A magnetometer is necessary to be included in the design as to provide a yaw value or rotation around the z-axis. On the palm-side of each fingertip a pressure sensor is placed. Figure 4.2 shows an image of the sensor placement on the glove wearer. The pressure sensor was chosen because of its superior accuracy, flexibility and its minimal thickness. The low profile of these sensors allows the glove to be minimally obstructive for the glove-wearer. Accelerometers, gyroscopes, and magnetometers were chosen above other sensors because of their advantages in the clinical setting.



Fig. 4.2 On the left of the image, blue rectangles indicate the placement of *mpu6050* sensors. In the middle of the hand is a red sensor that is the *Sensorstick*. The palm side displays 5 round ovals that represent pressure sensor placement.

Accelerometers, gyroscopes, and magnetometers do not require an external transmitter and can be very small. These IMUs can be easily integrated into a glove device and will be minimally obstructive.

G. New Glove Design

G.1 Sensors and Hardware

a) Motion Sensors

Motion sensing capabilities were achieved using accelerometers, gyroscopes, and magnetometers. While accelerometer and gyroscope were sufficient in determining the pitch and roll of the device, *yaw* is more accurately calculated using a magnetometer [27, 28]. Previous motion capture gloves neglected to use magnetometers in their design [25]. Accelerometers measure the gravity force vector and external accelerations experienced by the sensors. Gravity acceleration is the only acceleration experienced when an accelerometer is static. If the sensor is static, then the resultant data reflect the gravity vector in three dimensions. The resulting gravity vector provides information regarding the orientation of the accelerometer itself which is the fundamental knowledge used in orientation determination of the glove. The Euler angles corresponding to the hand orientation is shown in Fig. 4.3. A gravity vector is in the negative *z-direction* and rotations around the *y-axis* and *x-axis* can be obtained from the gravity vector alone. However, if the hand in the *xy-plane* is rotated around the *z-axis* (*yaw*) the gravity vector will not change (Fig. 4.3) [27, 28].

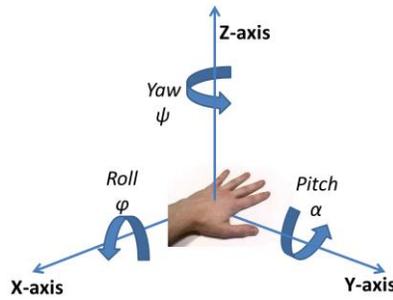


Fig. 4.3 Pitch, roll and yaw defined in three dimensional coordinate planes.

Orientation can be calculated by an accelerometer alone; however the accelerometer sensor can be susceptible to noise. One way an accelerometer can be corrupted is by experiencing transient accelerations. These will induce an acceleration on an axis that could be different than the gravity vector and the gravity vector will be difficult to calculate [27, 28]. While a device is moving one would still like to know the orientation of the device. This is why a gyroscope can be used in conjunction with the accelerometer. A gyroscope measures the angular acceleration of a device and is accurate during transient rotations. Gyroscopes provide a measurement in the “short term” while accelerometers are more reliable in the “long term” and we use this information in our fusion of data and filtering algorithms. Gyroscopes do contain errors that occur while the device is at rest. Angle can be determined by integrating the output of the device, and while the device is at rest the angle output can be corrupted by drift from bias errors [27, 28, 46]. Drift in gyroscopes appear as a linear increase or decrease in the angle output while the device is at rest (Fig.4.4). There are ways to combat this error, one of which is used in the data fusion technique which will be described in the later portion of this paper.

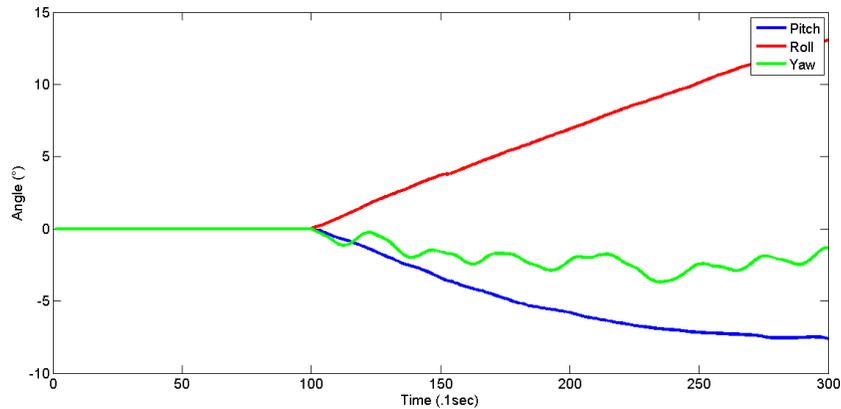


Fig. 4.4 A test run prior to proper filtering algorithms shows drift in the roll and pitch axes. Drift is characterized by a linear increase or decrease which is evident in the above image.

Magnetometers are the last of the motion sensors described. Magnetometers sense the strength and direction of the magnetic field and can be used to determine orientation and heading [27, 28, 46]. Hard-iron and soft-iron errors may occur in magnetometers and are caused by external sources of magnetism. A hard iron distortion is due to external magnetic sources that cause a constant field to be measured by the magnetometer. In order to accommodate for this hard-iron effect, the magnetometer is rotated 360° around the z-axis and then the maximum and minimum for x and y values are averaged to obtain the hard-iron offset value [48]. Soft-iron effects require a more sophisticated algorithm to remove and can be minimized by reducing ferromagnetic materials near the device [27, 48]. Magnetometers in three dimensions provide the tilt as well as the orientation to magnetic North, providing an accurate way to obtain yaw rotations [27, 48].

b) Pressure Sensors

Pressure sensors output a voltage response to pressures applied to the sensing region of the sensor. The analog pressure sensors have a wide range of force sensing capabilities and are flexible and extremely thin. Minimal thickness is ideal for these pressure sensors, as the fingertips of the wearer should be unobstructed by the sensors. The pressure sensors function by acting as variable resistors and their resistance is compared to a feedback resistor, R_2 shown in Fig. 4.5. R_2 can be changed to make the pressure sensor more or less sensitive. These sensors, as with other sensors used must be calibrated prior to use in the device which will be described later in this chapter.

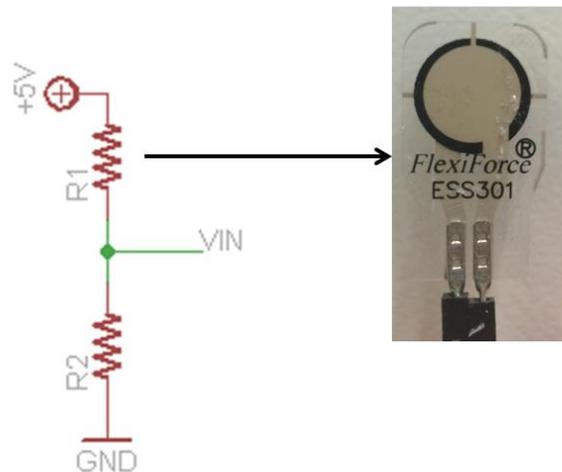


Fig. 4.5 The used Flexiforce sensor (*Tekscan* ESS301) acts as a variable resistor R_s . The response of the output voltage determines the pressure applied to the sensing area.

c) I₂C communications

The motion sensors involved in the device communicate using I₂C communications or (Inter-integrated circuit). I₂C communications hold several advantages over asynchronous serial parts. One important advantage is that serial ports are designed for the communication of only two devices and do not require any external hardware to function properly [51]. SPI communications can be used in many situations but there are many pins and wires that must be connected and can take up space on a printed circuit board (PCB) [51]. I₂C communications require only two wires: SDA (data line) and SCL (clock). The I₂C protocol can support up to 10008 devices (e.g., motion sensors in this design) and can allow more than one microcontroller to communicate to all devices. In order to begin the procedure a start condition pulls SCL high and SDA low and the next step is to call on the address of the motion sensors [51]. Each sensor has an available address so it can be called upon; the *MPU6050* sensor has two possible addresses. We use five *MPU6050* sensors which each have two possible addresses; this means that not every sensor can be called upon individually. We use a multiplexer to solve this addressing problem.

d) 16-Channel Multiplexer

A multiplexer routes one input signal and connects it to its output line. In this 16-channel multiplexer we chose, there can be 16 input signals to be chosen from. Essentially this device is a switch that can be addressed to read a specific input channel. The data line of each *MPU6050* sensor was attached to separate channels of the multiplexer. A series of high and low signals are sent to the signal address pins of

the multiplexer. These signals work in binary to choose a channel to read from.

Crosstalk is a problem that must be addressed and in order to combat this one channel is addressed, data is read, and the channel is closed to switch to another channel. In order to integrate all of these devices and read data off of the sensors a microcontroller was incorporated.

e) Arduino Due

In order to receive data from all of the sensors and interpret data, an Arduino Due (*Atmel SAM3X8E ARM Cortex-M3 CPU*) was used as Arduino microcontrollers are user-friendly. The board contains 54 digital pins, 12 analog pins, and is based on a 32-bit core microcontroller. This board is powerful and was chosen for its strength and many pins: digital and analog. The Due was chosen as the glove device may need more capabilities in future designs and our group was positive this microcontroller has the potential to accommodate future improvements. In order to communicate with a laptop, a USB cable was connected to the microcontroller. Programming for the Due was completed on free Arduino software (*Arduino 1.5.6*) which the user codes in C++.

f) Calibration of Sensors

Each motion sensor and pressure sensor used was calibrated prior to use in the device. If sensors are improperly calibrated there may be large errors in the output data [27-30]. Motion sensors were the first to be calibrated and each sensor integrated on each chip must be calibrated by itself. The first step was to connect the device to a laptop to read data off of the sensors via LabVIEW software. The accelerometer was tilted to where the positive x -axis is vertical. The increases and reaches a maximum (around 16384) when completely vertical. In order to maintain the sensor completely vertical, the sensor was placed against a wall. The sensor was held in place for 10-15 seconds and data was read. From this data the maximum value was determined. This same procedure was repeated for each of the axes in positive and negative directions. To determine the offset of the x and y accelerations, the accelerometer was to remain still and upright in the xy -plane with the z -axis pointed upward. The accelerometer then rested for 10-15 seconds and data was recorded. The x and y values were averaged to determine what values the offsets are (as they should be 0 in an ideal situation), these are known as biases or offsets [27-30]. During this procedure one can obtain gyro

$$x_{max} = 17116, x_{min} = -17936 \quad \frac{(17116+17936)}{2} = 17526 \quad (\text{min to max should equal } 2 \text{ g's})$$

$$\frac{(A_{rx}-b_x)}{17526} + .0234 = A_x \quad (\text{adding } .0234 \text{ makes } 17116/17525 = 1\text{g})$$

Fig. 4.6 Data from raw accelerometer data \vec{A}_r is used to convert to accelerations in terms of gravity force \vec{A} . This example uses the x component of the accelerometer and example values of minimum and maximums. The range of the raw data is determined and divided by 2 to find what is equal to 1 gravity force (accelerometer range = $\pm 1\text{g}$). The bias value is subtracted from the raw accelerometer value and divided by what is equal to 1g. .234 is found by plugging in b_x and solving for when A_x is equal to 1 and A_{rx} is at its maximum.

offsets. The accelerometer and gyroscope integrated sensor sat still for 10-30 seconds and data was recorded. The average of the x , y , and z values of the gyro provide the gyroscope offset. When the maximum, minimum, and bias values were determined for both gyroscope and accelerometers the raw data can be converted into usable values (Fig. 4.6).

Magnetometer calibration requires a rotation around each axis to determine minimum and maximum values for each axis. The average of maximum and minimum values for each axis is averaged to determine the offset. This calibration accounts for hard-iron effects caused by external magnetic fields which create a constant measurement offset error [48]. Soft-iron errors are due to ferromagnetic materials around the sensor which can alter the local magnetic field. Removal of ferromagnetic materials near the sensor will reduce these soft-iron effects and is done so in our applications.

Pressure sensors are calibrated by connecting the sensors to the circuit using a feedback resistor to compare the analog response from the sensor viewing in LabVIEW and MATLAB. The theory behind calibration of an analog pressure sensing device is simple; apply pressure to the device and record the output voltage. The *ESS301* pressure sensors were small enough to fit on a human fingertip and in order to place known pressure on to the sensing area a “puck” must be placed on the sensing area to ensure pressure is only focused on this area. In our calibration, a dime was placed on top of the pressure to be used as a “puck”. Weights were then placed onto the puck and voltage responses were recorded for each weight. A plot displaying the voltage response with the associated pressure or force in this case was used to find a

trend between the two. An expected response is a linear one, but in our calibration this was not the case. A cubic fit provided best results in the device and is shown in Figure 4.7. Linear fits were tested and compared to other fits to determine the best functional relationship between voltage response and force applied. The cubic fit was determined for each sensor individually and then the equation of the fit was used for each separate sensor. The cubic fit provides a problem of 'bumps' in the data where the data changes from increasing to decreasing back to increasing (Fig. 4.7). In order to account for this error, a linear fit was used for values below 3V to the origin. A linear fit of the calibration values that were at 3V or less was made to provide more reliable data at these voltage levels. Outputs max out at 5V which should correspond to around 130N; this was not seen in many cases of the calibration procedure as several sensors were very close to maxing out at much smaller force values.

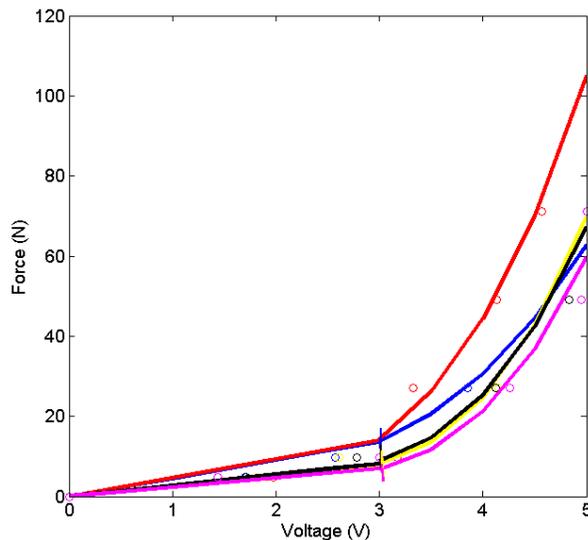


Fig. 4.7 Each pressure sensor is plotted with their voltage response on the x-axis and corresponding force value in N on the y-axis. The cubic fit does not model calibration data at $< 3V$ but this is accounted for by taking a linear fit for values at or below 3V value to the origin. Blue corresponds to thumb sensor data, red is index sensor data, yellow is middle sensor data, black is ring sensor data, and magenta is the pinky sensor.

When calibration of each sensor was completed, the next step was to combine all sensors and devices and begin validation studies. An electrical schematic was developed to outline how each sensor was connected in the glove device. Data from the device is acquired through a LabVIEW program and data analysis was completed in MATLAB software. Several challenges and problems were encountered at every step in the design process. The design process does not entail only hardware development but includes software development, experimental design, data analysis, and algorithm development.

H. Challenges

Challenges through the product development process are to be expected. A primary challenge was to choose a sensor system that was capable of capturing orientation of a hand. The *Leap Motion Controller (LEAP Motion)* was first investigated to determine if it could be used in orientation determination. Extracting data and recording data via LabVIEW was the primary difficulty in working with the Leap Motion device. Normal vectors to each finger were able of being extracted, however, tracking was inconsistent and line-of-sight issues proved to be a determining factor of using other sensors for motion capture.

A commercially available motion sensing glove was primarily explored to see its capabilities (*Meta Motion Acceleglove*) [25]. A LabVIEW program was developed to read data from the device and analyze the raw data. Several programs were needed to calibrate and read the data from the Acceleglove. A challenge was to output data from LabVIEW into MATLAB and to analyze the raw data. Raw data provided was difficult to

interpret and as the Aceleglove only had accelerometers, there were significant drawbacks to the design. To improve capabilities of the glove design our group moved towards the development of a separate product that used accelerometers, gyroscopes, and magnetometers. Determining which IMUs to be used presented a challenge in itself and was the next step in the process.

After selecting proper IMU sensors the calibration was the next step which did involve several challenges. While a calibration protocol was described above, there were several challenges that one had to be aware of during accelerometer, gyroscope, or magnetometer calibration. When tilting the accelerometers, axes names did not always reflect how the data were being output. Labeling on breakout boards was not always correct: *x-axis* readings were output later than *y* and *z-axis* on several of the sensors and the direction of positive and negative readings were not always properly labeled. If axes are misinterpreted then algorithms to find orientation would provide inaccurate results. The range of measurement for the inertial and magnetic sensors also provided a challenge; this was solved by observing similar projects online to decipher proper ranges and then the ranges were programmed into the Arduino code.

Pressure sensors also were difficult to calibrate because of their small size and sensing area. In order for sensors to be properly calibrated, weight must be only placed on the sensing area. To solve this problem, we used a dime as a 'puck' to be placed atop the sensing area and then weights were placed on top of the dime. A cubic fit was the best fit for the sensor data but was not good as a linear fit was expected and this presented a challenge of interpreting the fit. Around 3V several of the cubic equations exhibited trends that did not make sense as the equation changed direction multiple

times. In order to accommodate for this trend, data were fit linearly from 3V to 0V.

Weights used in the calibration procedure were limited and we used a .5 kg, 1 kg, 2.77 kg (6lbs), 5 kg, and a 7.27 kg (16lbs) weight.

When sensors were successfully calibrated the next step was to connect all sensors and devices to ensure proper functioning of the whole system. Since there are five sensors of the same type (*MPU6050*), and these sensors each only has two addresses to be called upon, *I²C* communications will not be able to read from each sensor individually. In order to solve this problem, our group chose a multiplexor that can be programmed to switch between inputs, thus allowing a communication to all sensors with a small delay (<.05sec). When using the multiplexor in the design, our group had to keep in mind a few things to ensure proper functioning of the device. The various sensors and signals all attached to the same device provide a possibility of electrical crosstalk. To combat this, the channels must be chosen and data read off and then closed before switching to the next channel. If crosstalk arises then the validity of the signals may be corrupted for each finger. The multiplexor must also have decoupling capacitors to improve data filtering. This multiplexor chosen already had a capacitor integrated into the board connecting 3.3V to *GND*.

Reading data from the device was a task that proved to be challenging in itself. In order for the data to be read from Arduino device to be used into MATLAB, our group used LabVIEW as an intermediary to acquire data from the device and export it to MATLAB. LabVIEW provides an easier environment to interact with the device and MATLAB is more conducive to data analysis. LabVIEW accesses the data from the Arduino software, while Arduino software is used to access data from hardware and

export data to LabVIEW and ultimately into MATLAB for analysis. Data output from the glove was aligned so that the data from the hand was first read and the fingers following, this order was chosen to line-up with the software analysis. If an interrupt in the data stream occurred then the MATLAB program would skip several lines to realign the sensors to their according finger position. Aligning data with its corresponding sensor is essential to the functionality of the device so this was of utmost importance.

When the device was properly calibrated and functioning, testing was performed. In order to assess accuracy of the motion capture device we had to compare the glove with an external motion capture system. We chose an electromagnetic sensing device (*trakSTAR*) with *MotionMonitor* software to compare our device with. The next step was determining the testing environment and apparatus. Choosing a hand mannequin to place the developed glove prototype was the next step. Several mannequins were discussed before choosing a mannequin that had bendable fingers as well as an arm with an attached clamp. These features provided a way to clamp the hand to a table and keep the hand stable. The environment is shown in Figure 4.8. The glove was clamped to a table out of electromagnetic sensing range and the glove was placed in front of the *trakSTAR* transmitter. All metal and ferromagnetic materials were kept outside of the sensing range (660mm) of the *trakSTAR* transmitter to ensure no interference occurred. Placement of sensors on the hand mannequin was challenging and it was necessary to place electromagnetic sensors on top of each IMU to ensure the two sensors would be comparable (Fig.4.9). Figure 4.9 displays the electromagnetic sensor on top of one of the IMUs on the pinky finger of the mannequin. All other fingers on the hand mannequin have only an IMU on top of them for this image.

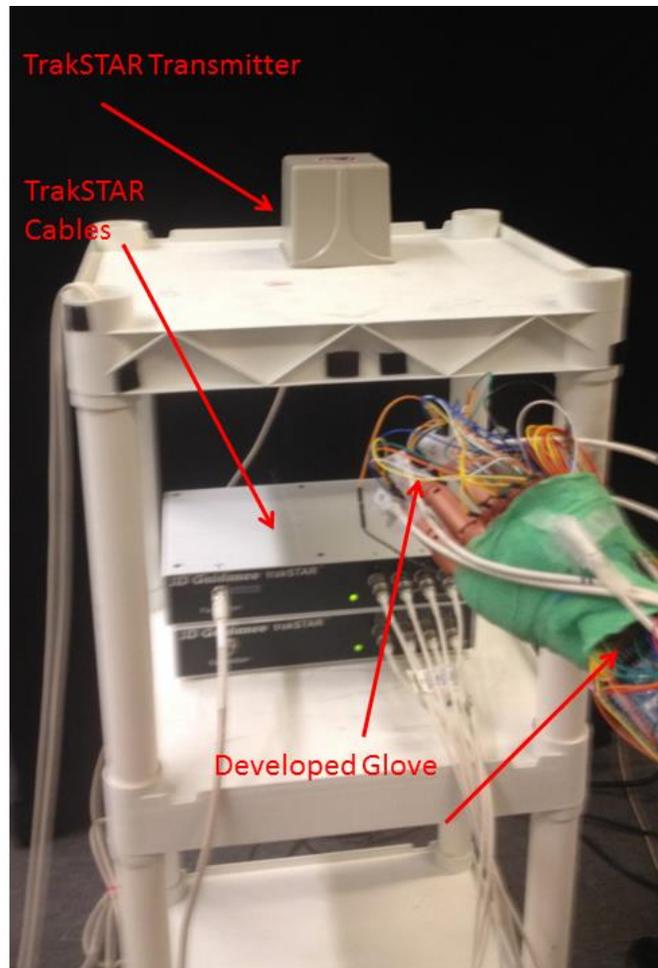


Fig. 4.8 The motion capture testing apparatus is shown above. The trakSTAR transmitter and cables are labeled and placed upon a shelf made of non-ferromagnetic materials. The glove device is attached to a table nearby, but outside of electromagnetic sensing range. Wired electromagnetic sensors are also attached to the developed glove prototype.

The attachment of both of an electromagnetic sensor and an IMU on each finger created a crowded hand mannequin. Aligning coordinate planes of each of the electromagnetic sensors and IMU sensors was necessary prior to completing each test. After the hand mannequin was clamped to the table and placed in a prone position, tests could be performed. Each test began with the hand in a prone position which it lay still at in the *xy-plane*. After 10 seconds, the hand mannequin was moved by a user into a known position and then the hand would rest at this position until a time of 35 seconds

had ended. Achieving the same known position was difficult to control and is a potential source of variability in the results. In order for the test to run correctly it was crucial that the hand mannequin and its electromagnetic sensors stayed within range of the transmitter. If the electromagnetic sensors came too close to the transmitter then results will not be reliable. To prevent this from occurring we viewed the results from each short 35 second test viewing a chosen sensor to compare with the sensor results from the glove. The short testing time period allowed for a simple “eye-test” to be performed on results to be sure each sensor system was capturing what it needed to.

Disconnection of wires during tests provided an additional problem to these validation studies. Wires were soldered together in the glove system but became loose after adhesion to the hand mannequin and would come detached on occasions. If a detachment occurred then a test had to be performed again. To fix this problem, the device was re-soldered in areas of weakness. Future glove designs should be adhered permanently on a printed circuit board (PCB) and this problem would not occur. A problem that also occurred was that data “jumps” occurred in the data at $\pm 180^\circ$ which created large RMSE when comparing the two systems. To combat this problem, an algorithm was developed between two systems detected this significant difference ($>300^\circ$) and corrected for it (Appendix, A.2). Another challenge that was presented while comparing two sensor systems was that because the electromagnetic sensors and IMUs were not in the *exact* same position, an offset must be determined to compare the two sets.

The placement of the electromagnetic sensor was shown in Fig. 4.9. One can see that the electromagnetic sensor is placed on top of the IMU. The slight difference in

placement will account for a slight difference in rotation as one sensor may not rotate as much as the other. In order to calculate an offset for each sensor, the hand was rotated 45° in each of the Euler angles; positive and negative directions. An offset was determined by using the difference between each sensor system for each direction. Offsets for each sensor were to be constant throughout the testing of the motion sensing systems and can therefore be determined prior to testing without affecting result validity. Another calculation that had to be made prior to experimental motion testing was the offset in yaw rotations for each finger. Only one magnetometer was used in the glove system which was placed on the back of the hand, meaning that yaw value for each finger in the system had to be derived from the yaw rotation of the hand.



Fig. 4.9 The electromagnetic sensor was placed atop of the red rectangular IMU sensors for each hand during testing. The red rectangular sensor with five wires attached are part of the glove device.

In order to calculate *yaw* rotation values a *yaw* rotation of 90° was applied to the hand and *yaw* values determined from each electromagnetic sensor were compared to that of the *yaw* rotation angle found from the magnetometer on the back of the hand mannequin. *Yaw* offsets were determined for each finger sensor.

When motion sensing accuracy and capabilities were confirmed, force sensing capabilities were to be tested. The apparatus consisted of a rectangular force plate with a meter stick clamped to it in the middle of the force plate with one end of the meter stick hanging off of the edge (12.7mm). The end of the meter stick hanging off of the edge had a 12.7cm-diameter (5in) spherical Styrofoam shape attached. This size of sphere was chosen as to mimic the size of a normal infant head diameter of 10cm (3.94in) [52]. This testing apparatus is intended to approximate the size and orientation of an infant's head, neck, and torso (Fig. 4.10). Each test performed began with the hand mannequin in a prone position resting for 10 sec. The hand was then rotated to a stable position and a force was applied by the hand mannequin in a normal direction to the palm of the hand for 20 sec. An image displaying the hand mannequin applying force to the Styrofoam sphere is shown in Figure 4.11. Several things need to be accounted for when using this experimental set up to assure proper forces are being recorded by the force plate.



Fig. 4.10 Force testing apparatus is shown above. Two clamps hold the meter stick in place on the force plate and prevent the meter stick from moving when forces are applied to Styrofoam sphere. Diameter of the Styrofoam sphere is displayed as well as the distance of the meter stick to the Styrofoam ball that extends off of the force plate.

One of the first problems that is obvious when viewing the force apparatus is that the clamps will be applying a force when attached to the force plate. To nullify the force applied by these clamps, the force plate was zeroed prior to force application of the glove on the Styrofoam sphere. During each test, force was applied in the direction normal to the sensor directions. Force was applied by a user applying force on the hand mannequin to the Styrofoam sphere. If force was applied in a non-normal direction then then force vectors could not be accurately obtained. IMU sensors output the normal vector to each sensor and this is used to find force vector components.

A problem that comes up during force testing is that forces may be applied by the hand mannequin outside of the fingertips. In several tests, forces were clearly exhibited outside of the fingertip. To account for forces exerted outside of fingertips was to multiply forces recorded by a scale factor for each direction. This scale factor essentially accounts for a larger area of force application, as force is determined by multiplying pressure by area of pressure application. A scale factor was determined in each direction as force application outside of the hand mannequin differed for each dimension (x,y,z) (Appendix, B.1). After all tests were performed, seven had to be discarded because pressure sensors did not record values greater than 1N in any dimension. These tests show that pressure sensors are not capturing force and forces were not exerted on the fingertips. After all testing had been performed; it was found that the device was capable of measuring force and moments correctly for the direction of force application of greatest magnitude.

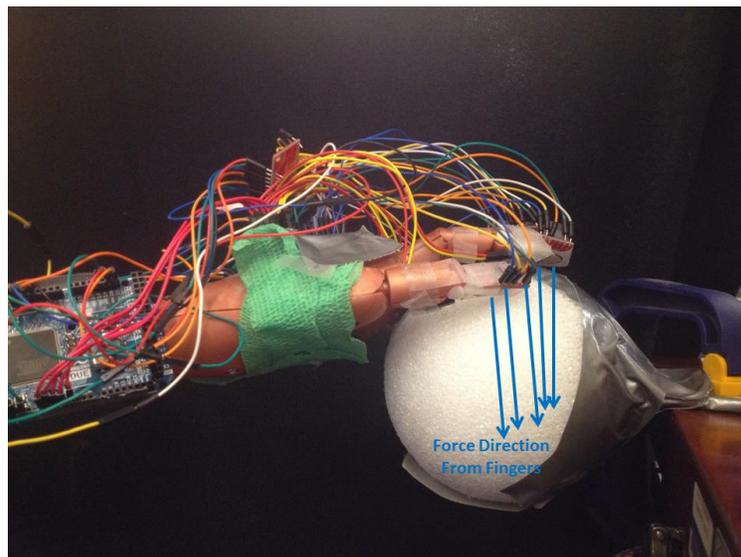


Fig. 4.11 Application of force during experimental trials were performed with placing the hand mannequin on top of the Styrofoam sphere and applying force. Blue arrows represent the direction of force application from each finger.

Several iterations of testing sensors individually were performed prior to developing a complete schematic for the device. In order to assure this electrical schematic would perform correctly, all sensors were tested individually. In the bottom-right corner of the electrical schematic is the 32bit Atmel AT91SAM3X8E microcontroller which is the microcontroller in the Arduino Due board which can operate as high as 84 MHz. This board is powerful and was chosen for this to prevent any limitations from being reached if the design changed. It contains 512 Kbytes of Flash memory and 96 Kbytes of SRAM This board has 12 analog inputs and 54 digital input/output pins as well as 4 hardware serial ports. The board was connected to a computer via USB cable, the board was programmed using Arduino software (Appendix, A.1.). In the schematic one can see that digital pins, analog pins, 3.3V, 5V, SDA, and SCL ports were used.

SDA and SCL ports are used to communicate with the motion sensors in the device; six accelerometers (Analog Devices ADXL345), six gyroscopes (InvenSense ITG-3200), and one magnetometer (Honeywell HMC5883L). One ADXL345, ITG-3200, and HMC5883L were located on one chip known as the 9DOF Sensor Stick board as

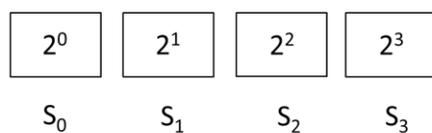


Fig. 4.13 Each S pin is sent high or low and represents a 1 or 0 multiplied by the corresponding value above. The channel is selected by the sum of this value.
ex) 1011 = 1+4+8 = channel 13.

the other five accelerometers and gyroscopes were integrated onto 5 separate boards (Ivinsense MPU6050). Each device communicated via I_2C communications which required only a connection of SDA and SCL lines. The SDA line is the data line and is connected from the Due to the 16-channel multiplexer (Texas Instruments CD74HCT4067) COM port. The COM port is then routed to the chosen channel (C0-C15). S0-S3 inputs on the 16-multiplexer act as an address line, S0 is the least significant bit and S3 is the most significant bit (Fig. 4.13). S0-S3 are wired to digital pins 2-5 on the Arduino Due and addresses are changed by sending the pins high or low. The channel chosen is the sum of the address bits. Crosstalk is a problem that must be addressed when using a multiplexer, to prevent this, one channel was addressed, data was read, and the channel was closed to switch to another channel. The multiplexer allows for each *MPU6050* to be read simultaneously as each of the sensors had the same I_2C address. The multiplexer EN pin was tied to ground with a termination resistor. Termination resistors prevent reflections in the signal. This multiplexer utilizes CMOS technology and functions from 2V-6V, in our case the supply voltage (V_{cc}) was 3.3V.

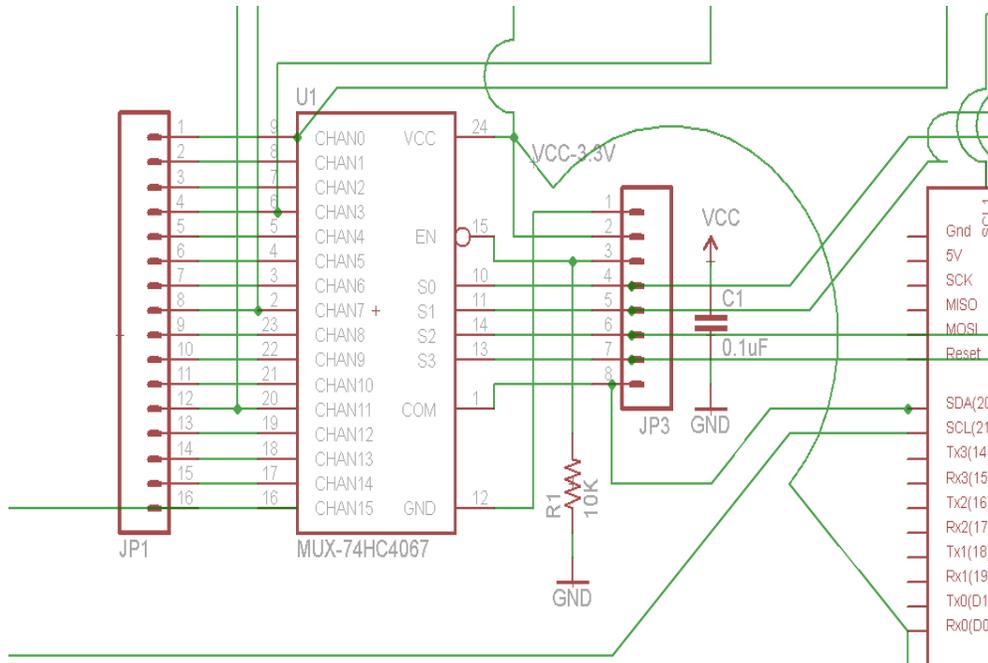


Fig. 4.14 The multiplexer in the schematic is shown above. 5 channels are connected to an *mpu6050* sensor. The right side of the image displays several of the pins of the Arduino Due board.

C0, C3, C7, C11, and C15 are each connected to *MPU6050* sensors. Each *MPU6050* sensor corresponds to a finger on the hand mannequin. SDA line of each IMU sensor is wired to a channel line and their SCL lines are all wired together as the clock lines must be. Each *MPU6050* sensor is powered by 3.3V (V_{DD}), uses pins SDA, SCL, and a GND pin. Another thing that is included in the schematic is the bypass capacitors used (Fig. 4.15). Two bypass capacitors are connected in parallel, one with capacitance of 0.1 μF and the other of 10 μF for each *MPU6050* sensor. Using two bypass capacitors filters out wider ranges of frequencies as higher frequencies may be necessary in future iterations of the device when implementing real-time and wireless capabilities. A general rule to follow is that each integrated circuit requires a bypass

capacitor; however, integrated circuits used in this schematic already had bypass capacitors on board.

The last IMU used was the board containing an accelerometer, gyroscope, and magnetometer integrated onto one board. The schematic displays these as three separate chips; an accelerometer (Analog Devices ADXL345), a gyroscope (InvenSense ITG-3200), and a magnetometer (Honeywell HMC5883L). These sensors all communicate via I_2C communications which require the connection of SDA and SCL lines. SDA1 and SCL1 lines were connected directly to the three sensors on the chip. These sensors could have been connected via the multiplexer but the extra SDA and SCL lines of the Arduino Due provided a simpler solution.

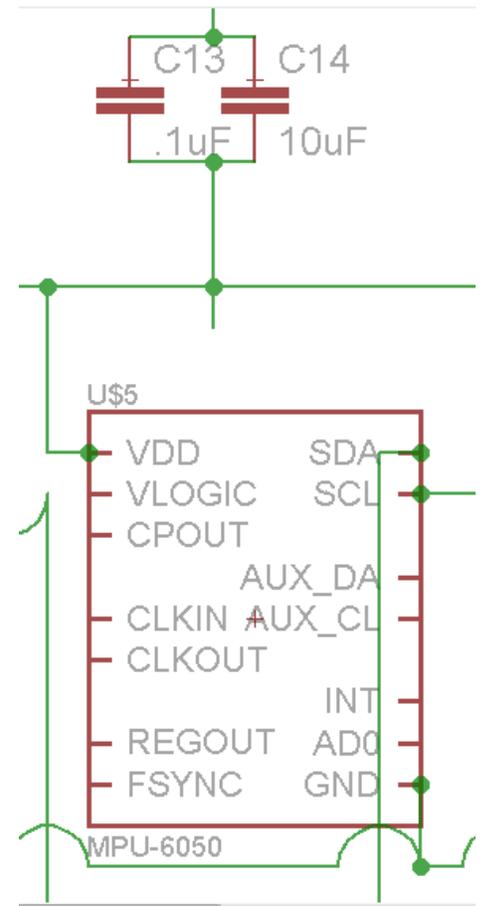


Fig. 4.15 MPU6050 sensor pins and connections are shown. The top of the figure shows bypass capacitors that were included to remove AC noise.

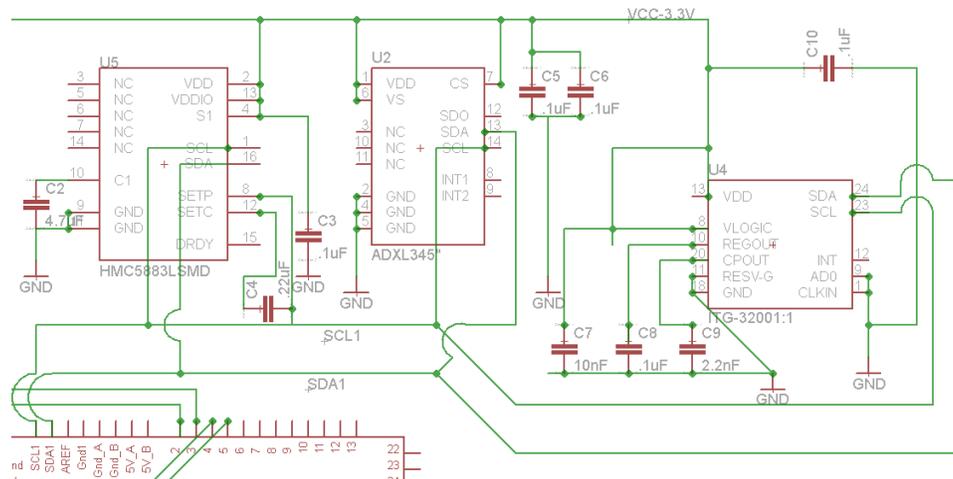


Fig. 4.16 Sensors from left to right are the magnetometer (HMC5883LSMD), the accelerometer (ADXL345) and the gyroscope (ITG-32001:1). In our device these are all integrated on to one chip called a Sensor Stick. Each device is connected via SDA1/SCL1 lines from Arduino Due.

The magnetometer contains capacitors already integrated on board and only requires a connection of SDA, SCL, V_{DD} , and GND pins to function properly. SETP and SETC pins are connected by a 22 μF capacitor (C4) referred to as a set/reset capacitor but is already connected on board. A 4.7 μF reservoir capacitor (C2) and a 0.1 μF bypass capacitor (C3) are also on board. The accelerometer needs only SDA, SCL, V_{DD} , and GND pins attached to read off data. Two parallel bypass capacitors (C5,C6) of 0.1 μF connecting V_{DD} to ground are also included by the accelerometer. The gyroscope schematic includes four capacitors; one charge pump capacitor of 2.2 nF (C9) connecting CPOUT to GND, one 0.1 μF V_{DD} bypass capacitor (C10), one regulator 0.1 μF filter capacitor (C8) connecting REGOUT to GND, and one 10nF V_{Logic} bypass capacitor (C7). Connections to the gyroscope were shared with the accelerometer and magnetometer using pins SDA, SCL, V_{DD} , and GND.

Pressure sensors are the remaining sensors that were used in the device. Five variable resistor pressure sensors (Tekscan ESS301) were used. A pressure sensitive ink fills this extremely thin and small sensor (Fig.

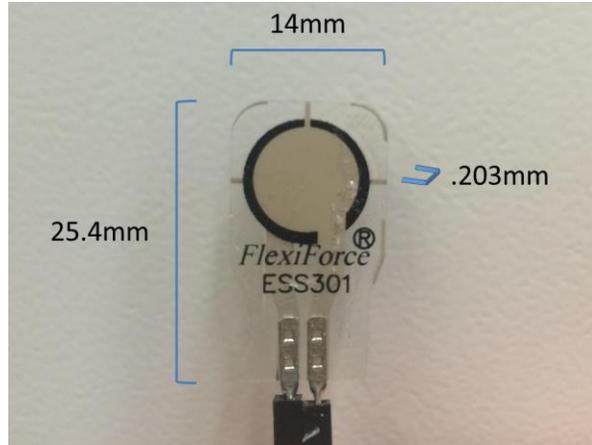


Fig. 4.17 Tekscan Flexiforce ESS301 sensor is shown above with its dimensions. The thickness of the sensor is very important in its glove application as it will be placed on each fingertip (0.203mm). The flexibility of the substrate is also an important feature.

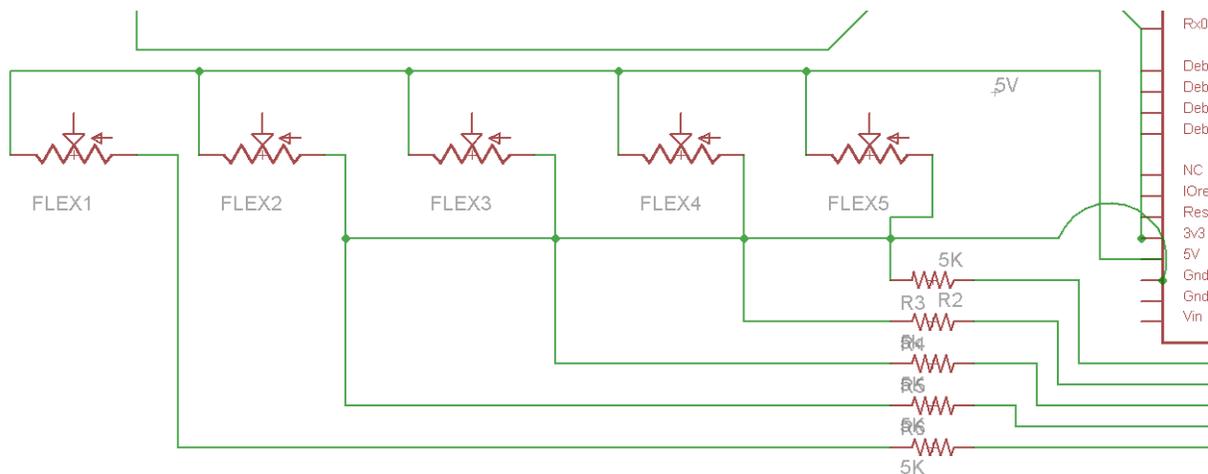


Fig. 4.18 FLEX sensors are the variable resistor pressure sensors and are connected in a voltage divider circuit with a loading resistor of 5 K Ω . These sensors are powered by 5V and connected to correspond analog inputs in the Arduino Due.

4.17). This sensor is connected in a voltage divider circuit with a 5 K Ω loading resistor (Fig. 4.18). Altering the loading resistor value changes the sensitivity and pressure range of the sensors. Using sensor data sheets a force measurement range of 0-133N was determined for a 5 K Ω loading resistor which captures the range deemed

necessary in SD cases by previous work [22, 24]. Each pressure sensor is connected to a 5V pin, GND pin, and an analog pin (A0-A4) on the Arduino Due. Analog signals were converted to a voltage and next into a force using the previously described calibration curve (Fig. 4.7). Pressure sensors were placed with sensing area centered on each fingertip of the hand mannequin. With IMUs placed on the back of each fingertip and the hand, the orientation of each finger could be captured along with the force applied by each fingertip.

1.2. Data Acquisition and Analysis

a) Arduino Script

In order for data to be acquired from the device the Arduino Due microcontroller must be programmed. Using free Arduino software a program was developed to acquire data from each sensor via USB port. Data acquired was raw sensor data and was passed through LabVIEW to be analyzed further using MATLAB software. Arduino software does have the capabilities of analyzing sensor data to some extent but our group wanted to minimize the responsibilities of Arduino to maximize speed of data acquisition. MATLAB is more powerful than Arduino and can complete complicated analysis and was designed to be used in the system environment regardless. Arduino code needs to accomplish the following; initialize all sensors, read data from each sensor, switch multiplexer channel, and repeat (Appendix, A.1).

I_2C communication protocol must be followed in order to obtain data from the IMU sensors. In order for data to be read from the IMU sensors the sensors must be alerted and then addressed. The code first begins serial communication which allows

data to be observed via the serial monitor. A channel is chosen to read from first by altering S0-S3 pins. A start sequence is sent to the read/write bit in order to allow the sensor device to “listen”. The next step is the initialization of the *MPU6050* sensor that is currently being accessed by the multiplexer. Initialization of the *MPU6050* first sets the clock source, sets gyroscope range, sets accelerometer range, and disables the sleep bit allowing for the sensor to be read from. Accelerometer and gyroscope data was then read from the *MPU6050* sensor that corresponds to the chosen channel. In our glove device we began with the sensor corresponding to the thumb of the glove wearer.

In order to obtain an accurate force sensor, the pressure sensor data from the thumb is read at the same time as the corresponding *MPU6050*. Pressure sensors used are analog sensors and only required an initialization of the corresponding pin. Then a simple read function was used to obtain the voltage value from the pressure sensor. Analog signals are 10bit values and must be converted to voltages accordingly. Data is displayed to the serial monitor for both sensors. When the iteration of reading the thumb is complete the channel is switched and a new *MPU6050* sensor is chosen to be read from. This process is repeated for each finger in the order of thumb, index, middle, ring, and pinky.

After data is read from each finger the next step is to read data from the *Sensor Stick*, which is the sensor containing an accelerometer, gyroscope, and magnetometer. This sensor is not connected to the multiplexer but is on its own *I²C* line as described previously. A start sequence is again sent to the sensors prior to initialization. All three sensors on board are initialized by writing the address and register of each sensor

setting the data receiving to a continuous mode. The next step is to read from the *Sensor Stick*. This is simply performed by accessing the correct registers and then reading data bytes from it. Data was displayed in the serial monitor for each of the sensors; accelerometer, gyroscope, and magnetometer. This entire loop of reading data from each *MPU6050*, each pressure sensor, and the *Sensor Stick* is repeated continuously and all data is displayed using the serial monitor (Fig. 4.19).

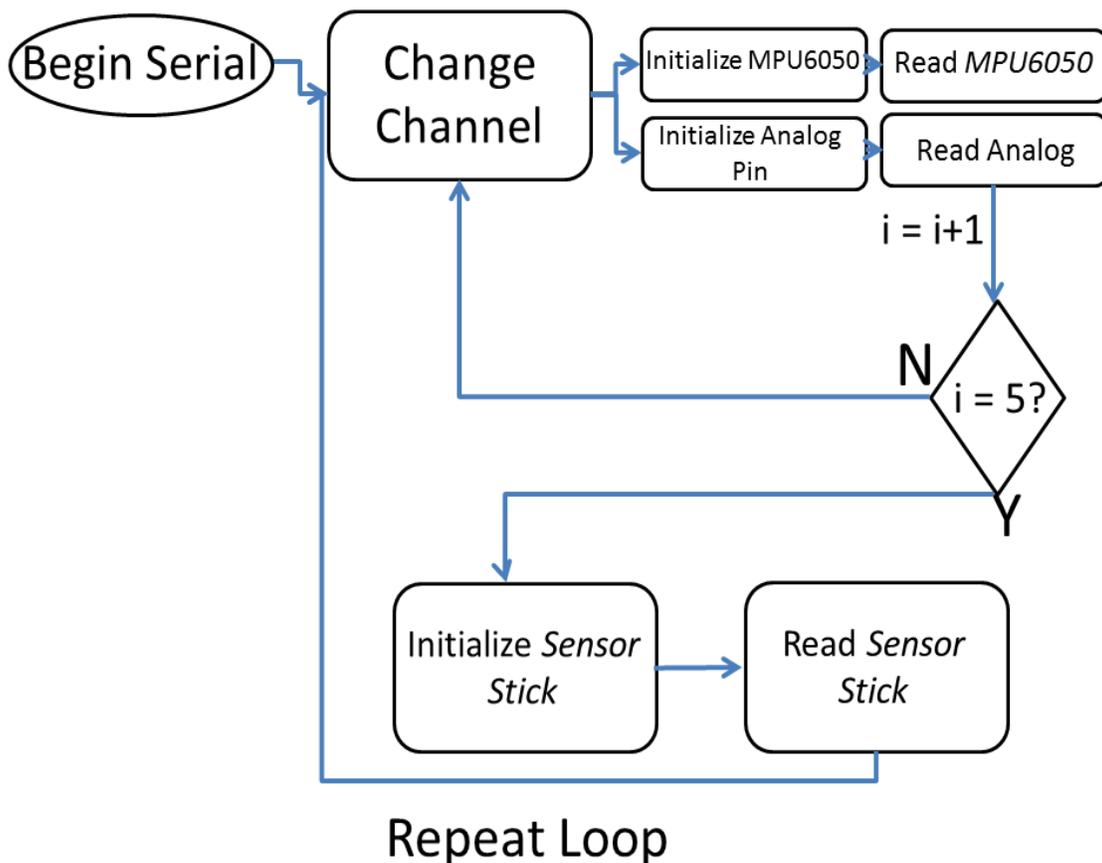


Fig. 4.19 A flowchart describing the structure of the Arduino code is displayed above. Serial communications are first turned on outside of the repeated loop. The multiplexer channel is chosen and each sensor is read from the corresponding channel. After this proceeds five times (i) then the data from the *Sensor Stick* is read. The five repeats correspond to each finger with a *MPU6050* sensor. This loop is repeated continuously during data acquisition.

b) LabVIEW Program

LabVIEW is used as an intermediary between the actual device and its data analysis being performed in MATLAB. Future developments of the glove could use LabVIEW as a way to observe hand position in real-time. LabVIEW excels in communicating with hardware and also can use MATLAB commands which is why it was chosen in this application. The goal of this developed program was to be able to start and stop recording data from the device while also putting the data into a format that could be analyzed.

The front panel of the .vi (LabVIEW program name) consists of an indicator that displays whether numbers are correctly being read or not. A slowing of the program or an object displayed that is not a decimal can be observed using the indicator. The program can be controlled using the start and stop tabs on the front panel or block diagram. The developed block diagram contains several virtual instrument software architecture (VISA) nodes that communicate through the serial port. The first node stands outside of the loop and is a configure node that sets which port to read from (COM8), sets the baud rate (57600), sets the number of bits of the incoming data (8), and sets the parity (none). These settings are chosen to receive data from the microcontroller properly. Another node that exists outside of the main loop is the creation of a .txt file that data is to be written to.

The main loop of the program consists first of a VISA property node that allows the accessing of data bits transmitted from the device. This is “wired” to the VISA read node which outputs the data in a string format and is “wired” to the created .txt file. Data is continuously streamed to the .txt file and “wired” to the indicator while the program is

running. When the program is stopped by the user, the VISA close node is used to close out the opened serial port and the .txt file is also closed. Data is then saved in the .txt file and can be accessed from MATLAB post data collection.

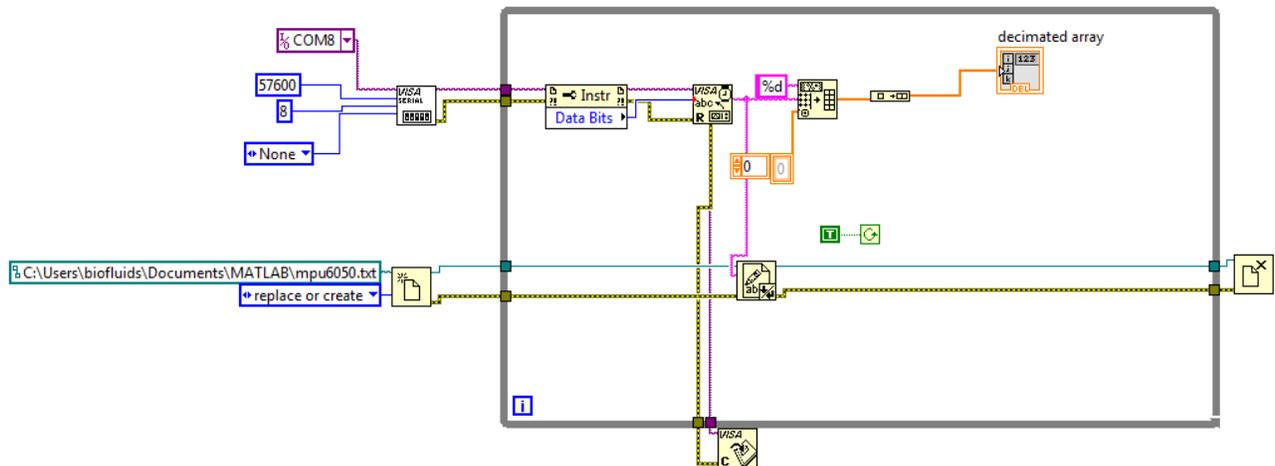


Fig. 4.20 The block diagram of the LabVIEW .vi is shown. The serial port configuration node and .txt file replace or create node lie outside of the main loop indicated by the solid gray line.

c) MATLAB scripts

After data was read into a .txt file it was to be analyzed and converted into force vectors. Several programs were developed to accomplish the goal of converting raw data into force vectors and additional programs were used to analyze experimental data (Appendix). The first step was reading data from the .txt file into a matrix in MATLAB. Data on the .txt file was formatted with pressure sensor data as a first digit then the next six digits were from the corresponding MPU6050 sensor accelerometer and gyroscope x, y, and z data. Data from each finger started a new line and then data from the *Sensor*

Stick was on its own line of nine digits as it includes x, y, and z data from a magnetometer. A difficulty was that the beginning of the .txt file was not always the data from the pinky. In order to have data read in the order of pinky, ring, middle, index, thumb, and hand manipulations had to be made in the file using MATLAB (Appendix, *readoff.m*). This accomplished finding the hand first as it contained 9 values while the rest had 7 values. Another difficulty was that data would be misread and would provide a line of data that contained non-decimal values. The *readoff.m* MATLAB file would identify problems in the data to obtain a matrix that contained all sensor data in the order as follows:

| | |
|--|--------|
| V A _x A _y A _z G _x G _y G _z | Pinky |
| V A _x A _y A _z G _x G _y G _z | Ring |
| V A _x A _y A _z G _x G _y G _z | Middle |
| V A _x A _y A _z G _x G _y G _z | Index |
| V A _x A _y A _z G _x G _y G _z | Thumb |
| A _x A _y A _z M _x M _y M _z G _x G _y G _z | Hand |

Where \vec{A} is the raw acceleration vector, \vec{G} is the raw gyroscope vector, \vec{M} is the raw magnetometer vector and V is the voltage read from the pressure sensor. Zeros were put in place of the last two values for all finger data to make the matrix have a uniform length of 9.

After sensor data is organized into matrices the data is divided into corresponding finger or hand positions. Finger data contains pressure sensor data as well as motion data and hand data contains only motion data. The next portion of the code converts motion and pressure into forces (Appendix, *allsensors.m*, *mpu6050.m*,

angleconverter.m). We first begin with data from the *Sensor Stick* which includes data from an accelerometer, gyroscope, and magnetometer. The first step is to convert raw data into gravity force (g's). Incoming values are 10 bit numbers and are converted into g's using a similar calibration procedure described earlier (4.G.1.f). The incoming values are divided by 256 as this is equal to 1024 states divided by 4 ($\pm 2g$ range). 256 is what the divisor should be if the accelerometer is perfectly performing but is adjusted using the calibration procedure described earlier (4.G.1.f). The absolute value of the sum of maximum and minimum value found when performing calibration procedures serve as the range of $\pm 1g$. Magnetometer values are 12 bit values and the range of Gauss measured was $\pm 1.3G$. In order to convert raw bit values to Gauss values the range of bit values possible were divided by 2.6 Gauss, this value is 1575.4 which is what all raw values must be divided by. These values are determined using the associated data sheets. Gyroscope conversion is calculated a similar way; values are 16 bit and range from ± 500 °/s so 65536 is divided by 1000. This resulting value of 65.536 is the conversion factor that raw values must be divided by to obtain °/s gyroscope values.

Still using *Sensor Stick* data, the next step was to remove sensor biases that may exist. Over the first 30 time frames (~3 sec) the *Sensor Stick* remained still and the data gathered over these first frames is averaged. These averages can then be subtracted for gyroscope and accelerometer values as gyroscopes should be reading all zeros and accelerometer should read zeros in the x and y components and 1 in the z component. The magnetometer bias was described earlier by using a hard iron calibration procedure (4.G.1.f). After biases were removed the accelerometer and magnetometer were run through a first-order low pass butterworth filter with a cutoff

frequency of half of the sampling rate to remove excess noise. Gyroscope data was smoothed by averaging every two values. Performance metrics that were used later in the algorithm were calculated at this step as calculated in chapter 3, section C, part 1. The change in magnitude from accelerometer data between consecutive time steps is compared to the chosen performance metric Δ_a , in our case was .02 g. If this performance metric was surpassed, then accelerometer data would be weighted by half. This same procedure was performed for magnetometer data and in our case Δ_m was .02 G. These values were chosen by viewing previous work as well adjusting these values to improve algorithm performance.

Magnetometer and accelerometer data were then normalized prior to performing data fusion algorithms. We use a vector based complementary filter to combine data from all sensor data along with a direction cosine matrix (DCM) filter [27, 28, 45, 46]. Our group successfully used an extended Kalman filter for data fusion using simulated data but this algorithm was not successfully integrated into the current glove system (Appendix, A.3). The data fusion techniques are described in chapter 3, section C, part 1 and a flowchart outlining its function is shown in Figure 3.3. Data from the accelerometer is used to estimate the gravity vector measured. The vector is then updated using the gyroscope to determine the amount the angle has changed by combining the current angle between x and z axes by its corresponding gyroscope value of the rotation around the y-axis [27, 28, 45, 46]. This is repeated for the y and z projection meaning the angle around the x-axis [27, 28, 45, 46]. A resultant vector is calculated using the progressed angles. A new gravity vector is found by finding a weighted average between the current accelerometer reading and the calculated vector

from angle progression [27, 28, 45, 46]. Weighting of the gyroscope can be changed in order to improve reading accuracy. The final step of the vector-based complementary filter is to normalize all vector components [27, 28, 45, 46].

Roll, pitch, and yaw are calculated by the following:

$$\varphi = -\text{atan2}\left(\sqrt{G_{Fy}^2 + G_{Fz}^2}, G_{Fx}^2\right)$$

$$\theta = -\text{atan2}\left(\sqrt{G_{Fx}^2 + G_{Fz}^2}, G_{Fy}^2\right)$$

$$\psi = \text{atan}\left(\frac{C_{My}}{C_{Mx}}\right), C_{Mx} = (M_x * \cos \theta) + (M_y * \sin \varphi * \sin \theta) + (M_z * \cos \varphi * \sin \theta)$$

$$C_{My} = (M_y * \cos \varphi) - (M_z * \sin \varphi)$$

where φ is roll, θ is pitch, ψ is yaw, \vec{G}_F is the calculated gravity resultant vector, and \vec{M} is the magnetometer data [47]. Euler angles (φ , θ , ψ) are calculated at every time step and when these are at 75°-105° a DCM complimentary filter is used to calculate them (3.C.1). The vector-based complimentary filter provides inaccurate results occasionally around 90° because gimbal lock occurs and a degree of freedom is lost. The DCM complimentary filter avoids gimbal lock issues but has other numerical issues of its own of singularities at certain values [27, 28, 45-47].

The switching between DCM and vector-based complementary filter provides a way to avoid gimbal lock issues and presents a way to compute Euler angles accurately from accelerometer, gyroscope, and magnetometer data [27, 28, 45-47]. Euler angles calculated from the vector have a rotation sequence of 3-2-1 (ψ - θ - φ) and the rotation sequence for DCM is 1-2-3 (φ - θ - ψ) [27, 28, 45-47]. Euler angles calculated from sensor data should begin at 0° and so all angles are initialized by subtracting the average value

of the first 100 time steps. This 10 second time lapse is considered a calibration procedure for the glove and is performed every test to ensure angles are initialized.

After obtaining Euler angles from the *Sensor Stick* data, *MPU6050* data is used in the next section of data analysis (Appendix, *mpu6050.m*). The same process is performed for *MPU6050* data as it was previously, however, sensor parameters are different for the accelerometer in the *MPU6050* sensor. Accelerometer values are 16 bit and their range is $\pm 2g$. Conversion to g values was achieved by following the calibration procedure in 4.G.1.f to determine the range of bit values achieved and divide this by the 2g range the accelerometer experienced. After accelerometer and gyroscope data was converted from bit values to usable data, the process was repeated as it was with *Sensor Stick* data. However, because the *MPU6050* sensors lack a magnetometer, yaw values were calculated by adding a specific offset for each finger (Appendix, offsets). Offsets were determined by performing a 90° yaw rotation and comparing the results of the electromagnetic sensor yaw values compared to the hand yaw values.

Euler angles are filtered through another butterworth first-order low pass filter to remove noisy data that still may exist. The MATLAB program calculates Euler angles for the thumb first and use thumb pressure sensor data to determine a force vector. Next, the program moves to the index finger, then middle, then ring and finally pinky. Hand data does not have a correlated pressure sensor and is used only to determine yaw values for each fingertip. Pressure sensor data is an analog signal but is already converted to a voltage reading from the Arduino script. Using calibration curves identified for each pressure sensor, the value of force in newtons is calculated in the program (4.G.1.f). This provides force data magnitude, but direction of each component

needs to be performed as well. Components of each force vector are determined by using the resultant gravity vector components and taking each component and dividing it by the resultant gravity vector magnitude:

$$[F_x, F_y, F_z] = \left[\frac{G_{Fx}}{\|\vec{G}_F\|} * f, \quad \frac{G_{Fy}}{\|\vec{G}_F\|} * f, \quad \frac{G_{Fz}}{\|\vec{G}_F\|} * f \right]$$

where f is the magnitude of force vector found from each pressure sensor, \vec{G} is the resultant gravity vector, and $[F_x, F_y, F_z]$ are the components of force.

After calculating every force component for each finger, the components are summed in each direction to have a total force vector that the glove user is applying. Moment components were found by using the distance from the center of the Styrofoam sphere where force was applied to the center of the force plate used and this was then multiplied by the corresponding force component that created a bending moment.

$M_{o_y} = F_z * 0.38815$ and $M_{o_z} = F_y * 0.38815$ were the calculations made to determine bending moments in the y direction M_{o_y} and in the z direction M_{o_z} . The 0.38815 m corresponds to the moment arm used in the validation experiments. A bending moment in the x direction was not able to be calculated, as the moment would be too complex to resolve as it would require a twisting of the Styrofoam ball (Fig. 4.21).

After converting sensor data to moment and force components, experiments were performed to assess accuracy of motion and force calculations. The first experiment performed was assessing the motion sensing capabilities of the glove. Each test began with a 10 second rest and then a movement of the hand to a different position totaling 35 seconds in testing. In order to assess accuracy, it was necessary for

each test to be separated into three distinct time phases; the primary resting phase, transient phase, and a final resting phase. Primary phase consisted of the time between 1-9 seconds, the transient phase was during the 12-20 second time slot, and the final resting phase was between the 24-32 second time slot (Fig. 4.22). A sample test run was performed to determine where these divisions should be.

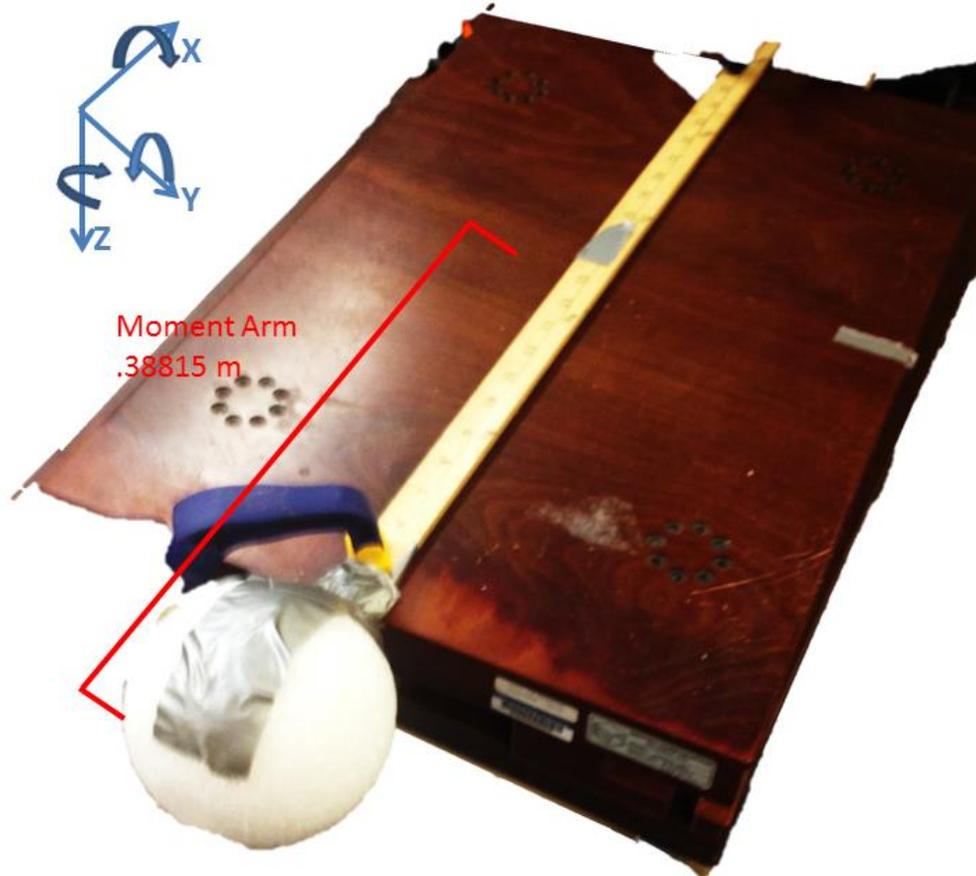


Fig. 4.21 The force testing apparatus is shown with the coordinate plane. Moments are shown in the coordinate plane. One can see that a twisting of the Styrofoam would be how a moment around the x axis is induced.

A MATLAB script was developed to compare electromagnetic sensor data to motion data obtained from the glove. Electromagnetic sensor data was divided into three time phases to align with time phases of the glove data. For each test, a MATLAB script compared data from each sensor; pinky, ring, middle, index, thumb, and hand. For each of the six sensors, the three Euler angles calculated were compared to those of the electromagnetic sensors using root-mean-square error (RMSE) and the average error (AVGE) between the two values. RMSE and AVGE were calculated by the following:

$$RMSE = \sqrt{\frac{\sum_{t=1}^n (em(t) - glove(t))^2}{n}}$$

$$AVGE = \frac{\sum_{t=1}^n (em(t) - glove(t))}{n}$$

where n is the number of time points in the time phase being compared, $glove$ is the Euler angle computed from glove data for the time phase being compared, and em is the Euler angle from electromagnetic sensor data for the time phase being compared. RMSE, AVGE, and the average angle for each time phase of the electromagnetic sensor are reported for each Euler angle and sensor used. There are 54 RMSE, AVGE, and angles calculated for every motion test (Appendix GLOVEtesting.m).

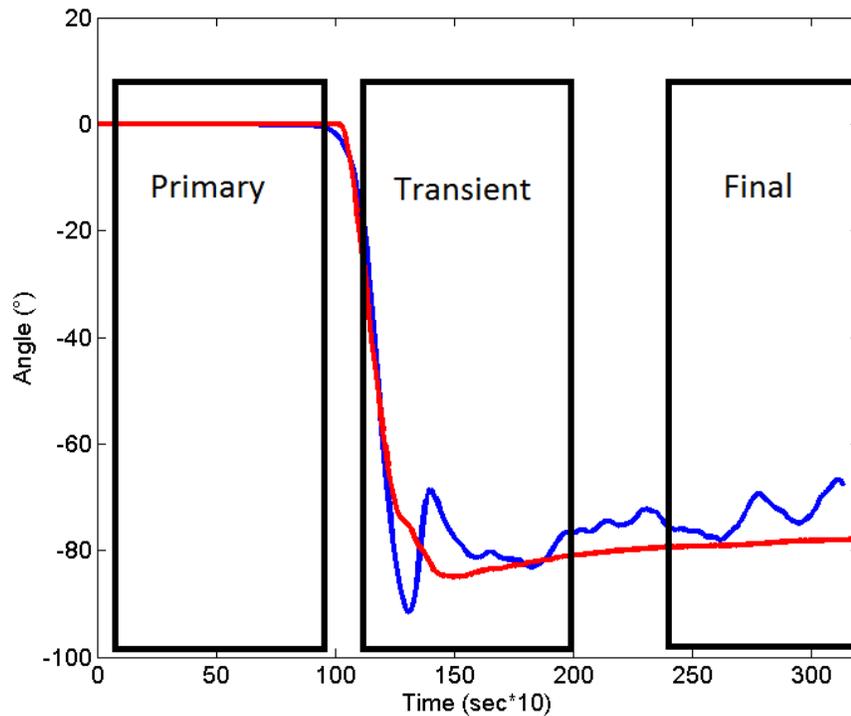


Fig. 4.22 Roll angle is shown above in a motion test experiment. Three separate time phases are shown above; primary from 1-9 seconds, transient from 12-20 seconds, and the final phase from 24-32 seconds. The roll angle calculated from the glove is in blue and the red line displays the roll calculated from electromagnetic sensor.

A separate MATLAB script was developed to combine the results of each test into a single variable to perform statistical tests on the resulting parameters (Appendix, GLOVEtesting.m). Angles found from the glove and the electromagnetic sensors were compared using a linear regression test function in MATLAB that output the equation of the line, a p -value, as well as an R^2 value. RMSE and AVGE values were divided into “rest” and “transient” time phases, the primary and final resting phases being considered the resting phases. We compared the results of both of these statistically using a t-test of means and found a significant difference in RMSE and AVGE values between “resting” and “transient” time phases. We also compared RMSE and AVGE

values between Euler angles as well as between sensors. One-way ANOVA tests were performed to determine significant differences between any groups.

Force testing was a secondary experimental step and required another script to be developed (Appendix, *ForceTesting.m*). Data obtained from the force plate is compared to force data collected from the glove. The primary step is converting force plate raw data into newtons. The first step in conversion is to divide the data received from channels by their respective gain which was manually chosen prior to force experiments. The next step was to multiply the six channels of the force plate; F_x , F_y , F_z , M_{o_x} , M_{o_y} , and M_{o_z} by a scale factor matrix given by the manufacturer (*Bertec*). Force plate data was then converted into the same data coordinate plane that the glove data was output in to ensure proper coordinates were being compared. Force data experiments followed a slightly different protocol than motion data experiments. For the first 10 seconds the glove was at rest and then turned to a different orientation and force was applied in a normal direction to the glove for 20 seconds. Force and moment components for force plate data and glove data were calculated over the 22-28 second time period during testing. Force components calculated from the glove are ordered from greatest magnitude to smallest and forces with average force magnitudes less than 1N were removed due to inaccurate sensor reading as this value corresponded to pressure sensors not experiencing force.

A primary force component was determined as the force component from the glove data with the largest magnitude. Primary moment was the moment component with the greatest magnitude obtained from glove data. Primary and secondary force components from the glove were compared to the corresponding components of the

force plate data. Primary moment data was compared with force plate data of the same direction. RMSE and AVGE values were determined between force plate data and glove data for the primary and secondary force components as well as the primary force component. A linear regression test was performed between force components calculated found from the glove and from the force plate using MATLAB functions. This linear regression test was performed for primary and secondary force components as well as the primary moment component.

In summary, many MATLAB scripts were used to assess motion and force sensing capabilities of the glove. MATLAB was used for converting data into meaningful values, fusion of sensor data, organizing data obtained from experiments, as well as statistical analysis. A flowchart below describes the functions used by MATLAB to perform all of these aforementioned tasks (Fig. 4.23). `Allsensors.m` accesses several of the other scripts and uses the raw sensor data to calculate force and moment vectors (Appendix). `Motion Testing.m` and `Force Testing.m` scripts use data collected during experiments to compare with the results of the calculated force and moment vectors from the glove. Statistical testing is performed by using simple built in MATLAB linear regression and *t-test* functions. One-way ANOVA tests were also used in MATLAB and *p-values* were determined using Bonferroni's correction. MATLAB software was used to analyze data while Arduino and LabVIEW software were used to obtain sensor data. MATLAB also can be used to display graphs and figures to observe the resulting data from the glove.

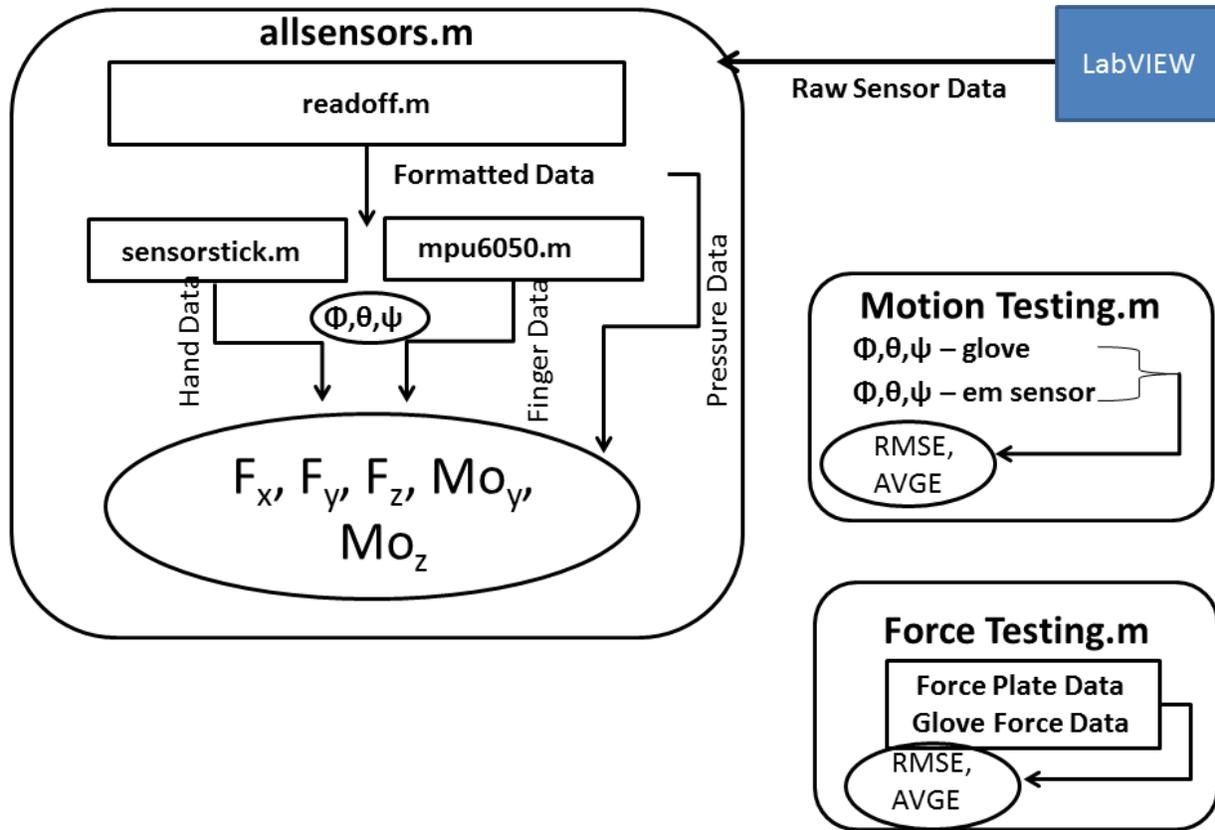


Fig. 4.23 `allsensors.m` is a script that accesses several other scripts: `readoff.m`, `sensorstick.m`, and `mpu6050.m`. Data is read into MATLAB via LabVIEW and then formatted using `readoff.m`. Data then proceeds to `mpu6050.m` if it is finger motion data or to `sensorstick.m` if it is data read from the *Sensor Stick* IMU. Euler angles are calculated and then pressure data is integrated to obtain F_x , F_y , F_z , Mo_y , and Mo_z . `Force Testing.m` compares these to force plate data and obtains RMSE and AVGE values. `Motion Testing.m` compares Euler angles to electromagnetic sensor data and obtains RMSE and AVGE values.

1.3 Final Design Specifications

- ✓ Capture forces ranging from 0-100 Newtons
- ✓ Minimal profile (thickness/height) $< .01m$
- ✓ Euler angle calculation error $< 10^\circ$
- ✓ Total Force calculation error $< 5\%$
- _ Data output $> 30 Hz$
- ✓ Delay between sensors $< .05 sec$

Increasing data frequency was not as important when assessing motion and force sensing accuracy. Future design iterations that require real-time force and moment calculations will need to improve this output speed as it is currently at 10 *Hz*. Using more elegant algorithms may improve the speed of the glove output. This design specification was not the primary concern when developing the glove, as data was to be analyzed post recording. Improvements may be also made in the accuracy of force and moment calculations as future designs may use the EKF designed for sensor data fusion (Appendix A.3).

Chapter 5: Conclusion

Our research group was able to develop and validate force and moment sensing gloves. A glove was developed using an Arduino microcontroller, six accelerometers, six gyroscopes, a magnetometer, and five pressure sensors. Using a data fusion algorithm our group was able to calculate orientation as well as force and moment vectors applied by the user of the developed glove. Motion capturing capabilities of the glove were validated by comparing the results of motion orientation to an external motion sensor system. Force and bending moment calculations from the glove were compared to data acquired from a force plate. The developed glove was capable of calculating Euler angle rotations within 16° in any direction. The glove is capable of measuring the bending moment component (M_{o1}) and force component with the largest magnitude (F_1), but only F_1 accurately (5.85% error). F_1 retains this accuracy in any direction the force is applied.

The secondary and tertiary components, in terms of magnitude, of force and moment vectors have larger errors than the primary components and cannot be considered reliable. Though the errors of primary moment component is quite large (58.78%), the linear regression tests for primary force and moment components provide additional evidence that the developed glove captures forces and moments related to what is truly being applied by the user. A strong linear trend exists between calculated force and moment and the corresponding values of force and moments observed by a force plate.

Our group is primarily concerned with force and moment sensing capabilities. The errors from Euler angles were small enough that primary force and moment components are able to be captured by the device. However, orientation calculation does have an impact on force and moment vector calculation accuracy. Improvements in the data fusion algorithm could be improved to a more sophisticated design such as the Kalman filter (Appendix). A quaternion-based Kalman filter would avoid Gimbal lock problems that may occur and improve angle determination accuracy. Yaw error values did not significantly differ among sensors on the fingers and hand. This proves that the offset values calculated to determine yaw angle among fingers was sufficient in determining yaw angle accurately; however, additional magnetometers may improve accuracy even further.

Errors found in the validation experiments were caused by a variety of factors. One is the data fusion algorithm, another is the use of only one magnetometer, and the last error was due to the experimental set up itself. Placement of IMU and electromagnetic sensors presented a challenge that could provide inconsistent results. Though the electromagnetic sensors were placed on top of IMU sensors, placement difference may vary from one set of sensors to the next, which is from one finger to the next. Placement of one sensor atop the other also induces errors because a rotation of the hand may rotate one sensor more-so than the other sensor. These errors were controlled by including offsets for each IMU sensor; however, these errors could differ in more complex hand rotation and gestures. A simpler experiment could have preceded this experiment by adhering one IMU and one electromagnetic sensor next to one another to a thin plane of wood. An apparatus would control the rotation of the wood

and these would provide more precise offsets and would provide more insight to innate differences among the two sensor systems: the IMU and electromagnetic sensors. This apparatus could also reduce the impact of human error that was a factor in errors experienced.

Force and bending moment vectors were able to be calculated from the developed glove device. Force and vectors were accurate only in the primary component of magnitude due to errors in the other two components for each vector. Large errors in experimental testing were due to a variety of factors. Forces applied outside of the fingertip sensors were a large source of error and were aimed to be corrected by scale factors multiplied by the glove forces calculated. On several more complex hand orientations, forces outside of the fingertips could vary. Errors from pressure sensor calibration would have a large impact on force and moment calculations. Pressure sensor calibration curves followed a cubic fit combined with a linear fit. These calibration curves would make more sense as a purely linear fit. Using more weights in more consistent increments in the calibration procedure could have improved this fit. We were limited to a set of five weights, as the weights had to be small enough to be placed on a puck placed on top of the pressure sensor.

An improvement on calibration as well as the use of a simple experiment to assess force sensing could reduce errors in force and moment vector calculations. Pressure sensors could be placed on a force plate and a constant force is applied to determine whether the force calculated from the sensor matched up with the reading from the force plate data. This assessment could be used to improve force calibration curves and improve the algorithm that determines force vectors calculated by the glove.

Another improvement would be to increase the loading resistor in the voltage divider circuit used with the variable resistor pressure sensor; this would increase the sensitivity of the pressure sensor. Our group chose the loading resistor to be 5 K Ω to have a sensing range of 0-133 N as several groups have found forces greater than 100 N applied to an infant head [22, 24]. In our experimental setup forces greater than 100 N seemed improbable to be applied by traction force alone during delivery of an infant as we observed a magnitude of 30N at our highest value.

Though we were able to compute the primary force component accurately, there are limitations to the capabilities of the device. Force and moment vectors are not currently able to be calculated completely accurate. Suggested improvements to experiments and the used algorithms may change these capabilities. A limitation to the device is that the force vector is calculated by determining the normal direction of the IMU. Forces in non-normal directions will not be captured. Forces applied outside of the sensors will not be able to be measured by the device. There is also an assumption when calculating the moment vector that all force is applied at one point on the infant head. Improvements in the algorithm could make an estimate of distances between each finger to improve this assumption. The algorithm could also make a bending moment in the *x-direction* of our apparatus, or a twisting moment applied by the user possible. This was another limitation of the current device. Use of only one magnetometer in the glove device prevents certain hand orientations like finger spreading to be captured accurately. This does not seem to create a large problem in the application of our device.

Though there are limitations to our device these findings provide our research group with a tool that can be further improved to be used in SD and routine vaginal delivery of infants to determine clinician-applied forces. Improvements of this device will allow the device to be used in mock deliveries and live deliveries [1, 9]. This device has an advantage over other developed tools to assess clinician-applied forces as it provides directional information that is important when determining brachial plexus stretch [5, 23]. The device is significant as it can be used as a tool to quantify clinician applied forces and moments and can be used as a tool to quantitatively assess birthing simulation training [10]. Quantitative data provides another metric to measure training course effectiveness that has not been explored [10].

To use the developed glove in practice there must be some alterations in electrical design as well as algorithm design. Additional pressure sensors as well as shear sensor forces could be used to capture more forces experienced by the glove wearer, including forces that are not in the normal direction of the IMUs. Inclusion of an infrared proximity sensor on the back of the middle finger could also improve moment arm distance calculation. Using a Kalman filter could improve the orientation calculation algorithm as well. The next step would be to improve the current glove design and develop a printed circuit board (PCB) using the developed electrical schematic. A PCB would allow the device to be integrated onto a Lycra glove material. The glove would also need to communicate via Bluetooth to be used in practice. This glove could then be used to examine forces and moments experienced in PROMPT simulation training courses. Ultimately, the glove could then be used to measure forces and moments in a live delivery providing no complications during birthing simulations occurred.

References

1. Schwartz, C., Kieweg, S.L., Weiner, C.P., Wilson, S.E. , *Obstetrician Hand Pressures During Mock Deliveries*, in *ASME 2013 Summer Bioengineering Conference*. 2013.
2. Grimm, M.J., R.E. Costello, and B. Gonik, *Effect of clinician-applied maneuvers on brachial plexus stretch during a shoulder dystocia event: investigation using a computer simulation model*. *Am J Obstet Gynecol*, 2010. **203**(4): p. 339 e1-5.
3. Gurewitsch, E.D. and R.H. Allen, *Reducing the risk of shoulder dystocia and associated brachial plexus injury*. *Obstet Gynecol Clin North Am*, 2011. **38**(2): p. 247-69, x.
4. Piatt, J.H., Jr., *Birth injuries of the brachial plexus*. *Clin Perinatol*, 2005. **32**(1): p. 39-59, v-vi.
5. Gonik, B., N. Zhang, and M.J. Grimm, *Prediction of brachial plexus stretching during shoulder dystocia using a computer simulation model*. *Am J Obstet Gynecol*, 2003. **189**(4): p. 1168-72.
6. Marques, J.B. and A. Reynolds, *[Shoulder dystocia: an obstetrical emergency]*. *Acta Med Port*, 2011. **24**(4): p. 613-20.
7. Draycott, T.J., et al., *Improving neonatal outcome through practical shoulder dystocia training*. *Obstetrics and Gynecology*, 2008. **112**(1): p. 14-20.
8. Allen, R.H., et al., *Comparing Clinician-Applied Loads for Routine, Difficult, and Shoulder Dystocia Deliveries*. *American Journal of Obstetrics and Gynecology*, 1994. **171**(6): p. 1621-1627.
9. Kieweg, S.L., Wilson, S.E., Markovich, G., Manamendra, H.I., Weiner, C.P. . *Biomechanics of Birth – The Fallacy of Gentle Birth: Physician Exerted Pressures in Vaginal and Cesarean Delivery*. in *IFMBE 2010*.
10. Weiner, C.P., Samuelson, L., Collins, L., and Satterwhite, C. , *5-Year Experience with PROMPT (PRactical Obstetric Multidisciplinary Training) Reveals Sustained and Progressive Improvements in Obstetric Outcomes at a US Hospital*, in *The Pregnancy Meeting*. 2013, Society for Maternal-Fetal Medicine: New Orleans, LA.
11. Allen, R., J. Sorab, and B. Gonik, *Risk factors for shoulder dystocia: an engineering study of clinician-applied forces*. *Obstet Gynecol*, 1991. **77**(3): p. 352-5.
12. Cavanaugh, J.M. *Biomechanics of Nerve Injury*. in *Neonatal Brachial Plexus Palsy Conference*. 2014. Wayne State University, Detroit, MI.
13. Borschel, G. *Brachial Plexus Anatomy 101: A Guided Tour*. in *Neonatal Brachial Plexus Palsy Conference*. 2014. Wayne State University, Detroit, MI.
14. Singh, A., Lu, Y., Chen, C., Cavanaugh, J.M. , *Mechanical properties of spinal nerve roots subjected to tension at different strain rates*. *J. Biomechanics*, 2006. **39**: p. 1669-76.
15. Gherman, R.B., et al., *Shoulder dystocia: the unpreventable obstetric emergency with empiric management guidelines*. *Am J Obstet Gynecol*, 2006. **195**(3): p. 657-72.
16. Baskett, T.F., *Shoulder dystocia*. *Best Pract Res Clin Obstet Gynaecol*, 2002. **16**(1): p. 57-68.
17. Baxley, E.G. and R.W. Gobbo, *Shoulder dystocia*. *Am Fam Physician*, 2004. **69**(7): p. 1707-14.
18. Gottlieb, A.G. and H.L. Galan, *Shoulder dystocia: an update*. *Obstet Gynecol Clin North Am*, 2007. **34**(3): p. 501-31, xii.
19. Athukorala, C., et al., *Women with gestational diabetes mellitus in the ACHOIS trial: risk factors for shoulder dystocia*. *Aust N Z J Obstet Gynaecol*, 2007. **47**(1): p. 37-41.
20. Dandolu, V., et al., *Trends in the rate of shoulder dystocia over two decades*. *J Matern Fetal Neonatal Med*, 2005. **18**(5): p. 305-10.
21. Stitely, M.L. and R.B. Gherman, *Shoulder dystocia: management and documentation*. *Semin Perinatol*, 2014. **38**(4): p. 194-200.
22. Sorab, J., R.H. Allen, and B. Gonik, *Tactile sensory monitoring of clinician-applied forces during delivery of newborns*. *IEEE Trans Biomed Eng*, 1988. **35**(12): p. 1090-3.

23. Allen, R.H., *On the mechanical aspects of shoulder dystocia and birth injury*. Clin Obstet Gynecol, 2007. **50**(3): p. 607-23.
24. Crofts, J.F., et al., *Pattern and degree of forces applied during simulation of shoulder dystocia*. Am J Obstet Gynecol, 2007. **197**(2): p. 156 e1-6.
25. Hernandez-Rebollar, J.L., Kyriakopoulos, N., Lindeman, R.W. , *The AcceleGlove: A whole-hand input device for virtual reality, (Technical Sketch)*. ACM SIGGRAPH 2002, 2002: p. 259.
26. Saggio, G., Lagati, A., Orengo, G. , *Wireless sensory glove system developed for advanced human computer interface* International Journal of Information Science, 2012. **2**(5): p. 54-59.
27. Sabatini, A.M., *Quaternion-based extended Kalman filter for determining orientation by inertial and magnetic sensing*. IEEE Trans Biomed Eng, 2006. **53**(7): p. 1346-56.
28. Sabatini, A.M., *Estimating three-dimensional orientation of human body parts by inertial/magnetic sensing*. Sensors (Basel), 2011. **11**(2): p. 1489-525.
29. Tian, Y., H. Wei, and J. Tan, *An adaptive-gain complementary filter for real-time human motion tracking with MARG sensors in free-living environments*. IEEE Trans Neural Syst Rehabil Eng, 2013. **21**(2): p. 254-64.
30. Yoo, T.S., et al., *Gain-scheduled complementary filter design for a MEMS based attitude and heading reference system*. Sensors (Basel), 2011. **11**(4): p. 3816-30.
31. American College of Obstetrics and Gynecologists, *ACOG Practice Bulletin. Clinical Management Guidelines for Obstetrician-Gynecologists: number 41, December 2002*. Obstet Gynecol, 2002. **100**(6): p. 1389-402.
32. Inglis, S.R., et al., *Effects of shoulder dystocia training on the incidence of brachial plexus injury*. Am J Obstet Gynecol, 2011. **204**(4): p. 322 e1-6.
33. Mitrani, I., *Simulation Techniques for Discrete Event Systems*. 1982, Cambridge, UK: Cambridge University Press.
34. Metaizeau, J.P., C. Gayet, and F. Plenat, *[Brachial plexus birth injuries. An experimental study (author's transl)]*. Chir Pediatr, 1979. **20**(3): p. 159-63.
35. Mollberg, M., et al., *Comparison in Obstetric Management on Infants With Transient and Persistent Obstetric Brachial Plexus Palsy*. Journal of Child Neurology, 2008. **23**(12): p. 1424-1432.
36. Kramer, J., *Force feedback and textures simulating interface device*. Patent US5184319 A. 1993.
37. Kramer, J., Yim, M., Tremblay, M., Gomez, D., *Force-feedback interface device for the hand*. Patent US6413229 B1. 2002.
38. Land, E., Mantelmacher, H., *Low profile hand-extension/flexion device*. Patent US8348810 B2. 2013.
39. Yazadi, F., Schelbert, M., Miller, L., *Motion capture data glove*. Patent 20120025945. 2002.
40. Sivak, M., Holden, M., Mavroidis, C., Bajpai, A., Bintz, C., Chrisos, J., Clark, A., Lentz, D. , *Multi-user smartglove for virtual environment-based rehabilitation*. Patent EP2389152 A1. 2011.
41. Bergelin, B., Ihrke, C., Davis, D., Linn, D., Sanders, A., Askew, R., Laske, E., Ensley, K., *Control of a glove-based grasp assist device*. Patent US20130226350 A1. 2013.
42. Sun, Y., *Fingertip force, location, and orientation sensor*. Patent US8724861 B1. 2014.
43. Wang, W.C., Nuckley, D., Reinhall, P., Linders, D. , *Clinical force sensing glove*. Patent US20110302694 A1. 2011.
44. Montgomery, E., *Apparatuses and methods for evaluating a patient*. Patent US20130197399 A1. 2013.
45. Gebre-Egziabher, D., Hayward, R.C., Powell, J.D., *The Design of multi-sensor attitude determination systems*. IEEE Transactions on Aerospace and Electronic Systems, 2004. **40**(2): p. 627-649.

46. Euston, M., Coote, P., Mahony, R., Kim, J., Hamel, T. , *A Complementary Filter for Attitude Estimation of a Fixed-Wing UAV*, in *Intelligent Robots and Systems*. 2008. p. 340-345.
47. JKuipers, J., *Quaternions and Rotation Sequences*. 1999, Princeton, NJ: Princeton University Press. 363.
48. Caruso, M.J., *Applications of magnetic sensors for low cost compass systems*. IEEE 2000, 2000: p. 177-184.
49. Gallo, K., Oughstun, K., *Error Analysis of 3-D Electromagnetic Trackers*. Journal of Latex Class Files 2007. **6**(1): p. 1-6.
50. Gherman, R.B., J.G. Ouzounian, and T.M. Goodwin, *Obstetric maneuvers for shoulder dystocia and associated fetal morbidity*. Am J Obstet Gynecol, 1998. **178**(6): p. 1126-30.
51. Santosh Kumar, B., Ravi Chandra, L., Aditya, A. L. G. N., Basha, F. N., Praveen Blessington, T. , , *Design and Functional Verification of I2C Master Core using OVM*. International Journal of Soft Computing and Engineering, 2012. **2**(2): p. 528-533.
52. Larson, A. and D.E. Mandelbaum, *Association of head circumference and shoulder dystocia in macrosomic neonates*. Matern Child Health J, 2013. **17**(3): p. 501-4.

Appendix Table of Contents

| | |
|--|-----|
| A.Code Used to Acquire and Analyze Code..... | 123 |
| A.1 Arduino Code | 123 |
| A.2 MATLAB Files | 126 |
| a) readoff.m | 126 |
| b) angleconverter.m | 128 |
| c) mpu6050.m | 132 |
| d) allsensors.m | 138 |
| e) MotionTesting.m..... | 143 |
| f) GLOVEtesting.m | 155 |
| g) ForceTesting.m | 163 |
| h) FORCEHANDtesting.m | 166 |
| A.3 Potential Improvements to Code..... | 166 |
| a) Roll Method | 167 |
| b) kalmanfilterquaternion.m | 168 |
| B. Preliminary Calculations | 172 |
| B.1. Euler Angle Offsets | 172 |
| B.2. Force Scale Factors | 173 |
| C. Example Data Output | 174 |

Appendix

A. Code Used to Acquire and Analyze Data

A.1 Arduino Code

INITIALIZING MPU6050 and reading from it

```
accel_i2c_68.initialize();
void MPU6050::initialize() {
    setClockSource(MPU6050_CLOCK_PLL_XGYRO); //sets clock gyrobased
    setFullScaleGyroRange(MPU6050_GYRO_FS_500); // setting gyro
range( $\pm 500^\circ/s$ )
    setFullScaleAccelRange(MPU6050_ACCEL_FS_2); // setting accel
range( $\pm 2g$ )
    setSleepEnabled(false); //turns off sleep bit
}
void MPU6050::getMotion6(int16_t* ax, int16_t* ay, int16_t* az,
int16_t* gx, int16_t* gy, int16_t* gz) {
    I2Cdev::readBytes(devAddr, MPU6050_RA_ACCEL_XOUT_H, 14, buffer);
    *ax = (((int16_t)buffer[0]) << 8) | buffer[1]; //access 16 bit val
    *ay = (((int16_t)buffer[2]) << 8) | buffer[3];
    *az = (((int16_t)buffer[4]) << 8) | buffer[5];
    *gx = (((int16_t)buffer[8]) << 8) | buffer[9];
    *gy = (((int16_t)buffer[10]) << 8) | buffer[11];
    *gz = (((int16_t)buffer[12]) << 8) | buffer[13];
}
```

INITIALIZING SENSORSTICK

```
void init_adxl345() {
    byte data = 0; //initializing data

    i2c_write(ADXL345_ADDRESS, ADXL_REGISTER_PWRCTL,
ADXL_PWRCTL_MEASURE); //access accel setting into measure mode

    i2c_read(ADXL345_ADDRESS, ADXL_REGISTER_PWRCTL, 1, &data); //check to
see if worked
    Serial.println((unsigned int)data); // how data will be presented
}
void init_itg3200() {
    byte data = 0; //initializing data

    i2c_write(ITG3200_ADDRESS, ITG3200_REGISTER_DLPF_FS,
ITG3200_FULLSCALE | ITG3200_42HZ); //access gyro setting measure mode
```

```

    i2c_read(ITG3200_ADDRESS, ITG3200_REGISTER_DLPF_FS, 1,
&data); //check to see if worked

    Serial.println((unsigned int)data); //data format
}
void init_hmc5843() {
    byte data = 0;

    i2c_write(HMC5843_ADDRESS, HMC5843_REGISTER_MEASMODE,
HMC5843_MEASMODE_CONT); //access mag setting measure mode

    i2c_read(HMC5843_ADDRESS, HMC5843_REGISTER_MEASMODE, 1,
&data); //check to see if worked
    Serial.println((unsigned int)data); //data format
}

READ FROM SENSORSTICK
void read_adxl345() {
    byte bytes[6];
    memset(bytes,0,6); //clearing variable

    i2c_read(ADXL345_ADDRESS, ADXL345_REGISTER_XLSB, 6, bytes); //reading
data from accel 6 diff registers

    for (int i=0;i<3;++i) {
        accelerometer_data[i] = (int)bytes[2*i] + (((int)bytes[2*i + 1]) <<
8); //read 6 bytes goes lsb for x to msb of x, lsb for y etc.
    }
}
void read_itg3200() {
    byte bytes[6];
    memset(bytes,0,6); //clearing variables

    i2c_read(ITG3200_ADDRESS, ITG3200_REGISTER_XMSB, 6, bytes);
//reading data from gyro 6 diff registers
    for (int i=0;i<3;++i) {
        gyro_data[i] = (int)bytes[2*i + 1] + (((int)bytes[2*i]) << 8);
    } //read 6 bytes
}
void read_hmc5843() {
    byte bytes[6];
    memset(bytes,0,6); //clearing variables

    i2c_read(HMC5843_ADDRESS, HMC5843_REGISTER_XMSB, 6, bytes);
//reading data from mag 6 diff registers
    for (int i=0;i<3;++i) {

```

```

    magnetometer_data[i] = (int)bytes[2*i + 1] + (((int)bytes[2*i]) <<
8); //read 6 bytes of data
}

```

CHOOSING CHANNEL FOR MULTIPLEXER

```

void out2()
{
    digitalWrite(EN, LOW); //pin EN to 0
    digitalWrite(S0, HIGH);
    digitalWrite(S1, HIGH);
    digitalWrite(S2, LOW);
    digitalWrite(S3, LOW); // these write 1100= 3, chooses channel
}

```

SAMPLE LOOP FOR EACH CHANNEL

```

void loop() // main loop of program
{

    for(j == 0; j < 1; j++) //indexing to choose channel 0 then progress
to next loop
    {
        out1(); //channel 0 chosen
        k = 0;
        accel_i2c_68.initialize(); //initialize mpu6050 sensor
        accel_i2c_68.setFullScaleAccelRange(MPU6050_ACCEL_FS_2);
        accel_i2c_68.setFullScaleGyroRange(MPU6050_GYRO_FS_500);
        // Read the input on analog pin 0:
        int sensorValue1 = analogRead(A0); //read analog signal-pressure sen
        float voltage1 = sensorValue1 * (5.0 / 1023.0); // Convert the
analog reading (which goes from 0 - 1023) to a voltage (0 - 5V):
        Serial.print(voltage1); // print voltage
        Serial.print("\t");

        accel_i2c_68.getMotion6(&ax, &ay, &az, &gx, &gy, &gz); //get accel
and gyro values

        Serial.print(ax); //print accel x value
        Serial.print("\t"); //space
        Serial.print(ay);
        Serial.print("\t");
        Serial.print(az);
        Serial.print("\t");
        Serial.print(gx);
        Serial.print("\t");
        Serial.print(gy);
    }
}

```

```

    Serial.print("\t");
    Serial.print(gz);
    Serial.print("\t");
    Serial.print("\n"); //new line
}
SENSORSTICK LOOP
for(o == 0; o < 1; o++) //index to read after every channel read, no
channel chosen to read from sensorstick
{
    j =0;
    read_adxl345();
    //Serial.print("ACCEL: ");
    Serial.print(accelerometer_data[0]);
    Serial.print("\t");
    Serial.print(accelerometer_data[1]);
    Serial.print("\t");
    Serial.print(accelerometer_data[2]);
    Serial.print("\t");

    read_hmc5843();
    //Serial.print("MAG: ");
    Serial.print(magnetometer_data[0]);
    Serial.print("\t");
    Serial.print(magnetometer_data[1]);
    Serial.print("\t");
    Serial.print(magnetometer_data[2]);
    Serial.print("\t");

    read_itg3200();
    //Serial.print("GYRO: ");
    Serial.print(gyro_data[0]);
    Serial.print("\t");
    Serial.print(gyro_data[1]);
    Serial.print("\t");
    Serial.print(gyro_data[2]);
    Serial.print("\n");
    delay(50);
}
}

```

A.2 MATLAB files

a) readoff.m

This program reads off data from the .txt file developed by LabVIEW, it also formats the data into proper matrices for data to be easily accessed.

```
% fid = fopen('2fingerreach_test1'); %open 2fingerreach_test1 file to analyze
```

```

a = textscan(fid,'%f %f %f %f %f %f %f %f %f','Headerlines', 50); %Reads off
data with a 50 line header, fid is file name
endline = length(a{1,1})-1; %skips the very last line
a1 = textscan(fid,'%f %f %f %f %f %f %f %f %f','Headerlines', endline-220);
%If there is a break in the format of txt file this will enable correction as
it uses the endline where the break in the data is and uses it as a header
endline1 = length(a1{1,1})-1;

for k = 1:9
    a{1,k} = a{1,k}(1:endline,1); %This sizes both matrices to have 9 columns
- one column for each data pt (ax,ay,az,gx,gy,gz,mx,my,mz)
    a1{1,k} = a1{1,k}(1:endline1,1);
end

a = cell2mat(a); %Converts to matrix
b = size(a);

a1 = cell2mat(a1);
b1 = size(a1);

for i = 1:length(a)
    if isnan(a(i,8)) == 0 %Detects when the Sensorstick data begins as other
sensors only have 7 numbers (pressure sensor, accel(x,y,z),gyro(x,y,z))
        i = i+1;
        break
    end
end

for j = 1:length(a1)
    if isnan(a1(j,8)) == 0
        j = j+1;
        break
    end
end

C1 = a1;
C1 = a1(j:end,:);
C = a;
C = a(i:end,:); %C is output matrix thumb, index, middle, ring, pinky, back
of hand
lC = length(C);
lC1 = length(C1);
for l = 0:1:10
    if isnan(C(lC-1,9))== 1 %Checking whether sensorstick data is last line
so each sensor has same amt of lines
        l = l + 1; %Check further up from end
    elseif isnan(C(lC-1,9)) == 0 %If its sensorstick as last line stop
        break
    end
end

C = C(1:lC-l-1,:); %Making each sensor have same length

%Same as previous loop for C1
if isempty(C1) == 0;
    for m = 0:1:10
        if isnan(C1(1+m,9)) == 1
            m = m + 1;
        elseif isnan(C1(1+m,9)) == 0

```

```

        break
    end
    end
    end
    C1 = C1(1+m:end,:);
    C = [C; C1]; %If there was a break in data structure this will compensate
for it
end

```

b) angleconverter.m

This script is for sensorstick data. This converts raw sensor data into orientation of the hand.

```

%Sources: http://www.instructables.com/id/Accelerometer-Gyro-
Tutorial/step3/Combining-the-Accelerometer-and-Gyro/
%http://www.starlino.com/wp-
content/uploads/data/dcm_tutorial/Starlino_DCM_Tutorial_01.pdf
last = length(sensorstick); %%length of sensorstick - which is sensorstick
data 9 colums
[B,A] = butter(1,1); %Returns filter coefficients for a 1st order low pass
filter cutoff half sampling rate
dt = .1; %time between samples
%Using sensorstick data accelerometer
testaccel = sensorstick(:,1:3); %accelerometer values from sensorstick first
3 columns
accel_y = -(testaccel(:,1)/253 - .0198); %Converting 10 bit values to +/- 2g,
1g = 256 without offset. y axis read off first xmin = -249, xmax = 257
accel_x = (testaccel(:,2)/259 - .0347); %ymin = -260, ymax = 261
accel_z = (testaccel(:,3)/253 +.0198); %zmin = -258, zmax = 248
accel_x = accel_x - mean(accel_x(1:30)); %removing bias calculated from
accel_y = accel_y - mean(accel_y(1:30)); %first 30 pts while device is still
accel = [accel_x,accel_y,accel_z];
filt_acc = filtfilt(B,A,accel(:,,:)); %Filtering using Low Pass Butterworth
Filter

%Converting sensorstick data magnetometer
testmag = (sensorstick(:,4:6)); %Loading magnetometer values
mag = (testmag(:,)/1575); %Converting 12 bit # to Gauss(±1.3G)
filt_mag = filtfilt(B,A,mag(:,,:)); %Lowpass filter mag data
x_mag = filt_mag(:,1);
z_mag = -filt_mag(:,2); %Negative values shown for z
y_mag = filt_mag(:,3);
filt_mag = [x_mag,y_mag,z_mag];

%Using sensorstick data gyroscope
testgyro = (sensorstick(:,7:9)); %Accessing gyro data
gyro = (testgyro(:,)/65.536); %Converting 16 bit # to +/- 500 deg/sec
y_gyro = gyro(:,1); %y values and x values switched
x_gyro = gyro(:,2);
z_gyro = -gyro(:,3);
gyro = [x_gyro,y_gyro,z_gyro];
    for i = 1:length(gyro)
        gyro(i,:) = gyro(i,:) - mean(gyro(1:50,:)); %removing bias
    end %Demeaning data reduces drift in gyro

%Magnetometer hard iron calibration - shifting centroid to origin was found

```

```

for i = 1:1:length(filt_mag)
    filt_mag(i,:) = filt_mag(i,:) - [.2052, -.2031, 0.7172];
end
sa = zeros(1,length(filt_acc));%weighting vectors
sm = zeros(1,length(filt_mag));
delta_m = .025; %Threshold value for mag
delta_a = .02; %Threshold value for accel
for i = 2:length(filt_mag)
    if abs(norm(filt_mag(i,:)) - norm(filt_mag(i-1,:)))>= delta_m;
        sm(i) = 5; %If surpass threshold decrease weighting of mag
    else
        sm(i) = 10; %otherwise keep it the same
    end
end
end
for i = 2:length(filt_acc)
    if abs(norm(filt_acc(i,:)) - norm(filt_acc(i-1,:)))>= delta_a;
        sa(i) = 5;
    else
        sa(i) = 10;
    end
end
end
%Normalizing vectors
%Normalizing accelerometer data
for i = 1:1:length(filt_acc)
    filt_acc(i,:) = filt_acc(i,:)/norm(filt_acc(i,:)); %Normalize accel at every
time pt.
end

%Normalizing mag data
for i = 1:1:length(filt_mag)
    filt_mag(i,:) = filt_mag(i,:)/norm(filt_mag(i,:));
end

for i = 2:length(gyro)
    GyroRate(i,1)=(gyro(i,1)+gyro(i-1,1))/2 ; %Finding diff in one gyro time
pt to the next to reduce drift
    GyroRate(i,2)=(gyro(i,2)+gyro(i-1,2))/2 ;
    GyroRate(i,3)=(gyro(i,3)+gyro(i-1,3))/2 ;
end
gyro = GyroRate(:,1:3);

pitch = zeros(1,length(accel)); %Initializing Euler angles
roll = zeros(1,length(accel));
yaw = zeros(1,length(accel));
%DCM Calculation
Kb = -filt_acc; %Vector for accelerometer data - aligned with zenith
Ib = filt_mag; %Magnetometer vector aimed at north
% sa = 10; %This sets the weighting of the accelerometer can be changed with
performance metric
sg = 10;
% sm = 10;
DCM = zeros(1:3,1:3,1:length(filt_acc)); %Initializing DCM
for i = 2:1:length(filt_acc)-1
    Jb(i,:) = cross(Kb(i,:),Ib(i,:)); %Vector that is orthogonal to both vectors
    DCM(1:3,1:3,i) = [Ib(1,:); Jb(1,:); Kb(1,:)];
    d_thetag(i,:) = dt*gyro(i,:);

```

```

Kbg(i,:) = Kb(i,:) + (cross(d_thetag(i,:),Kb(1,:)));
d_thetaa(i,:) = cross(Kb(i,:),Kb(i+1:)-Kb(1,:));
d_thetam(i,:) = cross(Ib(i,:),Ib(i+1:)-Ib(1,:));
d_theta(i,:) = (sa(i)*d_thetaa(i,:) + sg*d_thetag(i,:) +
sm(i)*d_thetam(i,:))/(sa(i)+sg+sm(i));%This is where weighting comes into
play making angle progression
Kb(i+1,:) = Kb(i,:)+ cross(d_theta(i,:),Kb(1,:)); %Update each vector
Ib(i+1,:) = Ib(i,:)+ cross(d_theta(i,:),Ib(1,:));
Jb(i+1,:) = Jb(i,:)+ cross(d_theta(i,:),Jb(1,:));
Err(i) = (dot(Ib(i+1,:),Jb(i+1,:))); %Ensure all vectors are norm and orthog
Ib(i+1,:) = Ib(i+1,:) - Err(i)*Ib(i+1,:);
Jb(i+1,:) = Jb(i+1,:) - Err(i)*Jb(i+1,:);
Ib(i+1,:) = Ib(i+1,:)/norm(Ib(i+1,:)); %Normalize
Jb(i+1,:) = Jb(i+1,:)/norm(Jb(i+1,:));
Kb(i+1,:) = cross(Ib(i+1,:),Jb(i+1,:));
Kb(i+1,:) = Kb(i+1,:)/norm(Kb(i+1,:)); %Normalize
DCM(1:3,1:3,i) = [Ib(i,:); Jb(i,:); Kb(i,:)];
rollDCM(1,i) = -atan2(DCM(3,1,i),DCM(1,1,i))*180/pi; %Calculated Euler angles
from DCM
pitchDCM(1,i) = -asin(DCM(1,1,i))*180/pi;
yawDCM(1,i) = -atan2(DCM(2,1,i),DCM(1,1,i))*180/pi;

end
%%%%%%R VECTOR METHOD
%Combining accelerometer and gyro data
R_est = filt_acc; %Assume gravity vector is accel value
R = zeros(length(accel),1); %Initialize all
gyro_avg = zeros(length(accel),3);
Axz = zeros(length(accel),1);
Ayz = zeros(length(accel),1);
Rx_gyro = zeros(length(accel),1);
Ry_gyro = zeros(length(accel),1);
Rz_gyro = zeros(length(accel),1);
testgyro = (sensorstick(:,7:9));
gyro = (testgyro(:,:)/65.536); %Converting 16 bit # to +/- 500 deg/sec
y_gyro = gyro(:,1);
x_gyro = gyro(:,2);
z_gyro = -gyro(:,3);
gyro = [x_gyro,y_gyro,z_gyro];
for i = 1:1:length(gyro)
    gyro(i,:) = gyro(i,:) - mean(gyro(1:50,:)); %removing bias
end %Demeaning data reduces drift in gyro

pitchzero = zeros(1,5);
rollzero = zeros(1,5);
pitch = zeros(length(accel),1);
roll = zeros(length(accel),1);
rollpre = zeros(length(accel),1);
yaw = zeros(length(accel),1);
CMx = zeros(length(filt_mag),1);
CMy = zeros(length(filt_mag),1);
sign = zeros(1,1);
wGyro = 10; %This is the weighting of the gyroscope 5-20 was not variable in
this program
for i = 2:1:length(accel)
    Axz(i-1)= atan2(R_est(i,1),R_est(i,3));%Finding angle between
projections x,z

```

```

    Ayz(i-1)= atan2(R_est(i,2),R_est(i,3));%between y,z
    gyro_avg(i,2) = (gyro(i,2)+ gyro(i-1,2))/2; %Average gyro y values
    gyro_avg(i,1) = (gyro(i,1)+ gyro(i-1,1))/2; %Average gyro x values
    Axz(i) = Axz(i-1)+gyro_avg(i,2)*.101; %Find new angle w/ gyro data
and T=.05 (framerate)
    Ayz(i) = Ayz(i-1)+gyro_avg(i,1)*.101;
    Rx_gyro(i) = sin(Axz(i))/sqrt(1+cos(Axz(i)).^2 *tan(Ayz(i)).^2);
    Ry_gyro(i) = sin(Ayz(i))/sqrt(1+cos(Ayz(i)).^2 *tan(Axz(i)).^2);
    if R_est(i-1,3) < 0 %Used for multiplying sign by Rz
        sign = -1;
    else
        sign = 1;
    end
    Rz_gyro(i) = sign.*sqrt(1-Rx_gyro(i).^2 - Ry_gyro(i).^2);
    R_est(i,1) = (filt_acc(i,1)+Rx_gyro(i)*wGyro)/(1+wGyro);
    R_est(i,2) = (filt_acc(i,2)+Ry_gyro(i)*wGyro)/(1+wGyro);
    R_est(i,3) = (filt_acc(i,3)+Rz_gyro(i)*wGyro)/(1+wGyro);
    R(i,1) = sqrt((R_est(i,1)^2)+(R_est(i,2)^2)+(R_est(i,3)^2));
    R_est(i,1) = R_est(i,1)/R(i,1);
    R_est(i,2) = R_est(i,2)/R(i,1);
    R_est(i,3) = R_est(i,3)/R(i,1);

for i = 2:length(filt_mag)
    if i < 6
        pitchzero(1,i) = -atan2(sqrt(R_est(i,2)^2+ R_est(i,3)^2),R_est(i,1));
%calculating initial Euler angles
        rollzero(1,i) = -atan2(sqrt(R_est(i,1)^2+ R_est(i,3)^2),R_est(i,2));
    else
        pitch(i,1) = -atan2(sqrt(R_est(i,2)^2+ R_est(i,3)^2),R_est(i,1))-
mean(pitchzero) ; %Subtracting initial Euler angle to be at 0 when begin
        roll(i,1) = -atan2(sqrt(R_est(i,1)^2+ R_est(i,3)^2),R_est(i,2))-
mean(rollzero);
    end
for i = 20:length(filt_mag) %Used to calculate yaw angle for vector-based
method
    CMx(i,1) =
filt_mag(i,1)*cos(pitch(i,1))+filt_mag(i,2)*sin(rollpre(i,1))*sin(pitch(i,1))
+filt_mag(i,3)*cos(rollpre(i,1))*sin(pitch(i,1));%This tilt compensates the
magnetometer data using pitch and roll
    CMy(i,1) = filt_mag(i,2)*cos(rollpre(i,1))-
filt_mag(i,3)*sin(rollpre(i,1));
    yaw(i,1)= atan(CMy(i,1)/CMx(i,1));
end

for i = 1:length(roll) %If roll, pitch, yaw 75-105 then use DCM vals
yaw(i) = yaw(i)*180/pi;
roll(i) = roll(i)*180/pi;
pitch(i) = pitch(i)*180/pi;
if roll(i) >= 75 && roll(i) <= 105
roll(i) = rollDCM(i);
end
if pitch(i) >= 75 && pitch(i) <= 105
pitch(i) = pitchDCM(i);
end
if yaw(i) >= 75 && yaw(i) <= 105
yaw(i) = yawDCM(i);
end
end

```

```

end

%THIS IS ONLY FOR SCALING PURPOSE
for i = 1:length(yaw)
    yaw(i,1) = yaw(i,1) + yaw(i,1)*.156;
    roll(i,1) = roll(i,1) + roll(i,1)*.189;
    pitch(i,1) = pitch(i,1) + pitch(i,1)*.288;
end
pitch = pitch';
roll = roll';
yaw = yaw';

[B,A] = butter(1,1); %Coefficients for filter
pitch = filtfilt(B,A,pitch(:,,:)); %Using low pass filter
roll = filtfilt(B,A,roll(:,,:));
yaw = filtfilt(B,A,yaw(:,,:));
pitch = pitch - pitch_0; %Subtracting original orientation to start at 0
roll = roll - roll_0;
yaw = yaw - yaw_0;

%FINDING FORCE VECTOR COEFFICIENTS
Fx = zeros(length(pitch),1);
Fy = zeros(length(roll),1);
Fz = zeros(length(yaw),1);
for i = 1:length(R_est)
    R_bias = [0 0 1] - mean(R_est(1:50,:)); %Subtracting int position
    R_est(i,:) = R_est(i, :)+R_bias;
end
Fx = zeros(length(pitch),1);
Fy = zeros(length(roll),1);
Fz = zeros(length(yaw),1);
%This calculates vector components of force vector
for i = 1:length(R_est)
    d(i) = norm(R_est(i,:));
    Fx(i,1) = R_est(i,1)/d(i); %Finds the coefficient for where direction of
F applied
    Fy(i,1) = R_est(i,2)/d(i);
    Fz(i,1) = R_est(i,3)/d(i);
end

```

c) mpu6050.m

This script is for every mpu6050 data. This converts raw sensor data into orientation of the hand.

```

last = length(mpu);
[B,A] = butter(1,1); %Returns filter coefficients for a first order low pass
filter

```

```

%CONVERTING RAW DATA TO ACCEL, GYRO DATA
%Using mpu6050 data
%THUMB

```

```

if whichsensor == 2; %Whichsensor is chosen in allsensors.m to pick sensor
    testaccel = mpu(:,1:3); %accelerometer values from sensorstick
    accel_x = ((testaccel(:,1))/16534 - .0258); %Converting values to +/- 2g,
1g = 16534 without offset. xmin = -16,108 xmax = 16,960
    accel_y = (testaccel(:,2)/16812 - .0271); %ymin = -16356, ymax = 17,268
    accel_z = (testaccel(:,3)/17002 + .0411); %zmin = -17,700, zmax = 16,304

accel_y = accel_y - mean(accel_y(1:50)); %Removing bias
accel_x = accel_x - mean(accel_x(1:50));
accel_z = accel_z + (1 - mean(accel_z(1:50)));
accel = [accel_x,accel_y,accel_z];

filt_acc = filtfilt(B,A,accel(:,:)); %Filtering using Low Pass Butterworth
Filter

%INDEX
else if whichsensor == 3; %Which sensor chooses index finger
    testaccel = mpu(:,1:3); %accelerometer values from sensorstick
    accel_x = ((testaccel(:,1))/16598 + .0069); %Converting values to +/- 1g,
1g = 16598 without offset. xmin = -16,628 xmax = 16,484
    accel_y = (testaccel(:,2)/16766 - .0052); %ymin = -16,688, ymax = 16,864
    accel_z = (testaccel(:,3)/17084 - .1082); %zmin = -18,932, zmax = 15,236

    accel_y = accel_y - mean(accel_y(1:50)); %Removing bias
    accel_x = accel_x - mean(accel_x(1:50));
    accel_z = accel_z - (1 - mean(accel_z(1:50)));
    accel = [accel_x,accel_y,accel_z];

filt_acc = filtfilt(B,A,accel(:,:)); %Filtering using Low Pass Butterworth
Filter

%MIDDLE
else if whichsensor == 4; %Which sensor chooses middle finger
    testaccel = mpu(:,1:3); %accelerometer values from sensorstick
    accel_x = ((testaccel(:,1))/16662 + .003); %Converting values to +/- 1g,
1g = 16662 without offset. xmin = -16,712 xmax = 16,612
    accel_y = (testaccel(:,2)/16874 - .0167); %ymin = -16,592, ymax = 17,156
    accel_z = (testaccel(:,3)/17102 - .1082); %zmin = -17,336, zmax = 16,868

    accel_y = accel_y - mean(accel_y(1:50)); %Removing bias
    accel_x = accel_x - mean(accel_x(1:50));
    accel_z = accel_z - (1 - mean(accel_z(1:50)));
    accel = [accel_x,accel_y,accel_z];

filt_acc = filtfilt(B,A,accel(:,:)); %Filtering using Low Pass Butterworth
Filter

%RING
else if whichsensor == 5; %Which sensor chooses ring finger
    testaccel = mpu(:,1:3); %accelerometer values from sensorstick
    accel_x = ((testaccel(:,1))/16628 - .0303); %Converting values to +/- 1g,
1g = 16628 without offset. xmin = -16,124 xmax = 17,132
    accel_y = (testaccel(:,2)/16740 - .0005); %ymin = -16,732, ymax = 16,748
    accel_z = (testaccel(:,3)/16664 + .0595); %zmin = -17,656, zmax = 15,672

    accel_y = accel_y - mean(accel_y(1:50));%Removing bias
    accel_x = accel_x - mean(accel_x(1:50));

```

```

accel_z = accel_z + (1 - mean(accel_z(1:50)));
accel = [accel_x,accel_y,accel_z];
filt_acc = filtfilt(B,A,accel(:, :)); %Filtering using Low Pass Butterworth
Filter

    %PINKY
        else if whichsensor == 6; %Which sensor chooses pinky finger
            testaccel = mpu(:,1:3); %accelerometer values from sensorstick
            accel_x = ((testaccel(:,1))/17526 + .0234); %Converting values to +/- 1g,
1g = 17,526 without offset. xmin = -17,936 xmax = 17,116
            accel_y = (testaccel(:,2))/17602 + .0303); %ymin = -18,136, ymax = 17,068
            accel_z = (testaccel(:,3))/17316 + .2354); %zmin = -21,392, zmax = 13,240

accel_y = accel_y - mean(accel_y(1:50)); %Removing bias
accel_x = accel_x - mean(accel_x(1:50));
accel_z = accel_z - (1 - mean(accel_z(1:50)));
accel = [accel_x,accel_y,accel_z];
filt_acc = filtfilt(B,A,accel(:, :)); %Filtering using Low Pass Butterworth
Filter

                end
            end
        end
    end

%Using mpu6050 data gyroscope
    testgyro = (mpu(:,4:6));
    gyro = (testgyro(:, :)/ 65.536); %Converting 16 bit # to +/- 500 deg/sec
    for i = 1:length(gyro)
        gyro(i, :) = gyro(i, :) - mean(gyro(1:50, :)); %demeaning data reduces
gyro drift
    end
    %Normalizing vectors
    %Normalizing accelerometer data
    for i = 1:length(filt_acc)
        for j = 1:3
            filt_acc(i, j) = filt_acc(i, j) ./ norm(filt_acc(i, :));
        end
    end

%DCM Calculation
Kb = -filt_acc; %Vector for accelerometer data - aligned with zenith
Ib = filt_mag; %Uses mag values from sensorstick
% sa = 10; %This sets the weighting of the accelerometer can be changed with
performance metric
sg = 10;
% sm = 10;
DCM = zeros(1:3,1:3,1:length(filt_acc)); %Initializing DCM
for i = 2:length(filt_acc)-1
    Jb(i, :) = cross(Kb(i, :), Ib(i, :)); %Vector that is orthogonal to both vectors
    DCM(1:3,1:3, i) = [Ib(1, :); Jb(1, :); Kb(1, :)];
    d_thetag(i, :) = dt*gyro(i, :);
    Kbg(i, :) = Kb(i, :) + (cross(d_thetag(i, :), Kb(1, :)));
    d_thetaa(i, :) = cross(Kb(i, :), Kb(i+1, :)-Kb(1, :));

```

```

d_thetam(i,:) = cross(Ib(i,:),Ib(i+1:)-Ib(1,:));
d_theta(i,:) = (sa(i)*d_thetaa(i,:) + sg*d_theta(i,:) +
sm(i)*d_thetam(i,:))/(sa(i)+sg+sm(i));%This is where weighting comes into
play making angle progression
Kb(i+1,:) = Kb(i,:)+ cross(d_theta(i,:),Kb(1,:)); %Update each vector
Ib(i+1,:) = Ib(i,:)+ cross(d_theta(i,:),Ib(1,:));
Jb(i+1,:) = Jb(i,:)+ cross(d_theta(i,:),Jb(1,:));
Err(i) = (dot(Ib(i+1,:),Jb(i+1,:))); %Ensure all vectors are norm and orthog
Ib(i+1,:) = Ib(i+1,:) - Err(i)*Ib(i+1,:);
Jb(i+1,:) = Jb(i+1,:) - Err(i)*Jb(i+1,:);
Ib(i+1,:) = Ib(i+1,:)/norm(Ib(i+1,:)); %Normalize
Jb(i+1,:) = Jb(i+1,:)/norm(Jb(i+1,:));
Kb(i+1,:) = cross(Ib(i+1,:),Jb(i+1,:));
Kb(i+1,:) = Kb(i+1,:)/norm(Kb(i+1,:)); %Normalize
DCM(1:3,1:3,i) = [Ib(i,:); Jb(i,:); Kb(i,:)];
rollDCM(1,i) = -atan2(DCM(3,1,i),DCM(1,1,i))*180/pi; %Calculated Euler angles
from DCM
pitchDCM(1,i) = -asin(DCM(1,1,i))*180/pi;
yawDCM(1,i) = -atan2(DCM(2,1,i),DCM(1,1,i))*180/pi;
end

%%%%%%%%%VECTOR METHOD
R_est = filt_acc; %Use accel to calculate grav force vector
R = zeros(length(accel),1);
gyro_avg = zeros(length(accel),3);
Axz = zeros(length(accel),1);
Ayz = zeros(length(accel),1);
Rx_gyro = zeros(length(accel),1);
Ry_gyro = zeros(length(accel),1);
Rz_gyro = zeros(length(accel),1);
pitchzero = zeros(1,5);
rollzero = zeros(1,5);
pitch = zeros(length(accel),1);
roll = zeros(length(accel),1);
yaw = zeros(length(accel),1);
CMx = zeros(length(mag),1);
CMy = zeros(length(mag),1);
sign = zeros(1,1);
wGyro = 7; %This is the weighting of the gyroscope 5-20, can be changed using
weighting scheme used earlier
for i = 2:1:length(accel)
    Axz(i-1)= atan2(R_est(i,1),R_est(i,3));%Finding angle between
projections x,z
    Ayz(i-1)= atan2(R_est(i,2),R_est(i,3));%between y,z
    gyro_avg(i,2) = (gyro(i,2)+ gyro(i-1,2))/2; %Average gyro y values
    gyro_avg(i,1) = (gyro(i,1)+ gyro(i-1,1))/2; %Average gyro x values
    Axz(i) = Axz(i-1)+gyro_avg(i,2)*.1; %Find new angle w/ gyro data and
T=.1( framerate)
    Ayz(i) = Ayz(i-1)+gyro_avg(i,1)*.1;
    Rx_gyro(i) = sin(Axz(i))/sqrt(1+cos(Axz(i)).^2 *tan(Ayz(i)).^2);
    Ry_gyro(i) = sin(Ayz(i))/sqrt(1+cos(Ayz(i)).^2 *tan(Axz(i)).^2);
    if R_est(i-1,3) < 0
        sign = -1; %Used for multiplying sign by Rz
    else
        sign = 1;
    end
    Rz_gyro(i) = sign.*sqrt(1-Rx_gyro(i).^2 - Ry_gyro(i).^2);

```

```

        R_est(i,1) = (filt_acc(i,1)+Rx_gyro(i)*wGyro)/(1+wGyro);
        R_est(i,2) = (filt_acc(i,2)+Ry_gyro(i)*wGyro)/(1+wGyro);
        R_est(i,3) = (filt_acc(i,3)+Rz_gyro(i)*wGyro)/(1+wGyro);
        R_est(i,:) = R_est(i,:)/norm(R_est(i,:));
end

for i = 1:length(filt_acc)
    if i < 6
        pitchzero(1,i) = -atan2(sqrt(R_est(i,2)^2+ R_est(i,3)^2),R_est(i,1));
%Calculating initial Euler angle
        rollzero(1,i) = -atan2(sqrt(R_est(i,1)^2+ R_est(i,3)^2),R_est(i,2));
        else
            pitch(i,1) = -atan2(sqrt(R_est(i,2)^2+ R_est(i,3)^2),R_est(i,1))-
mean(pitchzero);
            roll(i,1) = -atan2(sqrt(R_est(i,1)^2+ R_est(i,3)^2),R_est(i,2))-
mean(rollzero);
        end
%Using DCM vals if possible
for i = 1:length(roll) %If roll, pitch, yaw 75-105 then use DCM vals
yaw(i) = yaw(i)*180/pi;
roll(i) = roll(i)*180/pi;
pitch(i) = pitch(i)*180/pi;
if roll(i) >= 75 && roll(i) <= 105
roll(i) = rollDCM(i);
end
if pitch(i) >= 75 && pitch(i) <= 105
pitch(i) = pitchDCM(i);
end
if yaw(i) >= 75 && yaw(i) <= 105
yaw(i) = yawDCM(i);
end
end

yaw_0 = median(yaw(1:100,:));
pitch_0 = median(pitch(1:100,:));
roll_0 = median(roll(1:100,:));

pitch = pitch - pitch_0 ;
roll = roll - roll_0;
yaw = yaw-yaw_0;

pitch = filtfilt(B,A,pitch(:, :)); %Using low pass filter
roll = filtfilt(B,A,roll(:, :));
yaw = filtfilt(B,A,yaw(:, :));

%This code adds in the offsets that occur by comparing electromagnetic
%sensors to the glove sensors. Each offset may be different for each sensor
%placement.
% %THUMB SENSOR
if whichsensor == 2
    for i = 1:length(yaw_hand) %using Yaw calculated from hand
        yaw(i,1) = yaw_hand(i,1) - 29.3333;
        roll(i,1) = roll(i,1) - 28.3333; %offsets found
        pitch(i,1) = pitch(i,1) - 26.6666;
    end
end
end

```

```

%INDEX SENSOR
if whichsensor == 3
    for i = 1:length(yaw_hand) %using Yaw calculated from hand
        yaw(i,1) = yaw_hand(i,1) -19.3333;
        roll(i,1) = roll(i,1) + 11; %offsets found
        pitch(i,1) = pitch(i,1) - 2.3333;
    end
end
%MIDDLE SENSOR
if whichsensor == 4
    for i = 1:length(yaw_hand) %using Yaw calculated from hand
        yaw(i,1) = yaw_hand(i,1) - 3;
        roll(i,1) = roll(i,1) + 16; %offsets found
        pitch(i,1) = pitch(i,1) -7.3333;
    end
end
%RING SENSOR
if whichsensor == 5
    for i = 1:length(yaw_hand) %using Yaw calculated from hand
        yaw(i,1) = yaw_hand(i,1) - 10.6666;
        roll(i,1) = roll(i,1) + 23.3333; %offsets found
        pitch(i,1) = pitch(i,1);
    end
end
%PINKY SENSOR
if whichsensor == 6
    for i = 1:length(yaw_hand) %using Yaw calculated from hand
        yaw(i,1) = yaw_hand(i,1) - 29.6666;
        roll(i,1) = roll(i,1) - 5.3333; %offsets found
        pitch(i,1) = pitch(i,1) + 5;
    end
end

% %This calculates the vector components of force vector
for i = 1:length(R_est)
    R_bias = [0 0 1] - mean(R_est(1:50,:));
    R_est(i,:) = R_est(i, :)+R_bias;
end
Fx = zeros(length(pitch),1);
Fy = zeros(length(roll),1);
Fz = zeros(length(yaw),1);
%This calculates vector components of force vector
for i = 1:length(R_est)
    d(i) = norm(R_est(i,:));
    Fx(i,1) = R_est(i,1)/d(i);
    Fy(i,1) = R_est(i,2)/d(i);
    Fz(i,1) = R_est(i,3)/d(i);
end

```

d) `allsensors.m`

This script calls on `allsensors`, `mpu6050`, and `angleconverter` and outputs all Euler angle and force data for each sensor and sums the components.

```
readoff; %Call readoff to obtain data
hand = zeros(6:6:length(C),9);
index = zeros(6:6:length(C),7);
middle = zeros(6:6:length(C),7);
ring = zeros(6:6:length(C),7);
pinky = zeros(6:6:length(C),7);
thumb = zeros(6:6:length(C),7);
p = 1;
j = 1;
for i = 6:6:length(C) %This loop makes sure to skip lines in raw code if the
order of the data gets mixed up. Data format should always remain the same.

    if isnan(C(i,9)) == 1
        i = i - 1;
        p = i;
        if isnan(C(i,9)) == 1
            i = i-1;
            p = i;
            if isnan(C(i,9)) == 1
                i = i-1;
                p = i;
                if isnan(C(i,9)) == 1
                    i = i-1;
                    p = i;
                    if isnan(C(i,9)) == 1
                        i = i-1;
                        p = i;
                        if isnan(C(i,9)) == 1
                            i = i-1;
                            p = i;
                        end
                    end
                end
            end
        end
    end
end
end
end
end
end
hand(j,:) = C(i,:); %These separate the C matrix by sensor location
thumb(j,:) = C(i-5,1:7);
index(j,:) = C(i-4,1:7);
middle(j,:) = C(i-3,1:7);
ring(j,:) = C(i-2,1:7);
pinky(j,:) = C(i-1,1:7);
j = j + 1;
i = p;
end
[row, col] = find(isnan(hand)); %These if statements ensure that there are
no lines of data with Nan inside of them and if there are that the row be
skipped
if isempty(row) == 0; %If the row is not empty, meaning a Nan has been found
thumb = [thumb(1:row-1, :);thumb(row+1:end, :)];
hand = [hand(1:row-1, :);hand(row+1:end, :)];
index = [index(1:row-1, :);index(row+1:end, :)];
middle = [middle(1:row-1, :);middle(row+1:end, :)];
```

```

        ring = [ring(1:row-1, :);ring(row+1:end, :)];
        pinky = [pinky(1:row-1, :);pinky(row+1:end, :)];
    end
    [row, col] = find(isnan/thumb));
    if isempty(row) == 0
        thumb = [thumb(1:row-1, :);thumb(row+1:end, :)];
        hand = [hand(1:row-1, :);hand(row+1:end, :)];
        index = [index(1:row-1, :);index(row+1:end, :)];
        middle = [middle(1:row-1, :);middle(row+1:end, :)];
        ring = [ring(1:row-1, :);ring(row+1:end, :)];
        pinky = [pinky(1:row-1, :);pinky(row+1:end, :)];
    end
    [row, col] = find(isnan/index));
    if isempty(row) == 0;
    thumb = [thumb(1:row-1, :);thumb(row+1:end, :)];
        hand = [hand(1:row-1, :);hand(row+1:end, :)];
        index = [index(1:row-1, :);index(row+1:end, :)];
        middle = [middle(1:row-1, :);middle(row+1:end, :)];
        ring = [ring(1:row-1, :);ring(row+1:end, :)];
        pinky = [pinky(1:row-1, :);pinky(row+1:end, :)];
    end
    [row, col] = find(isnan/middle));
    if isempty(row) == 0
        thumb = [thumb(1:row-1, :);thumb(row+1:end, :)];
        hand = [hand(1:row-1, :);hand(row+1:end, :)];
        index = [index(1:row-1, :);index(row+1:end, :)];
        middle = [middle(1:row-1, :);middle(row+1:end, :)];
        ring = [ring(1:row-1, :);ring(row+1:end, :)];
        pinky = [pinky(1:row-1, :);pinky(row+1:end, :)];
    end
    [row, col] = find(isnan/ring));
    if isempty(row) == 0;
    thumb = [thumb(1:row-1, :);thumb(row+1:end, :)];
        hand = [hand(1:row-1, :);hand(row+1:end, :)];
        index = [index(1:row-1, :);index(row+1:end, :)];
        middle = [middle(1:row-1, :);middle(row+1:end, :)];
        ring = [ring(1:row-1, :);ring(row+1:end, :)];
        pinky = [pinky(1:row-1, :);pinky(row+1:end, :)];
    end
    [row, col] = find(isnan/pinky));
    if isempty(row) == 0
        thumb = [thumb(1:row-1, :);thumb(row+1:end, :)];
        hand = [hand(1:row-1, :);hand(row+1:end, :)];
        index = [index(1:row-1, :);index(row+1:end, :)];
        middle = [middle(1:row-1, :);middle(row+1:end, :)];
        ring = [ring(1:row-1, :);ring(row+1:end, :)];
        pinky = [pinky(1:row-1, :);pinky(row+1:end, :)];
    end
    [row, col] = find(isnan/hand));
    if isempty(row) == 0;
    thumb = [thumb(1:row-1, :);thumb(row+1:end, :)];
        hand = [hand(1:row-1, :);hand(row+1:end, :)];
        index = [index(1:row-1, :);index(row+1:end, :)];
        middle = [middle(1:row-1, :);middle(row+1:end, :)];
        ring = [ring(1:row-1, :);ring(row+1:end, :)];
        pinky = [pinky(1:row-1, :);pinky(row+1:end, :)];
    end
end

```

```

[row, col] = find(isnan(thumb));
if isempty(row) == 0
    thumb = [thumb(1:row-1,:);thumb(row+1:end,:)];
    hand = [hand(1:row-1,:);hand(row+1:end,:)];
    index = [index(1:row-1,:);index(row+1:end,:)];
    middle = [middle(1:row-1,:);middle(row+1:end,:)];
    ring = [ring(1:row-1,:);ring(row+1:end,:)];
    pinky = [pinky(1:row-1,:);pinky(row+1:end,:)];
end
[row, col] = find(isnan(index));
if isempty(row) == 0;
thumb = [thumb(1:row-1,:);thumb(row+1:end,:)];
    hand = [hand(1:row-1,:);hand(row+1:end,:)];
    index = [index(1:row-1,:);index(row+1:end,:)];
    middle = [middle(1:row-1,:);middle(row+1:end,:)];
    ring = [ring(1:row-1,:);ring(row+1:end,:)];
    pinky = [pinky(1:row-1,:);pinky(row+1:end,:)];
end
[row, col] = find(isnan(middle));
if isempty(row) == 0
    thumb = [thumb(1:row-1,:);thumb(row+1:end,:)];
    hand = [hand(1:row-1,:);hand(row+1:end,:)];
    index = [index(1:row-1,:);index(row+1:end,:)];
    middle = [middle(1:row-1,:);middle(row+1:end,:)];
    ring = [ring(1:row-1,:);ring(row+1:end,:)];
    pinky = [pinky(1:row-1,:);pinky(row+1:end,:)];
end
[row, col] = find(isnan(ring));
if isempty(row) == 0;
thumb = [thumb(1:row-1,:);thumb(row+1:end,:)];
    hand = [hand(1:row-1,:);hand(row+1:end,:)];
    index = [index(1:row-1,:);index(row+1:end,:)];
    middle = [middle(1:row-1,:);middle(row+1:end,:)];
    ring = [ring(1:row-1,:);ring(row+1:end,:)];
    pinky = [pinky(1:row-1,:);pinky(row+1:end,:)];
end
[row, col] = find(isnan(pinky));
if isempty(row) == 0
    thumb = [thumb(1:row-1,:);thumb(row+1:end,:)];
    hand = [hand(1:row-1,:);hand(row+1:end,:)];
    index = [index(1:row-1,:);index(row+1:end,:)];
    middle = [middle(1:row-1,:);middle(row+1:end,:)];
    ring = [ring(1:row-1,:);ring(row+1:end,:)];
    pinky = [pinky(1:row-1,:);pinky(row+1:end,:)];
end
thumb_force = zeros(length(thumb),1); %Initializing force data for each
sensor
index_force = zeros(length(index),1);
middle_force = zeros(length(middle),1);
ring_force = zeros(length(ring),1);
pinky_force = zeros(length(pinky),1);
Fx_total = zeros(length(pinky),1);
Fy_total = zeros(length(pinky),1);
Fz_total = zeros(length(pinky),1);
Fx_thumb = zeros(length(pinky),1);
Fy_thumb = zeros(length(pinky),1);
Fz_thumb = zeros(length(pinky),1);

```

```

Fx_index = zeros(length(pinky),1);
Fy_index = zeros(length(pinky),1);
Fz_index = zeros(length(pinky),1);
Fx_middle = zeros(length(pinky),1);
Fy_middle = zeros(length(pinky),1);
Fz_middle = zeros(length(pinky),1);
Fx_ring = zeros(length(pinky),1);
Fy_ring = zeros(length(pinky),1);
Fz_ring = zeros(length(pinky),1);
Fx_pinky = zeros(length(pinky),1);
Fy_pinky = zeros(length(pinky),1);
Fz_pinky = zeros(length(pinky),1);
i = 1;
for i = 1:6 %This loops through each motion sensor
    if i == 1
        sensorstick = hand; %Using sensorstick.m to fuse data from
            sensorstick9dof
            sensor
            whichsensor = 1; %Sets sensor to hand sensor
            angleconverter; %Uses data from sensorstick
            yaw_hand = yaw; %Sets Euler angle value from sensorstick to hand
            pitch_hand = pitch;
            roll_hand = roll;
        end
    if i == 2
        mpu = thumb(:,2:7); %Uses thumb motion data
        for i = 1:length(thumb(:,1))
            if thumb(i,1) >> 3 %If voltage is greater than 3 use cubic
                thumb_force(i) = 1.6657*(thumb(i,1)).^3 - 8.6641*(thumb(i,1)).^2 +
                    13.982*(thumb(i,1)); %Sensor 1 flexiforce
            else
                thumb_force(i) = 4.561*thumb(i,1); %Use linear fit if 3 or below
            end
        end
        end
        whichsensor = 2; %Sets sensor to thumb
        mpu6050; %Runs mpu6050 for thumb
        yaw_thumb = yaw; %Sets Euler angle value from mpu6050 to thumb
        pitch_thumb = pitch;
        roll_thumb = roll;
        for j = 1:length(Fx)
            Fx_thumb(j,1) = Fx(j,1)*thumb_force(j,1); %Multiplies coefficient to
                scalar value of thumb force
            Fy_thumb(j,1) = Fy(j,1)*thumb_force(j,1);
            Fz_thumb(j,1) = Fz(j,1)*thumb_force(j,1);
        end
    end
    if i == 3
        mpu = index(:,2:7);
        for i = 1:length(index(:,1))
            if index(i,1) >> 3 %If greater than 3 use cubic fit
                index_force(i) = 1.6595*(index(i,1)).^3 - 5.6514*(index(i,1)).^2 +
                    6.6808*(index(i,1)); %Sensor 2 flexiforce cubic fit
            else
                index_force(i) = 4.6633*(index(i,1)); %Use linear fit
            end
        end
    end
    end
    whichsensor = 3; %Sets sensor to index

```

```

mpu6050; %Runs mpu6050 for index
yaw_index = yaw; %sets Euler angle value from mpu6050 to index
pitch_index = pitch;
roll_index = roll;
for j = 1:length(Fx)
    Fx_index(j,1) = Fx(j,1)*index_force(j,1); %Multiplies coefficient to
    scalar value of thumb force
    Fy_index(j,1) = Fy(j,1)*index_force(j,1);
    Fz_index(j,1) = Fz(j,1)*index_force(j,1);
end
end
if i == 4
    mpu = middle(:,2:7); %Uses motion data from middle
    for i = 1:length(middle(i,1));
        if middle(i,1) >> 3 %If greater than 3 use cubic fit
            middle_force(i) = 2.1619*(middle(i,1)).^3 - 11.985*(middle(i,1)).^2
            +19.01*(middle(i,1)); %Sensor 3 flexiforce cubic fit
        else
            middle_force(i) = 2.5333*middle(i,1); %Use linear fit
        end
    end
    end
    whichsensor = 4; %Sets Sensor to middle
    mpu6050; %Runs mpu6050 for middle
    yaw_middle = yaw; %sets Euler angle value from mpu6050 to middle
    pitch_middle = pitch;
    roll_middle = roll;
    for j = 1:length(Fx)
        Fx_middle(j) = Fx(j,1)*middle_force(j,1); %Multiplies coefficient to
        scalar value of middle force
        Fy_middle(j) = Fy(j,1)*middle_force(j,1);
        Fz_middle(j) = Fz(j,1)*middle_force(j,1);
    end
    end
    if i == 5
        mpu = ring(:,2:7); %Uses motion data from ring
        for i = 1:length(ring(i,1));
            if ring(i,1) >> 3 %If greater than 3 use cubic fit
                ring_force(i) = 1.9377*(ring(i,1)).^3 - 10.301*(ring(i,1)).^2 +
                16.085*(ring(i,1)); %Sensor 4 flexiforce cubic fit
            else
                ring_force(i) = 2.721*(ring(i,1));
            end
        end
        end
        whichsensor = 5; %Sets sensor to ring
        mpu6050; %Runs mpu6050 for ring
        yaw_ring = yaw; %Sets Euler angle value from mpu6050 to ring
        pitch_ring = pitch;
        roll_ring = roll;
        for j = 1:length(Fx)
            Fx_ring(j) = Fx(j,1)*ring_force(j,1); %Multiplies coefficient to
            scalar value of ring force
            Fy_ring(j) = Fy(j,1)*ring_force(j,1);
            Fz_ring(j) = Fz(j,1)*ring_force(j,1);
        end
        end
        if i == 6
            mpu = pinky(:,2:7); %Uses motion data from pinky

```

```

for i = 1:length(pinky(i,1));
    if pinky(i,1) >> 3 %If greater than 3 use cubic fit
        pinky_force(i) = 1.8429*(pinky(i,1)).^3 - 9.6988*(pinky(i,1)).^2 +
            14.755*(pinky(i,1)); %Sensor 5 flexiforce cubic fit
    else
        pinky_force(i) = 2.3133*pinky(i,1); %Use linear fit
    end
end
whichsensor = 6; %Sets sensor to pinky
mpu6050; %Runs mpu6050 for pinky
yaw_pinky = yaw; %Sets Euler angle value from mpu6050 to pinky
pitch_pinky = pitch;
roll_pinky = roll;
for j = 1:length(Fx)
    Fx_pinky(j) = Fx(j,1)*pinky_force(j,1); %Multiplies coefficient to
        scalar value of pinky force
    Fy_pinky(j) = Fy(j,1)*pinky_force(j,1);
    Fz_pinky(j) = Fz(j,1)*pinky_force(j,1);
end
end
end
i = 1;

for i = 1:length(Fx) %Sums up every component at each time pt
    Fx_total(i,1) =
(Fx_thumb(i,1)+Fx_index(i,1)+Fx_middle(i,1)+Fx_ring(i,1)+Fx_pinky(i,1));
    Fy_total(i,1) =
(Fy_thumb(i,1)+Fy_index(i,1)+Fy_middle(i,1)+Fy_ring(i,1)+Fy_pinky(i,1));
    Fz_total(i,1) =
(Fz_thumb(i,1)+Fz_index(i,1)+Fz_middle(i,1)+Fz_ring(i,1)+Fz_pinky(i,1));
    My_total(i,1) = Fz_total(i,1)*.38815; %Moment arm of .38815m was used
    Mz_total(i,1) = Fy_total(i,1)*.38815;
    F_total(i,1) =
(abs/thumb_force(i,1))+abs(index_force(i,1))+abs(middle_force(i,1))+abs(ring_
force(i,1))+abs(pinky_force(i,1));
end

```

e) MotionTesting.m

This script calls on allsensors to determine error components for each Euler angle and sensor.

```

%mm = dlmread('whatyou are testing')
mm = dlmread('supineyaw90_test2.txt'); %Calling em data
fid = fopen('supineyaw90_test2'); %Calling glove data
test = 1; %Change this for test #
allsensors;
%
[L W] = size(mm); %Size of em data
for j = 2:W
mm(:,j) = mm(:,j) - mm(50,j); %Makes em data start at 0
end

%This loop is performed for each Euler angle and sensor this just gives a
loop of roll for the pinky sensor
%% %%Pinky

```

```

%%Roll
%flat1
mm_pinky_flat1_roll = mm(100:10:900,2); %EM Euler angle time phase 1
comp_pinky_flat1_roll = roll_pinky(10:90); %Glove data time phase 1
for i = 1:length(roll_pinky) %This loop prevents when data jumps from +180 to
-180 or vice versa
    if abs(mm_pinky_flat1_roll(i)-comp_pinky_flat1_roll(i)) >= 290
        comp_pinky_flat1_roll(i) = -comp_pinky_flat1_roll(i);
    end
end
pinky_flat1_roll_angle = mean(mm_pinky_flat1_roll); %Average angle EM
RMSE_comp_pinky_flat1_roll = sqrt(sum(((mm_pinky_flat1_roll)-
(comp_pinky_flat1_roll)').^2)/81); %RMSE between both
AVGE_comp_pinky_flat1_roll = sum((mm_pinky_flat1_roll)-
(comp_pinky_flat1_roll)')/81;

%flat2
mm_pinky_flat2_roll = mm(2400:10:3200,2); %Second time phase em angle
comp_pinky_flat2_roll = roll_pinky(240:320); %Second time phase glove angle
pinky_flat2_roll_angle = mean(mm_pinky_flat2_roll);
for i = 1:length(roll_pinky) %This loop prevents when data jumps from +180 to
-180 or vice versa
    if abs(mm_pinky_flat2_roll(i)-comp_pinky_flat2_roll(i)) >= 290
        comp_pinky_flat2_roll(i) = -comp_pinky_flat2_roll(i);
    end
end
RMSE_comp_pinky_flat2_roll = sqrt(sum(((mm_pinky_flat2_roll)-
(comp_pinky_flat2_roll)').^2)/81);
AVGE_comp_pinky_flat2_roll = sum((mm_pinky_flat2_roll)-
(comp_pinky_flat2_roll)')/81;
%transient
mm_pinky_transient_roll = mm(1200:10:2000,2); %Transient time phase em angle
comp_pinky_transient_roll = roll_pinky(120:200); %Transient time phase glove
angle
for i = 1:length(roll_pinky) %This loop prevents when data jumps from +180 to
-180 or vice versa
    if abs(mm_pinky_transient_roll(i)-comp_pinky_transient_roll(i)) >= 290
        comp_pinky_transient_roll(i) = -comp_pinky_transient_roll(i);
    end
end
pinky_transient_roll_angle = mean(mm_pinky_transient_roll);
RMSE_comp_pinky_transient_roll = sqrt(sum(((mm_pinky_transient_roll)-
(comp_pinky_transient_roll)').^2)/81); %RMSE transient roll
AVGE_comp_pinky_transient_roll = sum((mm_pinky_transient_roll)-
(comp_pinky_transient_roll)')/81;

%This portion of the code combines all data into one variable named glove
if test == 1
glove = {'pinky' 'ring' 'middle' 'index' 'thumb' 'hand'}; %Set variable
end
%PINKY
if test == 1 %Using test #
    j = 1;
end
if test == 2
    j = 10;
end

```

```

if test == 3
    j = 19;
end
if test == 1 || 2 || 3 %If any of the test numbers
    for i = 1:9
        if i == 1
            glove.pinky{j,i} = RMSE_comp_pinky_flat1_roll; %Flat 1 glove RMSE
            glove.pinky{j+1,i} = RMSE_comp_pinky_flat2_roll; %Flat 2 glove RMSE
            glove.pinky{j+2,i} = RMSE_comp_pinky_transient_roll; %Transient glove
RMSE
            glove.pinky{j+3,i} = AVGE_comp_pinky_flat1_roll; %Flat 1 glove AVGE
            glove.pinky{j+4,i} = AVGE_comp_pinky_flat2_roll; %Flat 2 glove AVGE
            glove.pinky{j+5,i} = AVGE_comp_pinky_transient_roll; %Transient glove
AVGE
            glove.pinky{j+6,i} = pinky_flat1_roll_angle; %Flat 1 glove avg ang
            glove.pinky{j+7,i} = pinky_flat2_roll_angle; %Flat 2 glove avg ang
            glove.pinky{j+8,i} = pinky_transient_roll_angle; %Transient glove avg
ang
        elseif i == 2 %If the second test, shift the row downward
            glove.pinky{j,i} = RMSE_comp_pinky_flat1_pitch;
            glove.pinky{j+1,i} = RMSE_comp_pinky_flat2_pitch;
            glove.pinky{j+2,i} = RMSE_comp_pinky_transient_pitch;
            glove.pinky{j+3,i} = AVGE_comp_pinky_flat1_pitch;
            glove.pinky{j+4,i} = AVGE_comp_pinky_flat2_pitch;
            glove.pinky{j+5,i} = AVGE_comp_pinky_transient_pitch;
            glove.pinky{j+6,i} = pinky_flat1_pitch_angle;
            glove.pinky{j+7,i} = pinky_flat2_pitch_angle;
            glove.pinky{j+8,i} = pinky_transient_pitch_angle;
        elseif i == 3
            glove.pinky{j,i} = RMSE_comp_pinky_flat1_yaw;
            glove.pinky{j+1,i} = RMSE_comp_pinky_flat2_yaw;
            glove.pinky{j+2,i} = RMSE_comp_pinky_transient_yaw;
            glove.pinky{j+3,i} = AVGE_comp_pinky_flat1_yaw;
            glove.pinky{j+4,i} = AVGE_comp_pinky_flat2_yaw;
            glove.pinky{j+5,i} = AVGE_comp_pinky_transient_yaw;
            glove.pinky{j+6,i} = pinky_flat1_yaw_angle;
            glove.pinky{j+7,i} = pinky_flat2_yaw_angle;
            glove.pinky{j+8,i} = pinky_transient_yaw_angle;
        elseif i == 4
            glove.pinky{j,i} = RMSE_q_pinky_flat1_roll;
            glove.pinky{j+1,i} = RMSE_q_pinky_flat2_roll;
            glove.pinky{j+2,i} = RMSE_q_pinky_transient_roll;
            glove.pinky{j+3,i} = AVGE_q_pinky_flat1_roll;
            glove.pinky{j+4,i} = AVGE_q_pinky_flat2_roll;
            glove.pinky{j+5,i} = AVGE_q_pinky_transient_roll;
            glove.pinky{j+6,i} = pinky_flat1_roll_angle;
            glove.pinky{j+7,i} = pinky_flat2_roll_angle;
            glove.pinky{j+8,i} = pinky_transient_roll_angle;
        elseif i == 5
            glove.pinky{j,i} = RMSE_q_pinky_flat1_pitch;
            glove.pinky{j+1,i} = RMSE_q_pinky_flat2_pitch;
            glove.pinky{j+2,i} = RMSE_q_pinky_transient_pitch;
            glove.pinky{j+3,i} = AVGE_q_pinky_flat1_pitch;
            glove.pinky{j+4,i} = AVGE_q_pinky_flat2_pitch;
            glove.pinky{j+5,i} = AVGE_q_pinky_transient_pitch;
            glove.pinky{j+6,i} = pinky_flat1_pitch_angle;
            glove.pinky{j+7,i} = pinky_flat2_pitch_angle;
        end
    end
end

```

```

glove.pinky{j+8,i} = pinky_transient_pitch_angle;
elseif i == 6
glove.pinky{j,i} = RMSE_q_pinky_flat1_yaw;
glove.pinky{j+1,i} = RMSE_q_pinky_flat2_yaw;
glove.pinky{j+2,i} = RMSE_q_pinky_transient_yaw;
glove.pinky{j+3,i} = AVGE_q_pinky_flat1_yaw;
glove.pinky{j+4,i} = AVGE_q_pinky_flat2_yaw;
glove.pinky{j+5,i} = AVGE_q_pinky_transient_yaw;
glove.pinky{j+6,i} = pinky_flat1_yaw_angle;
glove.pinky{j+7,i} = pinky_flat2_yaw_angle;
glove.pinky{j+8,i} = pinky_transient_yaw_angle;
elseif i == 7
glove.pinky{j,i} = RMSE_q1_pinky_flat1_roll;
glove.pinky{j+1,i} = RMSE_q1_pinky_flat2_roll;
glove.pinky{j+2,i} = RMSE_q1_pinky_transient_roll;
glove.pinky{j+3,i} = AVGE_q1_pinky_flat1_roll;
glove.pinky{j+4,i} = AVGE_q1_pinky_flat2_roll;
glove.pinky{j+5,i} = AVGE_q1_pinky_transient_roll;
glove.pinky{j+6,i} = pinky_flat1_roll_angle;
glove.pinky{j+7,i} = pinky_flat2_roll_angle;
glove.pinky{j+8,i} = pinky_transient_roll_angle;
elseif i == 8
glove.pinky{j,i} = RMSE_q1_pinky_flat1_pitch;
glove.pinky{j+1,i} = RMSE_q1_pinky_flat2_pitch;
glove.pinky{j+2,i} = RMSE_q1_pinky_transient_pitch;
glove.pinky{j+3,i} = AVGE_q1_pinky_flat1_pitch;
glove.pinky{j+4,i} = AVGE_q1_pinky_flat2_pitch;
glove.pinky{j+5,i} = AVGE_q1_pinky_transient_pitch;
glove.pinky{j+6,i} = pinky_flat1_pitch_angle;
glove.pinky{j+7,i} = pinky_flat2_pitch_angle;
glove.pinky{j+8,i} = pinky_transient_pitch_angle;
elseif i == 9
glove.pinky{j,i} = RMSE_q1_pinky_flat1_yaw;
glove.pinky{j+1,i} = RMSE_q1_pinky_flat2_yaw;
glove.pinky{j+2,i} = RMSE_q1_pinky_transient_yaw;
glove.pinky{j+3,i} = AVGE_q1_pinky_flat1_yaw;
glove.pinky{j+4,i} = AVGE_q1_pinky_flat2_yaw;
glove.pinky{j+5,i} = AVGE_q1_pinky_transient_yaw;
glove.pinky{j+6,i} = pinky_flat1_yaw_angle;
glove.pinky{j+7,i} = pinky_flat2_yaw_angle;
glove.pinky{j+8,i} = pinky_transient_yaw_angle;
end
end
end

%RING
if test == 1 || 2 || 3
for i = 1:9
if i == 1
glove.ring{j,i} = RMSE_comp_ring_flat1_roll;
glove.ring{j+1,i} = RMSE_comp_ring_flat2_roll;
glove.ring{j+2,i} = RMSE_comp_ring_transient_roll;
glove.ring{j+3,i} = AVGE_comp_ring_flat1_roll;
glove.ring{j+4,i} = AVGE_comp_ring_flat2_roll;
glove.ring{j+5,i} = AVGE_comp_ring_transient_roll;
glove.ring{j+6,i} = ring_flat1_roll_angle;
glove.ring{j+7,i} = ring_flat2_roll_angle;

```

```

glove.ring{j+8,i} = ring_transient_roll_angle;
elseif i == 2
glove.ring{j,i} = RMSE_comp_ring_flat1_pitch;
glove.ring{j+1,i} = RMSE_comp_ring_flat2_pitch;
glove.ring{j+2,i} = RMSE_comp_ring_transient_pitch;
glove.ring{j+3,i} = AVGE_comp_ring_flat1_pitch;
glove.ring{j+4,i} = AVGE_comp_ring_flat2_pitch;
glove.ring{j+5,i} = AVGE_comp_ring_transient_pitch;
glove.ring{j+6,i} = ring_flat1_pitch_angle;
glove.ring{j+7,i} = ring_flat2_pitch_angle;
glove.ring{j+8,i} = ring_transient_pitch_angle;
elseif i == 3
glove.ring{j,i} = RMSE_comp_ring_flat1_yaw;
glove.ring{j+1,i} = RMSE_comp_ring_flat2_yaw;
glove.ring{j+2,i} = RMSE_comp_ring_transient_yaw;
glove.ring{j+3,i} = AVGE_comp_ring_flat1_yaw;
glove.ring{j+4,i} = AVGE_comp_ring_flat2_yaw;
glove.ring{j+5,i} = AVGE_comp_ring_transient_yaw;
glove.ring{j+6,i} = ring_flat1_yaw_angle;
glove.ring{j+7,i} = ring_flat2_yaw_angle;
glove.ring{j+8,i} = ring_transient_yaw_angle;
elseif i == 4
glove.ring{j,i} = RMSE_q_ring_flat1_roll;
glove.ring{j+1,i} = RMSE_q_ring_flat2_roll;
glove.ring{j+2,i} = RMSE_q_ring_transient_roll;
glove.ring{j+3,i} = AVGE_q_ring_flat1_roll;
glove.ring{j+4,i} = AVGE_q_ring_flat2_roll;
glove.ring{j+5,i} = AVGE_q_ring_transient_roll;
glove.ring{j+6,i} = ring_flat1_roll_angle;
glove.ring{j+7,i} = ring_flat2_roll_angle;
glove.ring{j+8,i} = ring_transient_roll_angle;
elseif i == 5
glove.ring{j,i} = RMSE_q_ring_flat1_pitch;
glove.ring{j+1,i} = RMSE_q_ring_flat2_pitch;
glove.ring{j+2,i} = RMSE_q_ring_transient_pitch;
glove.ring{j+3,i} = AVGE_q_ring_flat1_pitch;
glove.ring{j+4,i} = AVGE_q_ring_flat2_pitch;
glove.ring{j+5,i} = AVGE_q_ring_transient_pitch;
glove.ring{j+6,i} = ring_flat1_pitch_angle;
glove.ring{j+7,i} = ring_flat2_pitch_angle;
glove.ring{j+8,i} = ring_transient_pitch_angle;
elseif i == 6
glove.ring{j,i} = RMSE_q_ring_flat1_yaw;
glove.ring{j+1,i} = RMSE_q_ring_flat2_yaw;
glove.ring{j+2,i} = RMSE_q_ring_transient_yaw;
glove.ring{j+3,i} = AVGE_q_ring_flat1_yaw;
glove.ring{j+4,i} = AVGE_q_ring_flat2_yaw;
glove.ring{j+5,i} = AVGE_q_ring_transient_yaw;
glove.ring{j+6,i} = ring_flat1_yaw_angle;
glove.ring{j+7,i} = ring_flat2_yaw_angle;
glove.ring{j+8,i} = ring_transient_yaw_angle;
elseif i == 7
glove.ring{j,i} = RMSE_q1_ring_flat1_roll;
glove.ring{j+1,i} = RMSE_q1_ring_flat2_roll;
glove.ring{j+2,i} = RMSE_q1_ring_transient_roll;
glove.ring{j+3,i} = AVGE_q1_ring_flat1_roll;
glove.ring{j+4,i} = AVGE_q1_ring_flat2_roll;

```

```

glove.ring{j+5,i} = AVGE_q1_ring_transient_roll;
glove.ring{j+6,i} = ring_flat1_roll_angle;
glove.ring{j+7,i} = ring_flat2_roll_angle;
glove.ring{j+8,i} = ring_transient_roll_angle;
elseif i == 8
glove.ring{j,i} = RMSE_q1_ring_flat1_pitch;
glove.ring{j+1,i} = RMSE_q1_ring_flat2_pitch;
glove.ring{j+2,i} = RMSE_q1_ring_transient_pitch;
glove.ring{j+3,i} = AVGE_q1_ring_flat1_pitch;
glove.ring{j+4,i} = AVGE_q1_ring_flat2_pitch;
glove.ring{j+5,i} = AVGE_q1_ring_transient_pitch;
glove.ring{j+6,i} = ring_flat1_pitch_angle;
glove.ring{j+7,i} = ring_flat2_pitch_angle;
glove.ring{j+8,i} = ring_transient_pitch_angle;
elseif i == 9
glove.ring{j,i} = RMSE_q1_ring_flat1_yaw;
glove.ring{j+1,i} = RMSE_q1_ring_flat2_yaw;
glove.ring{j+2,i} = RMSE_q1_ring_transient_yaw;
glove.ring{j+3,i} = AVGE_q1_ring_flat1_yaw;
glove.ring{j+4,i} = AVGE_q1_ring_flat2_yaw;
glove.ring{j+5,i} = AVGE_q1_ring_transient_yaw;
glove.ring{j+6,i} = ring_flat1_yaw_angle;
glove.ring{j+7,i} = ring_flat2_yaw_angle;
glove.ring{j+8,i} = ring_transient_yaw_angle;
end
end
end

%%%MIDDLE
if test == 1 || 2 || 3
for i = 1:9
if i == 1
glove.middle{j,i} = RMSE_comp_middle_flat1_roll;
glove.middle{j+1,i} = RMSE_comp_middle_flat2_roll;
glove.middle{j+2,i} = RMSE_comp_middle_transient_roll;
glove.middle{j+3,i} = AVGE_comp_middle_flat1_roll;
glove.middle{j+4,i} = AVGE_comp_middle_flat2_roll;
glove.middle{j+5,i} = AVGE_comp_middle_transient_roll;
glove.middle{j+6,i} = middle_flat1_roll_angle;
glove.middle{j+7,i} = middle_flat2_roll_angle;
glove.middle{j+8,i} = middle_transient_roll_angle;
elseif i == 2
glove.middle{j,i} = RMSE_comp_middle_flat1_pitch;
glove.middle{j+1,i} = RMSE_comp_middle_flat2_pitch;
glove.middle{j+2,i} = RMSE_comp_middle_transient_pitch;
glove.middle{j+3,i} = AVGE_comp_middle_flat1_pitch;
glove.middle{j+4,i} = AVGE_comp_middle_flat2_pitch;
glove.middle{j+5,i} = AVGE_comp_middle_transient_pitch;
glove.middle{j+6,i} = middle_flat1_pitch_angle;
glove.middle{j+7,i} = middle_flat2_pitch_angle;
glove.middle{j+8,i} = middle_transient_pitch_angle;
elseif i == 3
glove.middle{j,i} = RMSE_comp_middle_flat1_yaw;
glove.middle{j+1,i} = RMSE_comp_middle_flat2_yaw;
glove.middle{j+2,i} = RMSE_comp_middle_transient_yaw;

```

```

glove.middle{j+3,i} = AVGE_comp_middle_flat1_yaw;
glove.middle{j+4,i} = AVGE_comp_middle_flat2_yaw;
glove.middle{j+5,i} = AVGE_comp_middle_transient_yaw;
glove.middle{j+6,i} = middle_flat1_yaw_angle;
glove.middle{j+7,i} = middle_flat2_yaw_angle;
glove.middle{j+8,i} = middle_transient_yaw_angle;
elseif i == 4
glove.middle{j,i} = RMSE_q_middle_flat1_roll;
glove.middle{j+1,i} = RMSE_q_middle_flat2_roll;
glove.middle{j+2,i} = RMSE_q_middle_transient_roll;
glove.middle{j+3,i} = AVGE_q_middle_flat1_roll;
glove.middle{j+4,i} = AVGE_q_middle_flat2_roll;
glove.middle{j+5,i} = AVGE_q_middle_transient_roll;
glove.middle{j+6,i} = middle_flat1_roll_angle;
glove.middle{j+7,i} = middle_flat2_roll_angle;
glove.middle{j+8,i} = middle_transient_roll_angle;
elseif i == 5
glove.middle{j,i} = RMSE_q_middle_flat1_pitch;
glove.middle{j+1,i} = RMSE_q_middle_flat2_pitch;
glove.middle{j+2,i} = RMSE_q_middle_transient_pitch;
glove.middle{j+3,i} = AVGE_q_middle_flat1_pitch;
glove.middle{j+4,i} = AVGE_q_middle_flat2_pitch;
glove.middle{j+5,i} = AVGE_q_middle_transient_pitch;
glove.middle{j+6,i} = middle_flat1_pitch_angle;
glove.middle{j+7,i} = middle_flat2_pitch_angle;
glove.middle{j+8,i} = middle_transient_pitch_angle;
elseif i == 6
glove.middle{j,i} = RMSE_q_middle_flat1_yaw;
glove.middle{j+1,i} = RMSE_q_middle_flat2_yaw;
glove.middle{j+2,i} = RMSE_q_middle_transient_yaw;
glove.middle{j+3,i} = AVGE_q_middle_flat1_yaw;
glove.middle{j+4,i} = AVGE_q_middle_flat2_yaw;
glove.middle{j+5,i} = AVGE_q_middle_transient_yaw;
glove.middle{j+6,i} = middle_flat1_yaw_angle;
glove.middle{j+7,i} = middle_flat2_yaw_angle;
glove.middle{j+8,i} = middle_transient_yaw_angle;
elseif i == 7
glove.middle{j,i} = RMSE_q1_middle_flat1_roll;
glove.middle{j+1,i} = RMSE_q1_middle_flat2_roll;
glove.middle{j+2,i} = RMSE_q1_middle_transient_roll;
glove.middle{j+3,i} = AVGE_q1_middle_flat1_roll;
glove.middle{j+4,i} = AVGE_q1_middle_flat2_roll;
glove.middle{j+5,i} = AVGE_q1_middle_transient_roll;
glove.middle{j+6,i} = middle_flat1_roll_angle;
glove.middle{j+7,i} = middle_flat2_roll_angle;
glove.middle{j+8,i} = middle_transient_roll_angle;
elseif i == 8
glove.middle{j,i} = RMSE_q1_middle_flat1_pitch;
glove.middle{j+1,i} = RMSE_q1_middle_flat2_pitch;
glove.middle{j+2,i} = RMSE_q1_middle_transient_pitch;
glove.middle{j+3,i} = AVGE_q1_middle_flat1_pitch;
glove.middle{j+4,i} = AVGE_q1_middle_flat2_pitch;
glove.middle{j+5,i} = AVGE_q1_middle_transient_pitch;
glove.middle{j+6,i} = middle_flat1_pitch_angle;
glove.middle{j+7,i} = middle_flat2_pitch_angle;
glove.middle{j+8,i} = middle_transient_pitch_angle;
elseif i == 9

```

```

glove.middle{j,i} = RMSE_q1_middle_flat1_yaw;
glove.middle{j+1,i} = RMSE_q1_middle_flat2_yaw;
glove.middle{j+2,i} = RMSE_q1_middle_transient_yaw;
glove.middle{j+3,i} = AVGE_q1_middle_flat1_yaw;
glove.middle{j+4,i} = AVGE_q1_middle_flat2_yaw;
glove.middle{j+5,i} = AVGE_q1_middle_transient_yaw;
glove.middle{j+6,i} = middle_flat1_yaw_angle;
glove.middle{j+7,i} = middle_flat2_yaw_angle;
glove.middle{j+8,i} = middle_transient_yaw_angle;
end
end
end

%%% INDEX
if test == 1 || 2 || 3
    for i = 1:9
        if i == 1
            glove.index{j,i} = RMSE_comp_index_flat1_roll;
            glove.index{j+1,i} = RMSE_comp_index_flat2_roll;
            glove.index{j+2,i} = RMSE_comp_index_transient_roll;
            glove.index{j+3,i} = AVGE_comp_index_flat1_roll;
            glove.index{j+4,i} = AVGE_comp_index_flat2_roll;
            glove.index{j+5,i} = AVGE_comp_index_transient_roll;
            glove.index{j+6,i} = index_flat1_roll_angle;
            glove.index{j+7,i} = index_flat2_roll_angle;
            glove.index{j+8,i} = index_transient_roll_angle;
        elseif i == 2
            glove.index{j,i} = RMSE_comp_index_flat1_pitch;
            glove.index{j+1,i} = RMSE_comp_index_flat2_pitch;
            glove.index{j+2,i} = RMSE_comp_index_transient_pitch;
            glove.index{j+3,i} = AVGE_comp_index_flat1_pitch;
            glove.index{j+4,i} = AVGE_comp_index_flat2_pitch;
            glove.index{j+5,i} = AVGE_comp_index_transient_pitch;
            glove.index{j+6,i} = index_flat1_pitch_angle;
            glove.index{j+7,i} = index_flat2_pitch_angle;
            glove.index{j+8,i} = index_transient_pitch_angle;
        elseif i == 3
            glove.index{j,i} = RMSE_comp_index_flat1_yaw;
            glove.index{j+1,i} = RMSE_comp_index_flat2_yaw;
            glove.index{j+2,i} = RMSE_comp_index_transient_yaw;
            glove.index{j+3,i} = AVGE_comp_index_flat1_yaw;
            glove.index{j+4,i} = AVGE_comp_index_flat2_yaw;
            glove.index{j+5,i} = AVGE_comp_index_transient_yaw;
            glove.index{j+6,i} = index_flat1_yaw_angle;
            glove.index{j+7,i} = index_flat2_yaw_angle;
            glove.index{j+8,i} = index_transient_yaw_angle;
        elseif i == 4
            glove.index{j,i} = RMSE_q_index_flat1_roll;
            glove.index{j+1,i} = RMSE_q_index_flat2_roll;
            glove.index{j+2,i} = RMSE_q_index_transient_roll;
            glove.index{j+3,i} = AVGE_q_index_flat1_roll;
            glove.index{j+4,i} = AVGE_q_index_flat2_roll;
            glove.index{j+5,i} = AVGE_q_index_transient_roll;
            glove.index{j+6,i} = index_flat1_roll_angle;
            glove.index{j+7,i} = index_flat2_roll_angle;
            glove.index{j+8,i} = index_transient_roll_angle;
        elseif i == 5

```

```

glove.index{j,i} = RMSE_q_index_flat1_pitch;
glove.index{j+1,i} = RMSE_q_index_flat2_pitch;
glove.index{j+2,i} = RMSE_q_index_transient_pitch;
glove.index{j+3,i} = AVGE_q_index_flat1_pitch;
glove.index{j+4,i} = AVGE_q_index_flat2_pitch;
glove.index{j+5,i} = AVGE_q_index_transient_pitch;
glove.index{j+6,i} = index_flat1_pitch_angle;
glove.index{j+7,i} = index_flat2_pitch_angle;
glove.index{j+8,i} = index_transient_pitch_angle;
elseif i == 6
glove.index{j,i} = RMSE_q_index_flat1_yaw;
glove.index{j+1,i} = RMSE_q_index_flat2_yaw;
glove.index{j+2,i} = RMSE_q_index_transient_yaw;
glove.index{j+3,i} = AVGE_q_index_flat1_yaw;
glove.index{j+4,i} = AVGE_q_index_flat2_yaw;
glove.index{j+5,i} = AVGE_q_index_transient_yaw;
glove.index{j+6,i} = index_flat1_yaw_angle;
glove.index{j+7,i} = index_flat2_yaw_angle;
glove.index{j+8,i} = index_transient_yaw_angle;
elseif i == 7
glove.index{j,i} = RMSE_q1_index_flat1_roll;
glove.index{j+1,i} = RMSE_q1_index_flat2_roll;
glove.index{j+2,i} = RMSE_q1_index_transient_roll;
glove.index{j+3,i} = AVGE_q1_index_flat1_roll;
glove.index{j+4,i} = AVGE_q1_index_flat2_roll;
glove.index{j+5,i} = AVGE_q1_index_transient_roll;
glove.index{j+6,i} = index_flat1_roll_angle;
glove.index{j+7,i} = index_flat2_roll_angle;
glove.index{j+8,i} = index_transient_roll_angle;
elseif i == 8
glove.index{j,i} = RMSE_q1_index_flat1_pitch;
glove.index{j+1,i} = RMSE_q1_index_flat2_pitch;
glove.index{j+2,i} = RMSE_q1_index_transient_pitch;
glove.index{j+3,i} = AVGE_q1_index_flat1_pitch;
glove.index{j+4,i} = AVGE_q1_index_flat2_pitch;
glove.index{j+5,i} = AVGE_q1_index_transient_pitch;
glove.index{j+6,i} = index_flat1_pitch_angle;
glove.index{j+7,i} = index_flat2_pitch_angle;
glove.index{j+8,i} = index_transient_pitch_angle;
elseif i == 9
glove.index{j,i} = RMSE_q1_index_flat1_yaw;
glove.index{j+1,i} = RMSE_q1_index_flat2_yaw;
glove.index{j+2,i} = RMSE_q1_index_transient_yaw;
glove.index{j+3,i} = AVGE_q1_index_flat1_yaw;
glove.index{j+4,i} = AVGE_q1_index_flat2_yaw;
glove.index{j+5,i} = AVGE_q1_index_transient_yaw;
glove.index{j+6,i} = index_flat1_yaw_angle;
glove.index{j+7,i} = index_flat2_yaw_angle;
glove.index{j+8,i} = index_transient_yaw_angle;
end
end
end

%%% THUMB
if test == 1 || 2 || 3
    for i = 1:9
        if i == 1

```

```

glove.thumb{j,i} = RMSE_comp_thumb_flat1_roll;
glove.thumb{j+1,i} = RMSE_comp_thumb_flat2_roll;
glove.thumb{j+2,i} = RMSE_comp_thumb_transient_roll;
glove.thumb{j+3,i} = AVGE_comp_thumb_flat1_roll;
glove.thumb{j+4,i} = AVGE_comp_thumb_flat2_roll;
glove.thumb{j+5,i} = AVGE_comp_thumb_transient_roll;
glove.thumb{j+6,i} = thumb_flat1_roll_angle;
glove.thumb{j+7,i} = thumb_flat2_roll_angle;
glove.thumb{j+8,i} = thumb_transient_roll_angle;
elseif i == 2
glove.thumb{j,i} = RMSE_comp_thumb_flat1_pitch;
glove.thumb{j+1,i} = RMSE_comp_thumb_flat2_pitch;
glove.thumb{j+2,i} = RMSE_comp_thumb_transient_pitch;
glove.thumb{j+3,i} = AVGE_comp_thumb_flat1_pitch;
glove.thumb{j+4,i} = AVGE_comp_thumb_flat2_pitch;
glove.thumb{j+5,i} = AVGE_comp_thumb_transient_pitch;
glove.thumb{j+6,i} = thumb_flat1_pitch_angle;
glove.thumb{j+7,i} = thumb_flat2_pitch_angle;
glove.thumb{j+8,i} = thumb_transient_pitch_angle;
elseif i == 3
glove.thumb{j,i} = RMSE_comp_thumb_flat1_yaw;
glove.thumb{j+1,i} = RMSE_comp_thumb_flat2_yaw;
glove.thumb{j+2,i} = RMSE_comp_thumb_transient_yaw;
glove.thumb{j+3,i} = AVGE_comp_thumb_flat1_yaw;
glove.thumb{j+4,i} = AVGE_comp_thumb_flat2_yaw;
glove.thumb{j+5,i} = AVGE_comp_thumb_transient_yaw;
glove.thumb{j+6,i} = thumb_flat1_yaw_angle;
glove.thumb{j+7,i} = thumb_flat2_yaw_angle;
glove.thumb{j+8,i} = thumb_transient_yaw_angle;
elseif i == 4
glove.thumb{j,i} = RMSE_q_thumb_flat1_roll;
glove.thumb{j+1,i} = RMSE_q_thumb_flat2_roll;
glove.thumb{j+2,i} = RMSE_q_thumb_transient_roll;
glove.thumb{j+3,i} = AVGE_q_thumb_flat1_roll;
glove.thumb{j+4,i} = AVGE_q_thumb_flat2_roll;
glove.thumb{j+5,i} = AVGE_q_thumb_transient_roll;
glove.thumb{j+6,i} = thumb_flat1_roll_angle;
glove.thumb{j+7,i} = thumb_flat2_roll_angle;
glove.thumb{j+8,i} = thumb_transient_roll_angle;
elseif i == 5
glove.thumb{j,i} = RMSE_q_thumb_flat1_pitch;
glove.thumb{j+1,i} = RMSE_q_thumb_flat2_pitch;
glove.thumb{j+2,i} = RMSE_q_thumb_transient_pitch;
glove.thumb{j+3,i} = AVGE_q_thumb_flat1_pitch;
glove.thumb{j+4,i} = AVGE_q_thumb_flat2_pitch;
glove.thumb{j+5,i} = AVGE_q_thumb_transient_pitch;
glove.thumb{j+6,i} = thumb_flat1_pitch_angle;
glove.thumb{j+7,i} = thumb_flat2_pitch_angle;
glove.thumb{j+8,i} = thumb_transient_pitch_angle;
elseif i == 6
glove.thumb{j,i} = RMSE_q_thumb_flat1_yaw;
glove.thumb{j+1,i} = RMSE_q_thumb_flat2_yaw;
glove.thumb{j+2,i} = RMSE_q_thumb_transient_yaw;
glove.thumb{j+3,i} = AVGE_q_thumb_flat1_yaw;
glove.thumb{j+4,i} = AVGE_q_thumb_flat2_yaw;
glove.thumb{j+5,i} = AVGE_q_thumb_transient_yaw;
glove.thumb{j+6,i} = thumb_flat1_yaw_angle;

```

```

glove.thumb{j+7,i} = thumb_flat2_yaw_angle;
glove.thumb{j+8,i} = thumb_transient_yaw_angle;
elseif i == 7
glove.thumb{j,i} = RMSE_q1_thumb_flat1_roll;
glove.thumb{j+1,i} = RMSE_q1_thumb_flat2_roll;
glove.thumb{j+2,i} = RMSE_q1_thumb_transient_roll;
glove.thumb{j+3,i} = AVGE_q1_thumb_flat1_roll;
glove.thumb{j+4,i} = AVGE_q1_thumb_flat2_roll;
glove.thumb{j+5,i} = AVGE_q1_thumb_transient_roll;
glove.thumb{j+6,i} = thumb_flat1_roll_angle;
glove.thumb{j+7,i} = thumb_flat2_roll_angle;
glove.thumb{j+8,i} = thumb_transient_roll_angle;
elseif i == 8
glove.thumb{j,i} = RMSE_q1_thumb_flat1_pitch;
glove.thumb{j+1,i} = RMSE_q1_thumb_flat2_pitch;
glove.thumb{j+2,i} = RMSE_q1_thumb_transient_pitch;
glove.thumb{j+3,i} = AVGE_q1_thumb_flat1_pitch;
glove.thumb{j+4,i} = AVGE_q1_thumb_flat2_pitch;
glove.thumb{j+5,i} = AVGE_q1_thumb_transient_pitch;
glove.thumb{j+6,i} = thumb_flat1_pitch_angle;
glove.thumb{j+7,i} = thumb_flat2_pitch_angle;
glove.thumb{j+8,i} = thumb_transient_pitch_angle;
elseif i == 9
glove.thumb{j,i} = RMSE_q1_thumb_flat1_yaw;
glove.thumb{j+1,i} = RMSE_q1_thumb_flat2_yaw;
glove.thumb{j+2,i} = RMSE_q1_thumb_transient_yaw;
glove.thumb{j+3,i} = AVGE_q1_thumb_flat1_yaw;
glove.thumb{j+4,i} = AVGE_q1_thumb_flat2_yaw;
glove.thumb{j+5,i} = AVGE_q1_thumb_transient_yaw;
glove.thumb{j+6,i} = thumb_flat1_yaw_angle;
glove.thumb{j+7,i} = thumb_flat2_yaw_angle;
glove.thumb{j+8,i} = thumb_transient_yaw_angle;
end
end
end

%%% HAND
if test == 1 || 2 || 3
for i = 1:9
if i == 1
glove.hand{j,i} = RMSE_comp_hand_flat1_roll;
glove.hand{j+1,i} = RMSE_comp_hand_flat2_roll;
glove.hand{j+2,i} = RMSE_comp_hand_transient_roll;
glove.hand{j+3,i} = AVGE_comp_hand_flat1_roll;
glove.hand{j+4,i} = AVGE_comp_hand_flat2_roll;
glove.hand{j+5,i} = AVGE_comp_hand_transient_roll;
glove.hand{j+6,i} = hand_flat1_roll_angle;
glove.hand{j+7,i} = hand_flat2_roll_angle;
glove.hand{j+8,i} = hand_transient_roll_angle;
elseif i == 2
glove.hand{j,i} = RMSE_comp_hand_flat1_pitch;
glove.hand{j+1,i} = RMSE_comp_hand_flat2_pitch;
glove.hand{j+2,i} = RMSE_comp_hand_transient_pitch;
glove.hand{j+3,i} = AVGE_comp_hand_flat1_pitch;
glove.hand{j+4,i} = AVGE_comp_hand_flat2_pitch;
glove.hand{j+5,i} = AVGE_comp_hand_transient_pitch;
glove.hand{j+6,i} = hand_flat1_pitch_angle;

```

```

glove.hand{j+7,i} = hand_flat2_pitch_angle;
glove.hand{j+8,i} = hand_transient_pitch_angle;
elseif i == 3
glove.hand{j,i} = RMSE_comp_hand_flat1_yaw;
glove.hand{j+1,i} = RMSE_comp_hand_flat2_yaw;
glove.hand{j+2,i} = RMSE_comp_hand_transient_yaw;
glove.hand{j+3,i} = AVGE_comp_hand_flat1_yaw;
glove.hand{j+4,i} = AVGE_comp_hand_flat2_yaw;
glove.hand{j+5,i} = AVGE_comp_hand_transient_yaw;
glove.hand{j+6,i} = hand_flat1_yaw_angle;
glove.hand{j+7,i} = hand_flat2_yaw_angle;
glove.hand{j+8,i} = hand_transient_yaw_angle;
elseif i == 4
glove.hand{j,i} = RMSE_q_hand_flat1_roll;
glove.hand{j+1,i} = RMSE_q_hand_flat2_roll;
glove.hand{j+2,i} = RMSE_q_hand_transient_roll;
glove.hand{j+3,i} = AVGE_q_hand_flat1_roll;
glove.hand{j+4,i} = AVGE_q_hand_flat2_roll;
glove.hand{j+5,i} = AVGE_q_hand_transient_roll;
glove.hand{j+6,i} = hand_flat1_roll_angle;
glove.hand{j+7,i} = hand_flat2_roll_angle;
glove.hand{j+8,i} = hand_transient_roll_angle;
elseif i == 5
glove.hand{j,i} = RMSE_q_hand_flat1_pitch;
glove.hand{j+1,i} = RMSE_q_hand_flat2_pitch;
glove.hand{j+2,i} = RMSE_q_hand_transient_pitch;
glove.hand{j+3,i} = AVGE_q_hand_flat1_pitch;
glove.hand{j+4,i} = AVGE_q_hand_flat2_pitch;
glove.hand{j+5,i} = AVGE_q_hand_transient_pitch;
glove.hand{j+6,i} = hand_flat1_pitch_angle;
glove.hand{j+7,i} = hand_flat2_pitch_angle;
glove.hand{j+8,i} = hand_transient_pitch_angle;
elseif i == 6
glove.hand{j,i} = RMSE_q_hand_flat1_yaw;
glove.hand{j+1,i} = RMSE_q_hand_flat2_yaw;
glove.hand{j+2,i} = RMSE_q_hand_transient_yaw;
glove.hand{j+3,i} = AVGE_q_hand_flat1_yaw;
glove.hand{j+4,i} = AVGE_q_hand_flat2_yaw;
glove.hand{j+5,i} = AVGE_q_hand_transient_yaw;
glove.hand{j+6,i} = hand_flat1_yaw_angle;
glove.hand{j+7,i} = hand_flat2_yaw_angle;
glove.hand{j+8,i} = hand_transient_yaw_angle;
elseif i == 7
glove.hand{j,i} = RMSE_q1_hand_flat1_roll;
glove.hand{j+1,i} = RMSE_q1_hand_flat2_roll;
glove.hand{j+2,i} = RMSE_q1_hand_transient_roll;
glove.hand{j+3,i} = AVGE_q1_hand_flat1_roll;
glove.hand{j+4,i} = AVGE_q1_hand_flat2_roll;
glove.hand{j+5,i} = AVGE_q1_hand_transient_roll;
glove.hand{j+6,i} = hand_flat1_roll_angle;
glove.hand{j+7,i} = hand_flat2_roll_angle;
glove.hand{j+8,i} = hand_transient_roll_angle;
elseif i == 8
glove.hand{j,i} = RMSE_q1_hand_flat1_pitch;
glove.hand{j+1,i} = RMSE_q1_hand_flat2_pitch;
glove.hand{j+2,i} = RMSE_q1_hand_transient_pitch;
glove.hand{j+3,i} = AVGE_q1_hand_flat1_pitch;

```

```

    glove.hand{j+4,i} = AVGE_q1_hand_flat2_pitch;
    glove.hand{j+5,i} = AVGE_q1_hand_transient_pitch;
    glove.hand{j+6,i} = hand_flat1_pitch_angle;
    glove.hand{j+7,i} = hand_flat2_pitch_angle;
    glove.hand{j+8,i} = hand_transient_pitch_angle;
    elseif i == 9
    glove.hand{j,i} = RMSE_q1_hand_flat1_yaw;
    glove.hand{j+1,i} = RMSE_q1_hand_flat2_yaw;
    glove.hand{j+2,i} = RMSE_q1_hand_transient_yaw;
    glove.hand{j+3,i} = AVGE_q1_hand_flat1_yaw;
    glove.hand{j+4,i} = AVGE_q1_hand_flat2_yaw;
    glove.hand{j+5,i} = AVGE_q1_hand_transient_yaw;
    glove.hand{j+6,i} = hand_flat1_yaw_angle;
    glove.hand{j+7,i} = hand_flat2_yaw_angle;
    glove.hand{j+8,i} = hand_transient_yaw_angle;
    end
end
end
end

```

f) GLOVEtesting.m

Every glove variable is named for each orientation during experiments, this script puts them together and orders them.

```

GLOVE = [glove1 glove2 glove3 glove4 glove5 glove6 glove7 glove8 glove9
glove10 glove11 glove12 glove13 glove14]; %Orders all of the glove vars
together
rms_comp = zeros(length(GLOVE),162);
rms_roll = zeros(length(GLOVE),54);
rms_pitch = zeros(length(GLOVE),54);
rms_yaw = zeros(length(GLOVE),54);
rms_transient = zeros(length(GLOVE),54);
rms_flats = zeros(length(GLOVE), 108);

avge_roll = zeros(length(GLOVE),54);
avge_pitch = zeros(length(GLOVE),54);
avge_yaw = zeros(length(GLOVE),54);
avge_transient = zeros(length(GLOVE),54);
avge_flats = zeros(length(GLOVE), 108);

angles_roll = zeros(length(GLOVE),54);
angles_pitch = zeros(length(GLOVE),54);
angles_yaw = zeros(length(GLOVE),54);
angles_transient = zeros(length(GLOVE),54);
angles_flats = zeros(length(GLOVE), 108);

rms_q = zeros(length(GLOVE),162);
rms_q1 = zeros(length(GLOVE),162);
avge_comp = zeros(length(GLOVE),162);
avge_q = zeros(length(GLOVE),162);
avge_q1 = zeros(length(GLOVE),162);
angles = zeros(length(GLOVE),162);

```

```

for i = 1:length(GLOVE)
    %%RMS values for all tests
    rms_comp(i,1:27) = cell2mat([GLOVE(i).pinky(1:3,1)'
GLOVE(i).pinky(10:12,1)' GLOVE(i).pinky(19:21,1)' GLOVE(i).pinky(1:3,2)'
GLOVE(i).pinky(10:12,2)' GLOVE(i).pinky(19:21,2)' GLOVE(i).pinky(1:3,3)'
GLOVE(i).pinky(10:12,3)' GLOVE(i).pinky(19:21,3)']]);
    rms_comp(i,28:54) = cell2mat([GLOVE(i).ring(1:3,1)'
GLOVE(i).ring(10:12,1)' GLOVE(i).ring(19:21,1)' GLOVE(i).ring(1:3,2)'
GLOVE(i).ring(10:12,2)' GLOVE(i).ring(19:21,2)' GLOVE(i).ring(1:3,3)'
GLOVE(i).ring(10:12,3)' GLOVE(i).ring(19:21,3)']]);
    rms_comp(i,55:81) = cell2mat([GLOVE(i).middle(1:3,1)'
GLOVE(i).middle(10:12,1)' GLOVE(i).middle(19:21,1)' GLOVE(i).middle(1:3,2)'
GLOVE(i).middle(10:12,2)' GLOVE(i).middle(19:21,2)' GLOVE(i).middle(1:3,3)'
GLOVE(i).middle(10:12,3)' GLOVE(i).middle(19:21,3)']]);
    rms_comp(i,82:108) = cell2mat([GLOVE(i).index(1:3,1)'
GLOVE(i).index(10:12,1)' GLOVE(i).index(19:21,1)' GLOVE(i).index(1:3,2)'
GLOVE(i).index(10:12,2)' GLOVE(i).index(19:21,2)' GLOVE(i).index(1:3,3)'
GLOVE(i).index(10:12,3)' GLOVE(i).index(19:21,3)']]);
    rms_comp(i,109:135) = cell2mat([GLOVE(i).thumb(1:3,1)'
GLOVE(i).thumb(10:12,1)' GLOVE(i).thumb(19:21,1)' GLOVE(i).thumb(1:3,2)'
GLOVE(i).thumb(10:12,2)' GLOVE(i).thumb(19:21,2)' GLOVE(i).thumb(1:3,3)'
GLOVE(i).thumb(10:12,3)' GLOVE(i).thumb(19:21,3)']]);
    rms_comp(i,136:162) = cell2mat([GLOVE(i).hand(1:3,1)'
GLOVE(i).hand(10:12,1)' GLOVE(i).hand(19:21,1)' GLOVE(i).hand(1:3,2)'
GLOVE(i).hand(10:12,2)' GLOVE(i).hand(19:21,2)' GLOVE(i).hand(1:3,3)'
GLOVE(i).hand(10:12,3)' GLOVE(i).hand(19:21,3)']]);

    %%RMS ROLL
    rms_roll(i,1:9) = cell2mat([GLOVE(i).pinky(1:3,1)'
GLOVE(i).pinky(10:12,1)' GLOVE(i).pinky(19:21,1)']]);
    rms_roll(i,10:18) = cell2mat([GLOVE(i).ring(1:3,1)'
GLOVE(i).ring(10:12,1)' GLOVE(i).ring(19:21,1)']]);
    rms_roll(i,19:27) = cell2mat([GLOVE(i).middle(1:3,1)'
GLOVE(i).middle(10:12,1)' GLOVE(i).middle(19:21,1)']]);
    rms_roll(i,28:36) = cell2mat([GLOVE(i).index(1:3,1)'
GLOVE(i).index(10:12,1)' GLOVE(i).index(19:21,1)']]);
    rms_roll(i,37:45) = cell2mat([GLOVE(i).thumb(1:3,1)'
GLOVE(i).thumb(10:12,1)' GLOVE(i).thumb(19:21,1)']]);
    rms_roll(i,46:54) = cell2mat([GLOVE(i).hand(1:3,1)'
GLOVE(i).hand(10:12,1)' GLOVE(i).hand(19:21,1)']]);

    %%RMS PITCH
    rms_pitch(i,1:9) = cell2mat([GLOVE(i).pinky(1:3,2)'
GLOVE(i).pinky(10:12,2)' GLOVE(i).pinky(19:21,2)']]);
    rms_pitch(i,10:18) = cell2mat([GLOVE(i).ring(1:3,2)'
GLOVE(i).ring(10:12,2)' GLOVE(i).ring(19:21,2)']]);
    rms_pitch(i,19:27) = cell2mat([GLOVE(i).middle(1:3,2)'
GLOVE(i).middle(10:12,2)' GLOVE(i).middle(19:21,2)']]);
    rms_pitch(i,28:36) = cell2mat([GLOVE(i).index(1:3,2)'
GLOVE(i).index(10:12,2)' GLOVE(i).index(19:21,2)']]);
    rms_pitch(i,37:45) = cell2mat([GLOVE(i).thumb(1:3,2)'
GLOVE(i).thumb(10:12,2)' GLOVE(i).thumb(19:21,2)']]);
    rms_pitch(i,46:54) = cell2mat([GLOVE(i).hand(1:3,2)'
GLOVE(i).hand(10:12,2)' GLOVE(i).hand(19:21,2)']]);

    %%RMS YAW
    rms_yaw(i,1:9) = cell2mat([GLOVE(i).pinky(1:3,3)'
GLOVE(i).pinky(10:12,3)' GLOVE(i).pinky(19:21,3)']]);

```

```

    rms_yaw(i,10:18) = cell2mat([GLOVE(i).ring(1:3,3)'
GLOVE(i).ring(10:12,3)' GLOVE(i).ring(19:21,3)']);
    rms_yaw(i,19:27) = cell2mat([GLOVE(i).middle(1:3,3)'
GLOVE(i).middle(10:12,3)' GLOVE(i).middle(19:21,3)']);
    rms_yaw(i,28:36) = cell2mat([GLOVE(i).index(1:3,3)'
GLOVE(i).index(10:12,3)' GLOVE(i).index(19:21,3)']);
    rms_yaw(i,37:45) = cell2mat([GLOVE(i).thumb(1:3,3)'
GLOVE(i).thumb(10:12,3)' GLOVE(i).thumb(19:21,3)']);
    rms_yaw(i,46:54) = cell2mat([GLOVE(i).hand(1:3,3)'
GLOVE(i).hand(10:12,3)' GLOVE(i).hand(19:21,3)']);

    %RMS FLATS
    rms_flats(i,1:18) = cell2mat([GLOVE(i).pinky(1:2,1)'
GLOVE(i).pinky(10:11,1)' GLOVE(i).pinky(19:20,1)' GLOVE(i).pinky(1:2,2)'
GLOVE(i).pinky(10:11,2)' GLOVE(i).pinky(19:20,2)' GLOVE(i).pinky(1:2,3)'
GLOVE(i).pinky(10:11,3)' GLOVE(i).pinky(19:20,3)']);
    rms_flats(i,19:36) = cell2mat([GLOVE(i).ring(1:2,1)'
GLOVE(i).ring(10:11,1)' GLOVE(i).ring(19:20,1)' GLOVE(i).ring(1:2,2)'
GLOVE(i).ring(10:11,2)' GLOVE(i).ring(19:20,2)' GLOVE(i).ring(1:2,3)'
GLOVE(i).ring(10:11,3)' GLOVE(i).ring(19:20,3)']);
    rms_flats(i,37:54) = cell2mat([GLOVE(i).middle(1:2,1)'
GLOVE(i).middle(10:11,1)' GLOVE(i).middle(19:20,1)' GLOVE(i).middle(1:2,2)'
GLOVE(i).middle(10:11,2)' GLOVE(i).middle(19:20,2)' GLOVE(i).middle(1:2,3)'
GLOVE(i).middle(10:11,3)' GLOVE(i).middle(19:20,3)']);
    rms_flats(i,55:72) = cell2mat([GLOVE(i).index(1:2,1)'
GLOVE(i).index(10:11,1)' GLOVE(i).index(19:20,1)' GLOVE(i).index(1:2,2)'
GLOVE(i).index(10:11,2)' GLOVE(i).index(19:20,2)' GLOVE(i).index(1:2,3)'
GLOVE(i).index(10:11,3)' GLOVE(i).index(19:20,3)']);
    rms_flats(i,73:90) = cell2mat([GLOVE(i).thumb(1:2,1)'
GLOVE(i).thumb(10:11,1)' GLOVE(i).thumb(19:20,1)' GLOVE(i).thumb(1:2,2)'
GLOVE(i).thumb(10:11,2)' GLOVE(i).thumb(19:20,2)' GLOVE(i).thumb(1:2,3)'
GLOVE(i).thumb(10:11,3)' GLOVE(i).thumb(19:20,3)']);
    rms_flats(i,91:108) = cell2mat([GLOVE(i).hand(1:2,1)'
GLOVE(i).hand(10:11,1)' GLOVE(i).hand(19:20,1)' GLOVE(i).hand(1:2,2)'
GLOVE(i).hand(10:11,2)' GLOVE(i).hand(19:20,2)' GLOVE(i).hand(1:2,3)'
GLOVE(i).hand(10:11,3)' GLOVE(i).hand(19:20,3)']);
    %RMS TRANSIENT
    rms_transient(i,1:9) = rms_comp(i,3:3:27);
    rms_transient(i,10:18) = rms_comp(i,30:3:54);
    rms_transient(i,19:27) = rms_comp(i,57:3:81);
    rms_transient(i,28:36) = rms_comp(i,84:3:108);
    rms_transient(i,37:45) = rms_comp(i,111:3:135);
    rms_transient(i,46:54) = rms_comp(i,138:3:162);
    rms_q(i,1:27) = cell2mat([GLOVE(i).pinky(1:3,4)' GLOVE(i).pinky(10:12,4)'
GLOVE(i).pinky(19:21,4)' GLOVE(i).pinky(1:3,5)' GLOVE(i).pinky(10:12,5)'
GLOVE(i).pinky(19:21,5)' GLOVE(i).pinky(1:3,6)' GLOVE(i).pinky(10:12,6)'
GLOVE(i).pinky(19:21,6)']);
    rms_q(i,28:54) = cell2mat([GLOVE(i).ring(1:3,4)' GLOVE(i).ring(10:12,4)'
GLOVE(i).ring(19:21,4)' GLOVE(i).ring(1:3,5)' GLOVE(i).ring(10:12,5)'
GLOVE(i).ring(19:21,5)' GLOVE(i).ring(1:3,6)' GLOVE(i).ring(10:12,6)'
GLOVE(i).ring(19:21,6)']);
    rms_q(i,55:81) = cell2mat([GLOVE(i).middle(1:3,4)'
GLOVE(i).middle(10:12,4)' GLOVE(i).middle(19:21,4)' GLOVE(i).middle(1:3,5)'
GLOVE(i).middle(10:12,5)' GLOVE(i).middle(19:21,5)' GLOVE(i).middle(1:3,6)'
GLOVE(i).middle(10:12,6)' GLOVE(i).middle(19:21,6)']);
    rms_q(i,82:108) = cell2mat([GLOVE(i).index(1:3,4)'
GLOVE(i).index(10:12,4)' GLOVE(i).index(19:21,4)' GLOVE(i).index(1:3,5)'

```

```

GLOVE(i).index(10:12,5)' GLOVE(i).index(19:21,5)' GLOVE(i).index(1:3,6)'
GLOVE(i).index(10:12,6)' GLOVE(i).index(19:21,6)']];
rms_q(i,109:135) = cell2mat([GLOVE(i).thumb(1:3,4)'
GLOVE(i).thumb(10:12,4)' GLOVE(i).thumb(19:21,4)' GLOVE(i).thumb(1:3,5)'
GLOVE(i).thumb(10:12,5)' GLOVE(i).thumb(19:21,5)' GLOVE(i).thumb(1:3,6)'
GLOVE(i).thumb(10:12,6)' GLOVE(i).thumb(19:21,6)']]);
rms_q(i,136:162) = cell2mat([GLOVE(i).hand(1:3,4)'
GLOVE(i).hand(10:12,4)' GLOVE(i).hand(19:21,4)' GLOVE(i).hand(1:3,5)'
GLOVE(i).hand(10:12,5)' GLOVE(i).hand(19:21,5)' GLOVE(i).hand(1:3,6)'
GLOVE(i).hand(10:12,6)' GLOVE(i).hand(19:21,6)']]);
rms_q1(i,1:27) = cell2mat([GLOVE(i).pinky(1:3,7)'
GLOVE(i).pinky(10:12,7)' GLOVE(i).pinky(19:21,7)' GLOVE(i).pinky(1:3,8)'
GLOVE(i).pinky(10:12,8)' GLOVE(i).pinky(19:21,8)' GLOVE(i).pinky(1:3,9)'
GLOVE(i).pinky(10:12,9)' GLOVE(i).pinky(19:21,9)']]);
rms_q1(i,28:54) = cell2mat([GLOVE(i).ring(1:3,7)' GLOVE(i).ring(10:12,7)'
GLOVE(i).ring(19:21,7)' GLOVE(i).ring(1:3,8)' GLOVE(i).ring(10:12,8)'
GLOVE(i).ring(19:21,8)' GLOVE(i).ring(1:3,9)' GLOVE(i).ring(10:12,9)'
GLOVE(i).ring(19:21,9)']]);
rms_q1(i,55:81) = cell2mat([GLOVE(i).middle(1:3,7)'
GLOVE(i).middle(10:12,7)' GLOVE(i).middle(19:21,7)' GLOVE(i).middle(1:3,8)'
GLOVE(i).middle(10:12,8)' GLOVE(i).middle(19:21,8)' GLOVE(i).middle(1:3,9)'
GLOVE(i).middle(10:12,9)' GLOVE(i).middle(19:21,9)']]);
rms_q1(i,82:108) = cell2mat([GLOVE(i).index(1:3,7)'
GLOVE(i).index(10:12,7)' GLOVE(i).index(19:21,7)' GLOVE(i).index(1:3,8)'
GLOVE(i).index(10:12,8)' GLOVE(i).index(19:21,8)' GLOVE(i).index(1:3,9)'
GLOVE(i).index(10:12,9)' GLOVE(i).index(19:21,9)']]);
rms_q1(i,109:135) = cell2mat([GLOVE(i).thumb(1:3,7)'
GLOVE(i).thumb(10:12,7)' GLOVE(i).thumb(19:21,7)' GLOVE(i).thumb(1:3,8)'
GLOVE(i).thumb(10:12,8)' GLOVE(i).thumb(19:21,8)' GLOVE(i).thumb(1:3,9)'
GLOVE(i).thumb(10:12,9)' GLOVE(i).thumb(19:21,9)']]);
rms_q1(i,136:162) = cell2mat([GLOVE(i).hand(1:3,7)'
GLOVE(i).hand(10:12,7)' GLOVE(i).hand(19:21,7)' GLOVE(i).hand(1:3,8)'
GLOVE(i).hand(10:12,8)' GLOVE(i).hand(19:21,8)' GLOVE(i).hand(1:3,9)'
GLOVE(i).hand(10:12,9)' GLOVE(i).hand(19:21,9)']]);
%%AVGE
avge_comp(i,1:27) = cell2mat([GLOVE(i).pinky(4:6,1)'
GLOVE(i).pinky(13:15,1)' GLOVE(i).pinky(22:24,1)' GLOVE(i).pinky(4:6,2)'
GLOVE(i).pinky(13:15,2)' GLOVE(i).pinky(22:24,2)' GLOVE(i).pinky(4:6,3)'
GLOVE(i).pinky(13:15,3)' GLOVE(i).pinky(22:24,3)']]);
avge_comp(i,28:54) = cell2mat([GLOVE(i).ring(4:6,1)'
GLOVE(i).ring(13:15,1)' GLOVE(i).ring(22:24,1)' GLOVE(i).ring(4:6,2)'
GLOVE(i).ring(13:15,2)' GLOVE(i).ring(22:24,2)' GLOVE(i).ring(4:6,3)'
GLOVE(i).ring(13:15,3)' GLOVE(i).ring(22:24,3)']]);
avge_comp(i,55:81) = cell2mat([GLOVE(i).middle(4:6,1)'
GLOVE(i).middle(13:15,1)' GLOVE(i).middle(22:24,1)' GLOVE(i).middle(4:6,2)'
GLOVE(i).middle(13:15,2)' GLOVE(i).middle(22:24,2)' GLOVE(i).middle(4:6,3)'
GLOVE(i).middle(13:15,3)' GLOVE(i).middle(22:24,3)']]);
avge_comp(i,82:108) = cell2mat([GLOVE(i).index(4:6,1)'
GLOVE(i).index(13:15,1)' GLOVE(i).index(22:24,1)' GLOVE(i).index(4:6,2)'
GLOVE(i).index(13:15,2)' GLOVE(i).index(22:24,2)' GLOVE(i).index(4:6,3)'
GLOVE(i).index(13:15,3)' GLOVE(i).index(22:24,3)']]);
avge_comp(i,109:135) = cell2mat([GLOVE(i).thumb(4:6,1)'
GLOVE(i).thumb(13:15,1)' GLOVE(i).thumb(22:24,1)' GLOVE(i).thumb(4:6,2)'
GLOVE(i).thumb(13:15,2)' GLOVE(i).thumb(22:24,2)' GLOVE(i).thumb(4:6,3)'
GLOVE(i).thumb(13:15,3)' GLOVE(i).thumb(22:24,3)']]);
avge_comp(i,136:162) = cell2mat([GLOVE(i).hand(4:6,1)'
GLOVE(i).hand(13:15,1)' GLOVE(i).hand(22:24,1)' GLOVE(i).hand(4:6,2)'

```

```

GLOVE(i).hand(13:15,2)' GLOVE(i).hand(22:24,2)' GLOVE(i).hand(4:6,3)'
GLOVE(i).hand(13:15,3)' GLOVE(i).hand(22:24,3)']];
    %AVGE ROLL
    avge_roll(i,1:9) = cell2mat([GLOVE(i).pinky(4:6,1)'
GLOVE(i).pinky(13:15,1)' GLOVE(i).pinky(22:24,1)']]);
    avge_roll(i,10:18) = cell2mat([GLOVE(i).ring(4:6,1)'
GLOVE(i).ring(13:15,1)' GLOVE(i).ring(22:24,1)']]);
    avge_roll(i,19:27) = cell2mat([GLOVE(i).middle(4:6,1)'
GLOVE(i).middle(13:15,1)' GLOVE(i).middle(22:24,1)']]);
    avge_roll(i,28:36) = cell2mat([GLOVE(i).index(4:6,1)'
GLOVE(i).index(13:15,1)' GLOVE(i).index(22:24,1)']]);
    avge_roll(i,37:45) = cell2mat([GLOVE(i).thumb(4:6,1)'
GLOVE(i).thumb(13:15,1)' GLOVE(i).thumb(22:24,1)']]);
    avge_roll(i,46:54) = cell2mat([GLOVE(i).hand(4:6,1)'
GLOVE(i).hand(13:15,1)' GLOVE(i).hand(22:24,1)']]);

    %AVGE PITCH
    avge_pitch(i,1:9) = cell2mat([GLOVE(i).pinky(4:6,2)'
GLOVE(i).pinky(13:15,2)' GLOVE(i).pinky(22:24,2)']]);
    avge_pitch(i,10:18) = cell2mat([GLOVE(i).ring(4:6,2)'
GLOVE(i).ring(13:15,2)' GLOVE(i).ring(22:24,2)']]);
    avge_pitch(i,19:27) = cell2mat([GLOVE(i).middle(4:6,2)'
GLOVE(i).middle(13:15,2)' GLOVE(i).middle(22:24,2)']]);
    avge_pitch(i,28:36) = cell2mat([GLOVE(i).index(4:6,2)'
GLOVE(i).index(13:15,2)' GLOVE(i).index(22:24,2)']]);
    avge_pitch(i,37:45) = cell2mat([GLOVE(i).thumb(4:6,2)'
GLOVE(i).thumb(13:15,2)' GLOVE(i).thumb(22:24,2)']]);
    avge_pitch(i,46:54) = cell2mat([GLOVE(i).hand(4:6,2)'
GLOVE(i).hand(13:15,2)' GLOVE(i).hand(22:24,2)']]);
    %AVGE YAW
    avge_yaw(i,1:9) = cell2mat([GLOVE(i).pinky(4:6,3)'
GLOVE(i).pinky(13:15,3)' GLOVE(i).pinky(22:24,3)']]);
    avge_yaw(i,10:18) = cell2mat([GLOVE(i).ring(4:6,3)'
GLOVE(i).ring(13:15,3)' GLOVE(i).ring(22:24,3)']]);
    avge_yaw(i,19:27) = cell2mat([GLOVE(i).middle(4:6,3)'
GLOVE(i).middle(13:15,3)' GLOVE(i).middle(22:24,3)']]);
    avge_yaw(i,28:36) = cell2mat([GLOVE(i).index(4:6,3)'
GLOVE(i).index(13:15,3)' GLOVE(i).index(22:24,3)']]);
    avge_yaw(i,37:45) = cell2mat([GLOVE(i).thumb(4:6,3)'
GLOVE(i).thumb(13:15,3)' GLOVE(i).thumb(22:24,3)']]);
    avge_yaw(i,46:54) = cell2mat([GLOVE(i).hand(4:6,3)'
GLOVE(i).hand(13:15,3)' GLOVE(i).hand(22:24,3)']]);

    %AVGE FLATS
    avge_flats(i,1:18) = cell2mat([GLOVE(i).pinky(4:5,1)'
GLOVE(i).pinky(13:14,1)' GLOVE(i).pinky(22:23,1)' GLOVE(i).pinky(4:5,2)'
GLOVE(i).pinky(13:14,2)' GLOVE(i).pinky(22:23,2)' GLOVE(i).pinky(4:5,3)'
GLOVE(i).pinky(13:14,3)' GLOVE(i).pinky(22:23,3)']]);
    avge_flats(i,19:36) = cell2mat([GLOVE(i).ring(4:5,1)'
GLOVE(i).ring(13:14,1)' GLOVE(i).ring(22:23,1)' GLOVE(i).ring(4:5,2)'
GLOVE(i).ring(13:14,2)' GLOVE(i).ring(22:23,2)' GLOVE(i).ring(4:5,3)'
GLOVE(i).ring(13:14,3)' GLOVE(i).ring(22:23,3)']]);
    avge_flats(i,37:54) = cell2mat([GLOVE(i).middle(4:5,1)'
GLOVE(i).middle(13:14,1)' GLOVE(i).middle(22:23,1)' GLOVE(i).middle(4:5,2)'
GLOVE(i).middle(13:14,2)' GLOVE(i).middle(22:23,2)' GLOVE(i).middle(4:5,3)'
GLOVE(i).middle(13:14,3)' GLOVE(i).middle(22:23,3)']]);

```

```

    avge_flats(i,55:72) = cell2mat([GLOVE(i).index(4:5,1)'
GLOVE(i).index(13:14,1)' GLOVE(i).index(22:23,1)' GLOVE(i).index(4:5,2)'
GLOVE(i).index(13:14,2)' GLOVE(i).index(22:23,2)' GLOVE(i).index(4:5,3)'
GLOVE(i).index(13:14,3)' GLOVE(i).index(22:23,3)']]);
    avge_flats(i,73:90) = cell2mat([GLOVE(i).thumb(4:5,1)'
GLOVE(i).thumb(13:14,1)' GLOVE(i).thumb(22:23,1)' GLOVE(i).thumb(4:5,2)'
GLOVE(i).thumb(13:14,2)' GLOVE(i).thumb(22:23,2)' GLOVE(i).thumb(4:5,3)'
GLOVE(i).thumb(13:14,3)' GLOVE(i).thumb(22:23,3)']]);
    avge_flats(i,91:108) = cell2mat([GLOVE(i).hand(4:5,1)'
GLOVE(i).hand(13:14,1)' GLOVE(i).hand(22:23,1)' GLOVE(i).hand(4:5,2)'
GLOVE(i).hand(13:14,2)' GLOVE(i).hand(22:23,2)' GLOVE(i).hand(4:5,3)'
GLOVE(i).hand(13:14,3)' GLOVE(i).hand(22:23,3)']]);
    %AVGE TRANSIENT
    avge_transient(i,1:9) = avge_comp(i,3:3:27);
    avge_transient(i,10:18) = avge_comp(i,30:3:54);
    avge_transient(i,19:27) = avge_comp(i,57:3:81);
    avge_transient(i,28:36) = avge_comp(i,84:3:108);
    avge_transient(i,37:45) = avge_comp(i,111:3:135);
    avge_transient(i,46:54) = avge_comp(i,138:3:162);
    avge_q(i,1:27) = cell2mat([GLOVE(i).pinky(4:6,4)'
GLOVE(i).pinky(13:15,4)' GLOVE(i).pinky(22:24,4)' GLOVE(i).pinky(4:6,5)'
GLOVE(i).pinky(13:15,5)' GLOVE(i).pinky(22:24,5)' GLOVE(i).pinky(4:6,6)'
GLOVE(i).pinky(13:15,6)' GLOVE(i).pinky(22:24,6)']]);
    avge_q(i,28:54) = cell2mat([GLOVE(i).ring(4:6,4)' GLOVE(i).ring(13:15,4)'
GLOVE(i).ring(22:24,4)' GLOVE(i).ring(4:6,5)' GLOVE(i).ring(13:15,5)'
GLOVE(i).ring(22:24,5)' GLOVE(i).ring(4:6,6)' GLOVE(i).ring(13:15,6)'
GLOVE(i).ring(22:24,6)']]);
    avge_q(i,55:81) = cell2mat([GLOVE(i).middle(4:6,4)'
GLOVE(i).middle(13:15,4)' GLOVE(i).middle(22:24,4)' GLOVE(i).middle(4:6,5)'
GLOVE(i).middle(13:15,5)' GLOVE(i).middle(22:24,5)' GLOVE(i).middle(4:6,6)'
GLOVE(i).middle(13:15,6)' GLOVE(i).middle(22:24,6)']]);
    avge_q(i,82:108) = cell2mat([GLOVE(i).index(4:6,4)'
GLOVE(i).index(13:15,4)' GLOVE(i).index(22:24,4)' GLOVE(i).index(4:6,5)'
GLOVE(i).index(13:15,5)' GLOVE(i).index(22:24,5)' GLOVE(i).index(4:6,6)'
GLOVE(i).index(13:15,6)' GLOVE(i).index(22:24,6)']]);
    avge_q(i,109:135) = cell2mat([GLOVE(i).thumb(4:6,4)'
GLOVE(i).thumb(13:15,4)' GLOVE(i).thumb(22:24,4)' GLOVE(i).thumb(4:6,5)'
GLOVE(i).thumb(13:15,5)' GLOVE(i).thumb(22:24,5)' GLOVE(i).thumb(4:6,6)'
GLOVE(i).thumb(13:15,6)' GLOVE(i).thumb(22:24,6)']]);
    avge_q(i,136:162) = cell2mat([GLOVE(i).hand(4:6,4)'
GLOVE(i).hand(13:15,4)' GLOVE(i).hand(22:24,4)' GLOVE(i).hand(4:6,5)'
GLOVE(i).hand(13:15,5)' GLOVE(i).hand(22:24,5)' GLOVE(i).hand(4:6,6)'
GLOVE(i).hand(13:15,6)' GLOVE(i).hand(22:24,6)']]);
    avge_q1(i,1:27) = cell2mat([GLOVE(i).pinky(4:6,7)'
GLOVE(i).pinky(13:15,7)' GLOVE(i).pinky(22:24,7)' GLOVE(i).pinky(4:6,8)'
GLOVE(i).pinky(13:15,8)' GLOVE(i).pinky(22:24,8)' GLOVE(i).pinky(4:6,9)'
GLOVE(i).pinky(13:15,9)' GLOVE(i).pinky(22:24,9)']]);
    avge_q1(i,28:54) = cell2mat([GLOVE(i).ring(4:6,7)'
GLOVE(i).ring(13:15,7)' GLOVE(i).ring(22:24,7)' GLOVE(i).ring(4:6,8)'
GLOVE(i).ring(13:15,8)' GLOVE(i).ring(22:24,8)' GLOVE(i).ring(4:6,9)'
GLOVE(i).ring(13:15,9)' GLOVE(i).ring(22:24,9)']]);
    avge_q1(i,55:81) = cell2mat([GLOVE(i).middle(4:6,7)'
GLOVE(i).middle(13:15,7)' GLOVE(i).middle(22:24,7)' GLOVE(i).middle(4:6,8)'
GLOVE(i).middle(13:15,8)' GLOVE(i).middle(22:24,8)' GLOVE(i).middle(4:6,9)'
GLOVE(i).middle(13:15,9)' GLOVE(i).middle(22:24,9)']]);
    avge_q1(i,82:108) = cell2mat([GLOVE(i).index(4:6,7)'
GLOVE(i).index(13:15,7)' GLOVE(i).index(22:24,7)' GLOVE(i).index(4:6,8)'

```

```

GLOVE(i).index(13:15,8)' GLOVE(i).index(22:24,8)' GLOVE(i).index(4:6,9)'
GLOVE(i).index(13:15,9)' GLOVE(i).index(22:24,9)']];
    avge_q1(i,109:135) = cell2mat([GLOVE(i).thumb(4:6,7)'
GLOVE(i).thumb(13:15,7)' GLOVE(i).thumb(22:24,7)' GLOVE(i).thumb(4:6,8)'
GLOVE(i).thumb(13:15,8)' GLOVE(i).thumb(22:24,8)' GLOVE(i).thumb(4:6,9)'
GLOVE(i).thumb(13:15,9)' GLOVE(i).thumb(22:24,9)']]);
    avge_q1(i,136:162) = cell2mat([GLOVE(i).hand(4:6,7)'
GLOVE(i).hand(13:15,7)' GLOVE(i).hand(22:24,7)' GLOVE(i).hand(4:6,8)'
GLOVE(i).hand(13:15,8)' GLOVE(i).hand(22:24,8)' GLOVE(i).hand(4:6,9)'
GLOVE(i).hand(13:15,9)' GLOVE(i).hand(22:24,9)']]);

%%%%MOTION MONITOR ANGLES - EM data
    angles(i,1:27) = cell2mat([GLOVE(i).pinky(7:9,1)'
GLOVE(i).pinky(16:18,1)' GLOVE(i).pinky(25:27,1)' GLOVE(i).pinky(7:9,2)'
GLOVE(i).pinky(16:18,2)' GLOVE(i).pinky(25:27,2)' GLOVE(i).pinky(7:9,3)'
GLOVE(i).pinky(16:18,3)' GLOVE(i).pinky(25:27,3)']]);
    angles(i,28:54) = cell2mat([GLOVE(i).ring(7:9,1)' GLOVE(i).ring(16:18,1)'
GLOVE(i).ring(25:27,1)' GLOVE(i).ring(7:9,2)' GLOVE(i).ring(16:18,2)'
GLOVE(i).ring(25:27,2)' GLOVE(i).ring(7:9,3)' GLOVE(i).ring(16:18,3)'
GLOVE(i).ring(25:27,3)']]);
    angles(i,55:81) = cell2mat([GLOVE(i).middle(7:9,1)'
GLOVE(i).middle(16:18,1)' GLOVE(i).middle(25:27,1)' GLOVE(i).middle(7:9,2)'
GLOVE(i).middle(16:18,2)' GLOVE(i).middle(25:27,2)' GLOVE(i).middle(7:9,3)'
GLOVE(i).middle(16:18,3)' GLOVE(i).middle(25:27,3)']]);
    angles(i,82:108) = cell2mat([GLOVE(i).index(7:9,1)'
GLOVE(i).index(16:18,1)' GLOVE(i).index(25:27,1)' GLOVE(i).index(7:9,2)'
GLOVE(i).index(16:18,2)' GLOVE(i).index(25:27,2)' GLOVE(i).index(7:9,3)'
GLOVE(i).index(16:18,3)' GLOVE(i).index(25:27,3)']]);
    angles(i,109:135) = cell2mat([GLOVE(i).thumb(7:9,1)'
GLOVE(i).thumb(16:18,1)' GLOVE(i).thumb(25:27,1)' GLOVE(i).thumb(7:9,2)'
GLOVE(i).thumb(16:18,2)' GLOVE(i).thumb(25:27,2)' GLOVE(i).thumb(7:9,3)'
GLOVE(i).thumb(16:18,3)' GLOVE(i).thumb(25:27,3)']]);
    angles(i,136:162) = cell2mat([GLOVE(i).hand(7:9,1)'
GLOVE(i).hand(16:18,1)' GLOVE(i).hand(25:27,1)' GLOVE(i).hand(7:9,2)'
GLOVE(i).hand(16:18,2)' GLOVE(i).hand(25:27,2)' GLOVE(i).hand(7:9,3)'
GLOVE(i).hand(16:18,3)' GLOVE(i).hand(25:27,3)']]);

%ANGLES ROLL
    angles_roll(i,1:9) = angles(i,1:9);
    angles_roll(i,10:18) = angles(i,28:36);
    angles_roll(i,19:27) = angles(i,55:63);
    angles_roll(i,28:36) = angles(i,82:90);
    angles_roll(i,37:45) = angles(i,109:117);
    angles_roll(i,46:54) = angles(i,136:144);
%ANGLES pitch
    angles_pitch(i,1:9) = angles(i,10:18);
    angles_pitch(i,10:18) = angles(i,37:45);
    angles_pitch(i,19:27) = angles(i,64:72);
    angles_pitch(i,28:36) = angles(i,91:99);
    angles_pitch(i,37:45) = angles(i,118:126);
    angles_pitch(i,46:54) = angles(i,145:153);
%ANGLES yaw
    angles_yaw(i,1:9) = angles(i,19:27);
    angles_yaw(i,10:18) = angles(i,46:54);
    angles_yaw(i,19:27) = angles(i,73:81);
    angles_yaw(i,28:36) = angles(i,100:108);

```

```

angles_yaw(i,37:45) = angles(i,127:135);
angles_yaw(i,46:54) = angles(i,154:162);

%ANGLES FLATS
angles_flats(i,1:18) = cell2mat([GLOVE(i).pinky(7:8,1) '
GLOVE(i).pinky(16:17,1) ' GLOVE(i).pinky(25:26,1) ' GLOVE(i).pinky(7:8,2) '
GLOVE(i).pinky(16:17,2) ' GLOVE(i).pinky(25:26,2) ' GLOVE(i).pinky(7:8,3) '
GLOVE(i).pinky(16:17,3) ' GLOVE(i).pinky(25:26,3) ']);
angles_flats(i,19:36) = cell2mat([GLOVE(i).ring(7:8,1) '
GLOVE(i).ring(16:17,1) ' GLOVE(i).ring(25:26,1) ' GLOVE(i).ring(7:8,2) '
GLOVE(i).ring(16:17,2) ' GLOVE(i).ring(25:26,2) ' GLOVE(i).ring(7:8,3) '
GLOVE(i).ring(16:17,3) ' GLOVE(i).ring(25:26,3) ']);
angles_flats(i,37:54) = cell2mat([GLOVE(i).middle(7:8,1) '
GLOVE(i).middle(16:17,1) ' GLOVE(i).middle(25:26,1) ' GLOVE(i).middle(7:8,2) '
GLOVE(i).middle(16:17,2) ' GLOVE(i).middle(25:26,2) ' GLOVE(i).middle(7:8,3) '
GLOVE(i).middle(16:17,3) ' GLOVE(i).middle(25:26,3) ']);
angles_flats(i,55:72) = cell2mat([GLOVE(i).index(7:8,1) '
GLOVE(i).index(16:17,1) ' GLOVE(i).index(25:26,1) ' GLOVE(i).index(7:8,2) '
GLOVE(i).index(16:17,2) ' GLOVE(i).index(25:26,2) ' GLOVE(i).index(7:8,3) '
GLOVE(i).index(16:17,3) ' GLOVE(i).index(25:26,3) ']);
angles_flats(i,73:90) = cell2mat([GLOVE(i).thumb(7:8,1) '
GLOVE(i).thumb(16:17,1) ' GLOVE(i).thumb(25:26,1) ' GLOVE(i).thumb(7:8,2) '
GLOVE(i).thumb(16:17,2) ' GLOVE(i).thumb(25:26,2) ' GLOVE(i).thumb(7:8,3) '
GLOVE(i).thumb(16:17,3) ' GLOVE(i).thumb(25:26,3) ']);
angles_flats(i,91:108) = cell2mat([GLOVE(i).hand(7:8,1) '
GLOVE(i).hand(16:17,1) ' GLOVE(i).hand(25:26,1) ' GLOVE(i).hand(7:8,2) '
GLOVE(i).hand(16:17,2) ' GLOVE(i).hand(25:26,2) ' GLOVE(i).hand(7:8,3) '
GLOVE(i).hand(16:17,3) ' GLOVE(i).hand(25:26,3) ']);
%ANGLES TRANSIENT
angles_transient(i,1:9) = angles(i,3:3:27);
angles_transient(i,10:18) = angles(i,30:3:54);
angles_transient(i,19:27) = angles(i,57:3:81);
angles_transient(i,28:36) = angles(i,84:3:108);
angles_transient(i,37:45) = angles(i,111:3:135);
angles_transient(i,46:54) = angles(i,138:3:162);
end

[w l] = size(rms_roll);
len = w*1;
rms_roll_line = reshape(rms_roll, 1, len); %Reshaping all vars into arrays
rms_pitch_line = reshape(rms_pitch, 1, len);
rms_yaw_line = reshape(rms_yaw, 1, len);

avge_roll_line = reshape(avge_roll, 1, len);
avge_pitch_line = reshape(avge_pitch, 1, len);
avge_yaw_line = reshape(avge_yaw, 1, len);

angles_roll_line = reshape(angles_roll, 1, len);
angles_pitch_line = reshape(angles_pitch, 1, len);
angles_yaw_line = reshape(angles_yaw, 1, len);

[w l] = size(rms_flats);
len = w*1;
rms_flats_line = reshape(rms_flats, 1, len);
avge_flats_line = reshape(avge_flats, 1, len);
angles_flats_line = reshape(angles_flats, 1, len);

```

```

[w 1] = size(rms_transient);
len = w*1;
rms_transient_line = reshape(rms_transient, 1, len);
avge_transient_line = reshape(avge_transient, 1, len);
angles_transient_line = reshape(angles_transient, 1, len);

```

g) ForceTesting.m

Every glove variable is named for each orientation during experiments, this script puts them together and orders them.

```

%Force testing
mmf = dlmread('upward_test5.txt'); %Reading off force plate
fid = fopen('Upward_test5'); %Reading off glove
allsensors;
Ftest = 5;%Test #

mmf(:,2:3) = mmf(:,2:3)/50; %Dividing by gain for force plate
(Fx=50,Fy=50,Fz=20,Mx=100,My=50,Mz=50)
mmf(:,4) = mmf(:,4)/20;
mmf(:,5) = mmf(:,5)/100;
mmf(:,6:7) = mmf(:,6:7)/50;
mmf1 = mmf(:,2:7);
Cmatrix = [646.4 -13.4 .4 -3.9 7.9 -2.7; 4.2 649.6 .2 -8.9 -1.8 .8;
4.9 -7.8 973.1 8.5 .5 -1.8; .3 -37.5 -.8 367.5 .1 .1; 32 -1 1 -
.8 257.5 .8; -.6 .6 -1.5 -1.5 -.2 159.9]; %Cmatrix used to convert
readings into force in N
for i = 1:length(mmf1(:, :))
    forces(i,1:6) = mmf1(i,1:6)*Cmatrix;
    sumforce(i,1) = sum(abs(forces(i,1:6))); %Finding sum of all forces @
each pt
end

%Thumb %Order of each sensor
%Index
%Middle
%Ring
%Pinky
%SCALING PURPOSES
Fx_total = Fx_total*1.877;
Fy_total = Fy_total*2.230;
Fz_total = Fz_total*1.892;
My_total = My_total*1.892; %Scale factor from Fz used
Mz_total = Mz_total*2.23; %Scale factor from Fy used

if Ftest == 1
    j = 1;
    sumforce = zeros(length(mmf),1);
    FORCE = zeros(11,5);
    RMSE = zeros(5,5);
    AVGE = zeros(5,5);
    RMSE(1,j) = sqrt(sum((forces(750:5:1250,1)- (Fy_total(150:250)'))).^2)/61);
%Force plate data has 5 times as many data pts, RMSE value and AVGE value for
each component x of force plate corresponds to y of glove
    AVGE(1,j) = sum((forces(750:5:1250,1)- (Fy_total(150:250)')))/61;
    RMSE(2,j) = sqrt(sum((forces(750:5:1250,2)- (Fx_total(150:250)'))).^2)/61);

```

```

AVGE(2,j) = sum((forces(750:5:1250,2)- (Fx_total(150:250)')))/61;
RMSE(3,j) = sqrt(sum((forces(750:5:1250,3)- (Fz_total(150:250)')).^2)/61);
AVGE(3,j) = sum((forces(750:5:1250,3)- (Fz_total(150:250)')))/61;
RMSE(4,j) = sqrt(sum((forces(750:5:1250,4)- (-My_total(150:250)')).^2)/61);
AVGE(4,j) = sum((forces(750:5:1250,4)- (-My_total(150:250)')))/61;
RMSE(5,j) = sqrt(sum((forces(750:5:1250,6)- (Mz_total(150:250)')).^2)/61);
AVGE(5,j) = sum((forces(750:5:1250,6)- (Mz_total(150:250)')))/61;
FORCE(1,j) = mean(forces(750:5:1250,1)); %Average force data for each
component
FORCE(2,j) = mean(forces(750:5:1250,2));
FORCE(3,j) = mean(forces(750:5:1250,3));
FORCE(4,j) = mean(forces(750:5:1250,4));
FORCE(5,j) = mean(forces(750:5:1250,5));
FORCE(6,j) = mean(sumforce(750:5:1250,1));
FORCE(7,j) = mean(F_total(150:250)); %Total of all forces
FORCE(8,j) = mean(sumforce(750:5:1250,1))/mean(F_total(150:250)');%Ratio of
total forces for force plate data and glove data
FORCE(9,j) = mean(Fx_total(150:250)')/FORCE(7,j); %Ratio of component to
total
FORCE(10,j) = mean(Fy_total(150:250)')/FORCE(7,j);
FORCE(11,j) = mean(Fz_total(150:250)')/FORCE(7,j);
elseif Ftest == 2 %Same thing repeated for each test
    j = 2;
    RMSE(1,j) = sqrt(sum((forces(750:5:1250,1)- (Fy_total(150:250)')).^2)/61);
    AVGE(1,j) = sum((forces(750:5:1250,1)- (Fy_total(150:250)')))/61;
    RMSE(2,j) = sqrt(sum((forces(750:5:1250,2)- (Fx_total(150:250)')).^2)/61);
    AVGE(2,j) = sum((forces(750:5:1250,2)- (Fx_total(150:250)')))/61;
    RMSE(3,j) = sqrt(sum((forces(750:5:1250,3)- (Fz_total(150:250)')).^2)/61);
    AVGE(3,j) = sum((forces(750:5:1250,3)- (Fz_total(150:250)')))/61;
    RMSE(4,j) = sqrt(sum((forces(750:5:1250,4)- (-My_total(150:250)')).^2)/61);
    AVGE(4,j) = sum((forces(750:5:1250,4)- (-My_total(150:250)')))/61;
    RMSE(5,j) = sqrt(sum((forces(750:5:1250,6)- (Mz_total(150:250)')).^2)/61);
    AVGE(5,j) = sum((forces(750:5:1250,6)- (Mz_total(150:250)')))/61;
    FORCE(1,j) = mean(forces(750:5:1250,1));
    FORCE(2,j) = mean(forces(750:5:1250,2));
    FORCE(3,j) = mean(forces(750:5:1250,3));
    FORCE(4,j) = mean(forces(750:5:1250,4));
    FORCE(5,j) = mean(forces(750:5:1250,5));
    FORCE(6,j) = mean(sumforce(750:5:1250,1));
    FORCE(7,j) = mean(F_total(150:250));
    FORCE(8,j) = mean(sumforce(750:5:1250,1))/mean(F_total(150:250)');
    FORCE(9,j) = mean(Fx_total(150:250)')/FORCE(7,j);
    FORCE(10,j) = mean(Fy_total(150:250)')/FORCE(7,j);
    FORCE(11,j) = mean(Fz_total(150:250)')/FORCE(7,j);

elseif Ftest == 3
    j = 3;
    RMSE(1,j) = sqrt(sum((forces(750:5:1250,1)-
(Fy_total(150:250)')).^2)/61);
    AVGE(1,j) = sum((forces(750:5:1250,1)- (Fy_total(150:250)')))/61;
    RMSE(2,j) = sqrt(sum((forces(750:5:1250,2)- (Fx_total(150:250)')).^2)/61);
    AVGE(2,j) = sum((forces(750:5:1250,2)- (Fx_total(150:250)')))/61;
    RMSE(3,j) = sqrt(sum((forces(750:5:1250,3)- (Fz_total(150:250)')).^2)/61);
    AVGE(3,j) = sum((forces(750:5:1250,3)- (Fz_total(150:250)')))/61;
    RMSE(4,j) = sqrt(sum((forces(750:5:1250,4)- (-My_total(150:250)')).^2)/61);
    AVGE(4,j) = sum((forces(750:5:1250,4)- (-My_total(150:250)')))/61;
    RMSE(5,j) = sqrt(sum((forces(750:5:1250,6)- (Mz_total(150:250)')).^2)/61);

```

```

AVGE(5,j) = sum((forces(750:5:1250,6) - (Mz_total(150:250)')))/61;
FORCE(1,j) = mean(forces(750:5:1250,1));
FORCE(2,j) = mean(forces(750:5:1250,2));
FORCE(3,j) = mean(forces(750:5:1250,3));
FORCE(4,j) = mean(forces(750:5:1250,4));
FORCE(5,j) = mean(forces(750:5:1250,5));
FORCE(6,j) = mean(sumforce(750:5:1250,1));
FORCE(7,j) = mean(F_total(150:250));
FORCE(8,j) = mean(sumforce(750:5:1250,1))/mean(F_total(150:250)');
FORCE(9,j) = mean(Fx_total(150:250)')/FORCE(7,j);
FORCE(10,j) = mean(Fy_total(150:250)')/FORCE(7,j);
FORCE(11,j) = mean(Fz_total(150:250)')/FORCE(7,j);

elseif Ftest == 4
    j = 4;
    RMSE(1,j) = sqrt(sum((forces(750:5:1250,1) -
(Fy_total(150:250)'))).^2)/61);
AVGE(1,j) = sum((forces(750:5:1250,1) - (Fy_total(150:250)')))/61;
RMSE(2,j) = sqrt(sum((forces(750:5:1250,2) - (Fx_total(150:250)'))).^2)/61);
AVGE(2,j) = sum((forces(750:5:1250,2) - (Fx_total(150:250)')))/61;
RMSE(3,j) = sqrt(sum((forces(750:5:1250,3) - (Fz_total(150:250)'))).^2)/61);
AVGE(3,j) = sum((forces(750:5:1250,3) - (Fz_total(150:250)')))/61;
RMSE(4,j) = sqrt(sum((forces(750:5:1250,4) - (-My_total(150:250)'))).^2)/61);
AVGE(4,j) = sum((forces(750:5:1250,4) - (-My_total(150:250)')))/61;
RMSE(5,j) = sqrt(sum((forces(750:5:1250,6) - (Mz_total(150:250)'))).^2)/61);
AVGE(5,j) = sum((forces(750:5:1250,6) - (Mz_total(150:250)')))/61;
FORCE(1,j) = mean(forces(750:5:1250,1));
FORCE(2,j) = mean(forces(750:5:1250,2));
FORCE(3,j) = mean(forces(750:5:1250,3));
FORCE(4,j) = mean(forces(750:5:1250,4));
FORCE(5,j) = mean(forces(750:5:1250,5));
FORCE(6,j) = mean(sumforce(750:5:1250,1));
FORCE(7,j) = mean(F_total(150:250));
FORCE(8,j) = mean(sumforce(750:5:1250,1))/mean(F_total(150:250)');
FORCE(9,j) = mean(Fx_total(150:250)')/FORCE(7,j);
FORCE(10,j) = mean(Fy_total(150:250)')/FORCE(7,j);
FORCE(11,j) = mean(Fz_total(150:250)')/FORCE(7,j);
elseif Ftest == 5
    j = 5;
    RMSE(1,j) = sqrt(sum((forces(750:5:1250,1) - (Fy_total(150:250)'))).^2)/61);
AVGE(1,j) = sum((forces(750:5:1250,1) - (Fy_total(150:250)')))/61;
RMSE(2,j) = sqrt(sum((forces(750:5:1250,2) - (Fx_total(150:250)'))).^2)/61);
AVGE(2,j) = sum((forces(750:5:1250,2) - (Fx_total(150:250)')))/61;
RMSE(3,j) = sqrt(sum((forces(750:5:1250,3) - (Fz_total(150:250)'))).^2)/61);
AVGE(3,j) = sum((forces(750:5:1250,3) - (Fz_total(150:250)')))/61;
RMSE(4,j) = sqrt(sum((forces(750:5:1250,4) - (-My_total(150:250)'))).^2)/61);
AVGE(4,j) = sum((forces(750:5:1250,4) - (-My_total(150:250)')))/61;
RMSE(5,j) = sqrt(sum((forces(750:5:1250,6) - (Mz_total(150:250)'))).^2)/61);
AVGE(5,j) = sum((forces(750:5:1250,6) - (Mz_total(150:250)')))/61;
FORCE(1,j) = mean(forces(750:5:1250,1));
FORCE(2,j) = mean(forces(750:5:1250,2));
FORCE(3,j) = mean(forces(750:5:1250,3));
FORCE(4,j) = mean(forces(750:5:1250,4));
FORCE(5,j) = mean(forces(750:5:1250,5));
FORCE(6,j) = mean(sumforce(750:5:1250,1));
FORCE(7,j) = mean(F_total(150:250));
FORCE(8,j) = mean(sumforce(750:5:1250,1))/mean(F_total(150:250)');

```

```

FORCE(9,j) = mean(Fx_total(150:250)')/FORCE(7,j);
FORCE(10,j) = mean(Fy_total(150:250)')/FORCE(7,j);
FORCE(11,j) = mean(Fz_total(150:250)')/FORCE(7,j);
end
if j == 5
Forcehand.RMSE = RMSE(1:5,1:5); %All RMSE values
Forcehand.AVGE = AVGE(1:5,1:5); %All AVGE values
Forcehand.FORCE = FORCE(1:11,1:5); %Force values and ratios
End

```

h) FORCEHANDtesting.m

This script obtains data from all force tests and develops matrices to be analyzed further in statistical tests.

```

FORCEHAND = [Forcehand1 Forcehand2 Forcehand3 Forcehand4 Forcehand5
Forcehand6 Forcehand7 Forcehand8 Forcehand9 ];
for i = 1:length(FORCEHAND)
    if i == 1
        k = 1;
    end
    for j = 1:5
        RMSE_TOTAL(j,k:k+4) = FORCEHAND(i).RMSE(j,1:5);
        AVGE_TOTAL(j,k:k+4) = FORCEHAND(i).AVGE(j,1:5);
    end
    for j = 1:12
        FORCE_TOTAL(j,k:k+4) = FORCEHAND(i).FORCE(j,1:5);
    end
    for j = 1:3
        SF_TOTAL(j,k:k+4) = FORCEHAND(i).SF(j,1:5);
    end
    k = k+5;
end
%Using the dimension tested
RMSE_1 = zeros(9,5);
AVGE_1 = zeros(9,5);
RMSE_2 = zeros(9,5);
AVGE_2 = zeros(9,5);
RMSEM = zeros(9,5);
AVGEM = zeros(9,5);

RMSE_1(1:9,1:5) = [RMSE_TOTAL(2,1:4) 0; RMSE_TOTAL(3,8:10) 0 0;
RMSE_TOTAL(3,11:15); RMSE_TOTAL(2,16:20); RMSE_TOTAL(1,21:25);
RMSE_TOTAL(2,26:30); RMSE_TOTAL(1,31) RMSE_TOTAL(1,33:34) 0 0;
RMSE_TOTAL(3,42) RMSE_TOTAL(3,44:45) 0 0; RMSE_TOTAL(3,46:50)];
AVGE_1(1:9,1:5) = [AVGE_TOTAL(2,1:4) 0; AVGE_TOTAL(3,8:10) 0 0;
AVGE_TOTAL(3,11:15); AVGE_TOTAL(2,16:20); AVGE_TOTAL(1,21:25);
AVGE_TOTAL(2,26:30); AVGE_TOTAL(1,31) AVGE_TOTAL(1,33:34) 0 0;
AVGE_TOTAL(3,42) AVGE_TOTAL(3,44:45) 0 0; AVGE_TOTAL(3,46:50)];
FORCE_1(1:9,1:5) = [FORCE_TOTAL(2,1:4) 0; FORCE_TOTAL(3,8:10) 0 0;
FORCE_TOTAL(3,11:15); FORCE_TOTAL(2,16:20); FORCE_TOTAL(1,21:25);
FORCE_TOTAL(2,26:30); FORCE_TOTAL(1,31) FORCE_TOTAL(1,33:34) 0 0;
FORCE_TOTAL(3,42) FORCE_TOTAL(3,44:45) 0 0; FORCE_TOTAL(3,46:50)];

```

```

RMSE_2(1:9,1:5) = [RMSE_TOTAL(3,1:4) 0; RMSE_TOTAL(2,8:10) 0 0;
RMSE_TOTAL(2,11:15); RMSE_TOTAL(3,16:20); RMSE_TOTAL(2,21:25);
RMSE_TOTAL(3,26:30); RMSE_TOTAL(2,31) RMSE_TOTAL(2,33:34) 0 0;
RMSE_TOTAL(2,42) RMSE_TOTAL(2,44:45) 0 0; RMSE_TOTAL(2,46:50)];
AVGE_2(1:9,1:5) = [AVGE_TOTAL(3,1:4) 0; AVGE_TOTAL(2,8:10) 0 0;
AVGE_TOTAL(2,11:15); AVGE_TOTAL(3,16:20); AVGE_TOTAL(2,21:25);
AVGE_TOTAL(3,26:30); AVGE_TOTAL(2,31) AVGE_TOTAL(2,33:34) 0 0;
AVGE_TOTAL(2,42) AVGE_TOTAL(2,44:45) 0 0; AVGE_TOTAL(2,46:50)];
FORCE_2(1:9,1:5) = [FORCE_TOTAL(3,1:4) 0; FORCE_TOTAL(2,8:10) 0 0;
FORCE_TOTAL(2,11:15); FORCE_TOTAL(3,16:20); FORCE_TOTAL(2,21:25);
FORCE_TOTAL(3,26:30); FORCE_TOTAL(2,31) FORCE_TOTAL(2,33:34) 0 0;
FORCE_TOTAL(2,42) FORCE_TOTAL(2,44:45) 0 0; FORCE_TOTAL(2,46:50)];

```

```

%
RMSEM(1:9,1:5) = [RMSE_TOTAL(4,1:4) 0; RMSE_TOTAL(4,8:10) 0 0;
RMSE_TOTAL(4,11:15); RMSE_TOTAL(4,16:20); RMSE_TOTAL(5,16:20);
RMSE_TOTAL(4,26:30); RMSE_TOTAL(5,31) RMSE_TOTAL(5,33:34) 0 0;
RMSE_TOTAL(4,42) RMSE_TOTAL(4,44:45) 0 0; RMSE_TOTAL(4,46:50)];
AVGEM(1:9,1:5) = [AVGE_TOTAL(4,1:4) 0; AVGE_TOTAL(4,8:10) 0 0;
AVGE_TOTAL(4,11:15); AVGE_TOTAL(4,16:20); AVGE_TOTAL(5,16:20);
AVGE_TOTAL(4,26:30); AVGE_TOTAL(5,31) AVGE_TOTAL(5,33:34) 0 0;
AVGE_TOTAL(4,42) AVGE_TOTAL(4,44:45) 0 0; AVGE_TOTAL(4,46:50)];
FORCEM(1:9,1:5) = [FORCE_TOTAL(4,1:4) 0; FORCE_TOTAL(4,8:10) 0 0;
FORCE_TOTAL(4,11:15); FORCE_TOTAL(4,16:20); FORCE_TOTAL(5,16:20);
FORCE_TOTAL(4,26:30); FORCE_TOTAL(5,31) FORCE_TOTAL(5,33:34) 0 0;
FORCE_TOTAL(4,42) FORCE_TOTAL(4,44:45) 0 0; FORCE_TOTAL(4,46:50)];

```

```

RMSE_1line(1,1:45) = reshape(RMSE_1,1,45);
AVGE_1line(1,1:45) = reshape(AVGE_1,1,45);
FORCE_1line(1,1:45) = reshape(FORCE_1,1,45);
RMSE_2line(1,1:45) = reshape(RMSE_2,1,45);
AVGE_2line(1,1:45) = reshape(AVGE_2,1,45);
FORCE_2line(1,1:45) = reshape(FORCE_2,1,45);
RMSEM_line(1,1:45) = reshape(RMSEM,1,45);
AVGEM_line(1,1:45) = reshape(AVGEM,1,45);
FORCEM_line(1,1:45) = reshape(FORCEM,1,45);

```

A.3 Potential Improvements to Code

a) Roll improvement

```

%This could be used when roll values are larger than 70 degrees to check the
Euler angle calculation of roll - calculations sometimes jump near 90deg
if acos(filt_acc(i,3)) >= 1.9199 && abs(roll(i,1)) >= 1.22 %1.3090 1.5708
    yes = 1;
    if roll(i,1) >= 0
        si = 1;
    elseif roll(i,1) < 0
        si = -1;
    end
end
if yes == 1;
    roll(i:end,1) = si*acos(filt_acc(i:end,3));
end
end
end
end

```

b) kalmanfilterquaternion.m

This is an extended kalman filter (EKF) that could be used to fuse data from all of the sensors. There were problems when implementing this, specifically the covariances named sigmak, sigmaa, etc. If this was implemented correctly the Euler angle calculation may improve significantly. The first step of converting all sensors into usable data is not shown in the code below but follows the same procedure as earlier, however the data is not passed through a low pass filter before going through the EKF as this will change the 'sensor model' used in the EKF.

```
%Reference: Sabatini (27)
% %Constants and initial conditions
x0 = ones(10,1);
xtrue = ones(10,length(gyro)+1); %Generating true system state vector
wtrue = zeros(10,length(gyro)+1);
batrue = zeros(3,length(gyro)+1);
bmtrue = zeros(3,length(gyro)+1);
batrue(:,1) = [.0069, -.0052, -.1082]'; %Bias vector accel
bmtrue(:,1) = [.2052, -.2031, 0.7172]'; %Bias vector magn
q1true = zeros(4,length(gyro)+1);
%Changing these parameters did not work, if improved the EKF may work.
Suggested vals are in Sabatini's paper but did not work for our sensor data
sigmag = .04;%mrad/s,
sigmaa = .09;%m/s^2
sigmam = .03; %mGauss
asigmaw = .005;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
msigmaw = .05;%%%%%%%%%
asigmak = tsa*asigmaw^2 * eye(3,3);
msigmak = tsm*msigmaw^2 * eye(3,3);
w = zeros(3,length(gyro)+1);
g = [0 0 1]';%Gravity vector
h = filt_mag(100, :)';
ba = zeros(3,length(gyro)+1);
bm = zeros(3,length(gyro)+1);
ba(:,1) = [0, 0, 1]' - mean(accel(1:50, :))'; %Initial bias vectors
bm(:,1) = mag(1, :) - mean(mag(1:50, :));
q1 = zeros(4,length(gyro)+1);
q1(:,1) = [.5, .5, .5, .5]'; %Initial quaternion
q1(:,1) = q1(:,1)/norm(q1(:,1)); %Quaternion must be normalized
ek = zeros(3,length(gyro)+1);
ek(:,1) = q1(1:3,1);
z = zeros(6,length(gyro)+1);
%Initial state conditions
xhm = ones(10,length(gyro)+1)*0;
xhp = ones(10,length(gyro)+1)*0;
xhp(1:4,1) = q1(:,1); %Initial state vector (q1,ba,bm)
xhp(5:7,1) = ba(:,1); %Part of state vector
xhp(8:10,1) = bm(:,1); %Part of state vector
qhm = ones(4,length(gyro)+1);
Pp = eye(10)* 10; %Initial covar matrix
xhrec = zeros(10,length(gyro)+1);
pitchtrue = zeros(1,length(gyro)+1);
rolltrue = zeros(1,length(gyro)+1);
yawtrue = zeros(1,length(gyro)+1);
%
% %White noise variables
vg = sigmag*randn(3,1); %Random Gaussian white noise vars
```

```

va = sigmaa*randn(3,1);
vm = sigmam*randn(3,10);
wka = sqrt(tsa)*asigmaw*randn(3,1);
wkm = sqrt(tsm)*msigmaw*randn(3,1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Using measured data
z(1:3,1:length(accel)) = accel'; %z is measurement var and is accel and mag
data
z(4:6,1:length(accel)) = mag';
accel = accel'; %Must transpose to fit with functions
mag = mag';
for i = 2:1:length(gyro)-1
    ek(:,i) = q1(1:3,i-1);
    w(:,i) = gyro(i,:)'; %w variable is gyroscope data
    p = w(1,i); q = w(2,i); r = w(3,i);
    wcross = X(w(:,i)); %Vector cross product of w
    omega = 1/2*[wcross w(:,i); -w(:,i)' 0]; %Omega is used in discrete-
time model for quaternion determination
    Cbn = 1/norm(q1(1:4,i-1)).* [(q1(1,i-1)^2 - q1(2,i-1)^2 - q1(3,i-1)^2 +
q1(4,i-1)^2), 2*((q1(1,i-1)*q1(2,i-1))+(q1(3,i-1)*q1(4,i-1))), 2*((q1(1,i-
1)*q1(3,i-1))- ((q1(2,i-1)*q1(4,i-1))), 2*((q1(1,i-1)*q1(2,i-1))-(q1(3,i-
1)*q1(4,i-1))), -q1(1,i-1)^2 + q1(2,i-1)^2 - q1(3,i-1)^2 + q1(4,i-1)^2,
2*((q1(2,i-1)*q1(3,i-1))+(q1(4,i-1)*q1(1,i-1))), 2*((q1(1,i-1)*q1(3,i-
1))+(q1(2,i-1)*q1(4,i-1))), 2*((q1(2,i-1)*q1(3,i-1))-(q1(4,i-1)*q1(1,i-1))),
-q1(1,i-1)^2 - q1(2,i-1)^2 + q1(3,i-1)^2 + q1(4,i-1)^2]; %This is the
direction cosine matrix for a quaternion

%Measurement white noise - noise generated from measurement model
ekcross = X(ek(:,i));
wkq = (-ts/2)*[(ekcross)+q1(4,i-1)*eye(3,3); -ek(:,i)']*vg(:,1);
xi = [(ekcross)+q1(4,i-1)*eye(3,3); -ek(:,i)'];

%Process Noise Covariance matrix (10X10)
Q = zeros(10,10);
Q(1:4,1:4) = ((ts/2)^2)*xi*sigmag*eye(3,3)*xi';
Q(5:7,5:7) = asigmak;
Q(8:10,8:10) = msigmak;

%Observation variance weighting of measurement model
if norm(accel(:,i+1)) - norm(accel(:,i)) <= .2/9.8; %Calculating
acceleration deviations from lg, if transient accelerations exist then the R
variance is increased as it is not as reliable
Rsigmaa(i) = sigmaa;
else
    Rsigmaa(i) = 100000;
end

if abs(norm(mag(:,i)) - norm(mag(:,10))) <= .1%Gauss; %Calculating external
magnetic field deviations. If deviations are found then variance is increased
significantly because mag data is not as reliable.
    Rsigmam(i) = sigmam;
else
    Rsigmam(i) = 100000;
end
R = [(Rsigmaa(i)^2)*eye(3,3), zeros(3,3); zeros(3,3),
(Rsigmam(i)^2)*eye(3,3)]; %Covariance matrix of measurement model

```

```

%Models completed
phi = zeros(10,10);
phi(1:4,1:4) = expm(omega.*ts); %Used to simulate state transition vector -
how state vector progresses in time
phi(5:7,5:7) = eye(3,3); %Bias vectors assumed constant
phi(8:10,8:10) = eye(3,3);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%KALMAN FILTER IMPLEMENTATION
xhm(1:4,i) = expm(omega*ts)*q1(:,i-1);%A priori state estimate
xhm(5:7,i) = ba(:,i-1);
xhm(8:10,i) = bm(:,i-1);

Pm = phi*Pp*phi'+Q; %A priori covar matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
qhm(:,i) = xhm(1:4,i)/norm(xhm(1:4,i)); %Quaternion must be normalized each
time pt
bahm(:,i) = xhm(5:7,i); %Temporary bias vector
bmhm(:,i) = xhm(8:10,i);
ekhm(:,i) = qhm(1:3,i);
Cbn = 1/norm(qhm(1:4,i)).* [(qhm(1,i)^2 - qhm(2,i)^2 - qhm(3,i)^2 +
qhm(4,i)^2), 2*((qhm(1,i)*qhm(2,i))+(qhm(3,i)*qhm(4,i))),
2*((qhm(1,i)*qhm(3,i))-((qhm(2,i)*qhm(4,i))))); 2*((qhm(1,i)*qhm(2,i))-
(qhm(3,i)*qhm(4,i))), -qhm(1,i)^2 + qhm(2,i)^2 - qhm(3,i)^2 + qhm(4,i)^2,
2*((qhm(2,i)*qhm(3,i))+(qhm(4,i)*qhm(1,i)));
2*((qhm(1,i)*qhm(3,i))+(qhm(2,i)*qhm(4,i))), 2*((qhm(2,i)*qhm(3,i))-
(qhm(4,i)*qhm(1,i))), -qhm(1,i)^2 - qhm(2,i)^2 + qhm(3,i)^2 + qhm(4,i)^2];
%Use a priori quaternion to update DCM
qone = qhm(1,i); q2 = qhm(2,i); q3 = qhm(3,i); q4 = qhm(4,i);
gx = g(1,1); gy = g(2,1); gz = g(3,1);%Gravity vector
hx = h(1,1); hy = h(2,1); hz = h(3,1); %Magnetic north vector

F = [2*qone*gx+2*q2*gy+2*q3*gz , -2*q2*gx+2*qone*gy-2*q4*gz , -
2*q3*gx+2*q4*gy+2*qone*gz , 2*q4*gx+2*q3*gy-2*q2*gz , 1, 0, 0, 0, 0, 0; ...
2*q2*gx-2*qone*gy+2*q4*gz , 2*qone*gx+2*q2*gy+2*q3*gz , -2*q4*gx-
2*q3*gy+2*q2*gz , -2*q3*gx+2*q4*gy+2*qone*gz , 0, 1, 0, 0, 0, 0; ...
2*q3*gx-2*q4*gy-2*qone*gz , 2*q4*gx+2*q3*gy-2*q2*gz ,
2*qone*gx+2*q2*gy+2*q3*gz , 2*q2*gx-2*qone*gy+2*q4*gz , 0, 0, 1, 0, 0, 0; ...
2*qone*hx+2*q2*hy+2*q3*hz , -2*q2*hx+2*qone*hy-2*q4*hz , -
2*q3*hx+2*q4*hy+2*qone*hz , 2*q4*hx+2*q3*hy-2*q2*hz , 0, 0, 0, 1, 0, 0; ...
2*q2*hx-2*qone*hy+2*q4*hz , 2*qone*hx+2*q2*hy+2*q3*hz , -2*q4*hx-
2*q3*hy+2*q2*hz , -2*q3*hx+2*q4*hy+2*qone*hz , 0, 0, 0, 0, 1, 0; ...
2*q3*hx-2*q4*hy-2*qone*hz , 2*q4*hx+2*q3*hy-2*q2*hz ,
2*qone*hx+2*q2*hy+2*q3*hz , 2*q2*hx-2*qone*hy+2*q4*hz , 0, 0, 0, 0, 0, 1]; F
matrix is calculated from measurement model linearization first -order
Taylor-Mac Laurin expansion
K = Pm*F' * (inv(F*Pm*F' + R)); %Compute Kalman gain
xhp(:,i) = xhm(:,i) + K*(z(:,i) - [Cbn*g+bahm(:,i); Cbn*h(:,:)+bmhm(:,i)]); %A
posteriori state estimate

Pp = Pm - K*F*Pm; %A posteriori covar matrix
q1(:,i) = xhp(1:4,i)/norm(xhp(1:4,i));
ba(:,i) = xhp(5:7,i); %Bias vectors updated w latest state vectors
bm(:,i) = xhp(8:10,i);
xhrec(:,i) = xhp(:,i);
Prec(:,i) = diag(Pp);

```

```
%Converting quaternion to Euler Angles (3-2-1) rotation sequence
rollq(i) = atan2((2*(q1(4,i)*q1(2,i) + q1(3,i)*q1(1,i))), (1-
2*((q1(2,i)^2)+(q1(3,i))^2)))*180/pi;
pitchq(i) = asin(2*(q1(4,i)*q1(2,i) - q1(1,i)*q1(3,i)))*180/pi
yawq(i) = atan2((2*(q1(4,i)*q1(1,i) + q1(2,i)*q1(3,i))), (1-
2*((q1(3,i)^2)+(q1(1,i))^2)))*180/pi;
end
```

B. Preliminary Calculations

B.1 Euler Angle Offsets

Electromagnetic and IMU sensors were placed in slightly different locations on the hand mannequin, electromagnetic on top of IMU. Offsets had to be calculated to account for difference in Euler angle output that may arise solely to this different placement. The differences could occur in different rotations of Euler angles so we used three separate tests; one for each Euler angle to calculate these offsets. The hand mannequin began in a prone position and would be slowly rotated to 90° in the *roll*, *pitch*, and *yaw* rotation angles. A *roll* rotation is hand supination/pronation, a *pitch* rotation is wrist flexion/extension, and a *yaw* rotation is radial/ulnar deviation. Yaw offset for finger yaw values were used by only measuring electromagnetic sensor yaw angles and comparing these to the hand yaw calculated from the glove.

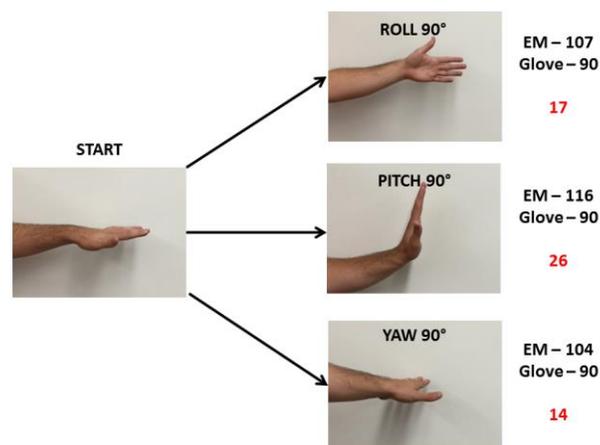


Fig. A.1. The hand mannequin begins prone, then moves to roll and is held and averaged over 10 seconds. This is repeated for each Euler angle rotation. Results of electromagnetic (EM) and glove data for the hand are shown next to figures to see how offset is determined. Glove data is subtracted from electromagnetic data to provide an offset.

B.2 Force Scale Factors

Forces during testing were sometimes experienced outside of the sensing area of the hand mannequin's finger sensors. In these cases, force plate data will be greater than forces measured by the glove. To accommodate for this, the force components of the glove will be multiplied by scale factors. As these scale factors will differ depending on the application of force, our group ran three tests to calculate scale factors in the x, y, and z-directions. The glove was first calibrated to a normal prone position for 10 seconds and then the hand mannequin was rotated and a normal force was applied on the Styrofoam sphere in a certain direction for 10 seconds. The average force component for force plate and glove data were averaged over the time of application.

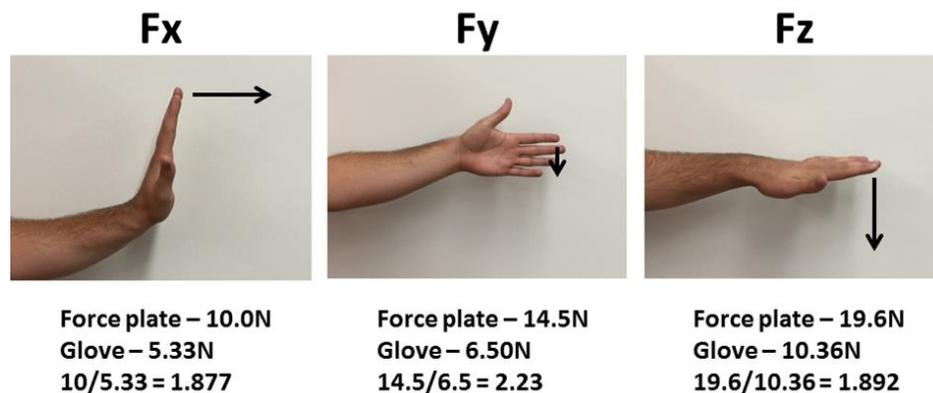
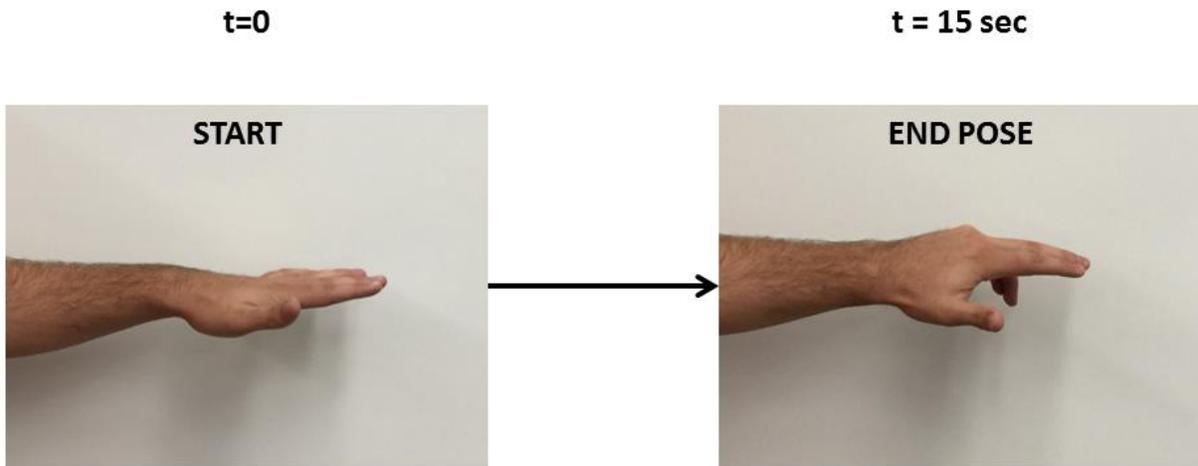


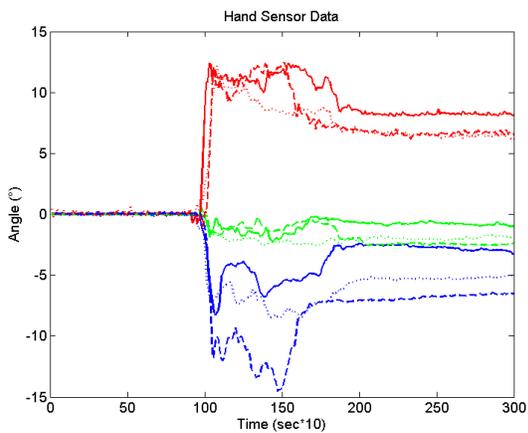
Fig. A.2. The hand mannequin is conformed to the above orientations. Force application direction is shown by the arrow. The F_y direction is out of the page. Force plate data averaged over the 10 seconds of application is divided by the force component value found from the glove. These offsets were then multiplied by each component for every test to account for a greater area of force application outside of the sensors located on the fingertips.

C. Example Data Output

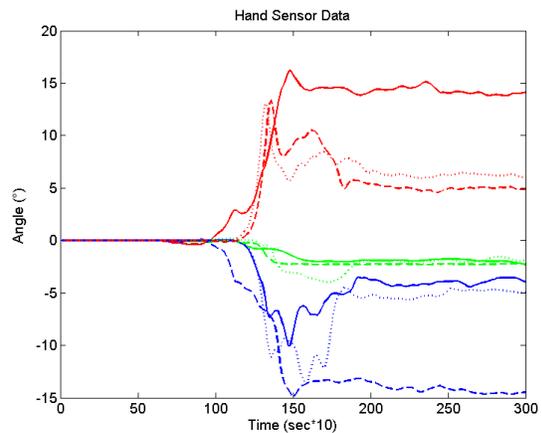
The output from three separate tests for one specific pose is shown below. This pose was called two finger reach. The plots show Euler angles for electromagnetic and glove sensors. When comparing the two systems we divided the poses into time phases.

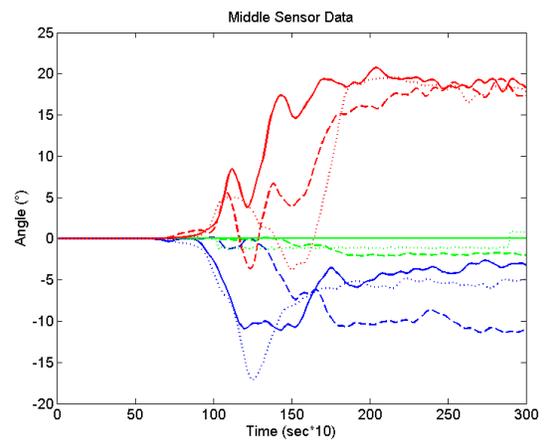
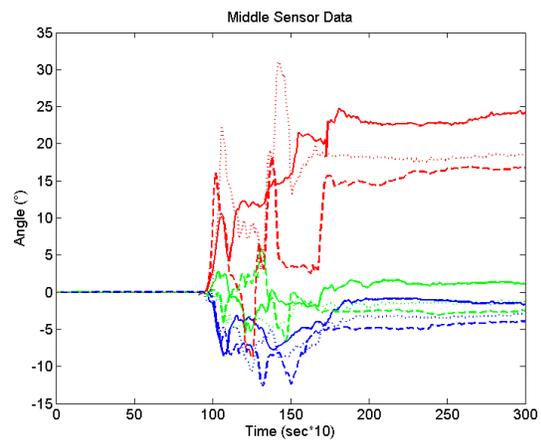
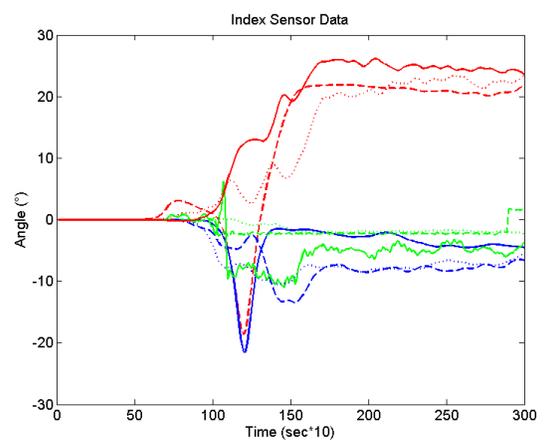
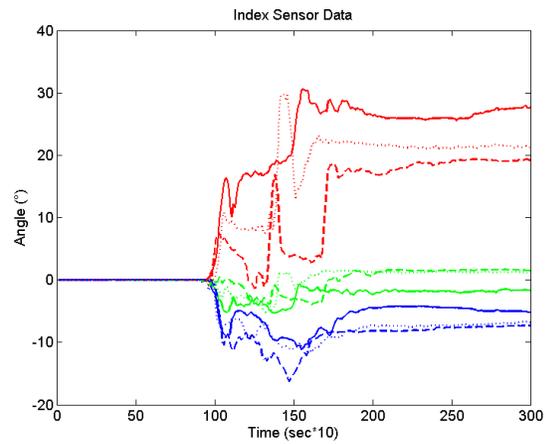
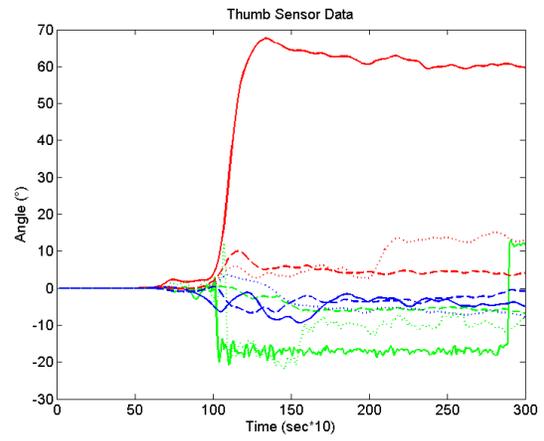
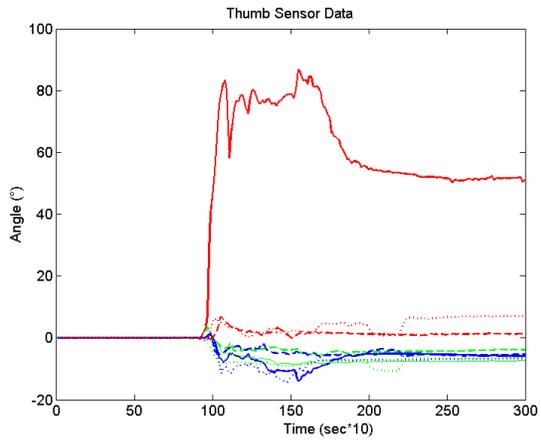


Electromagnetic Sensor Data



Glove Sensor Data





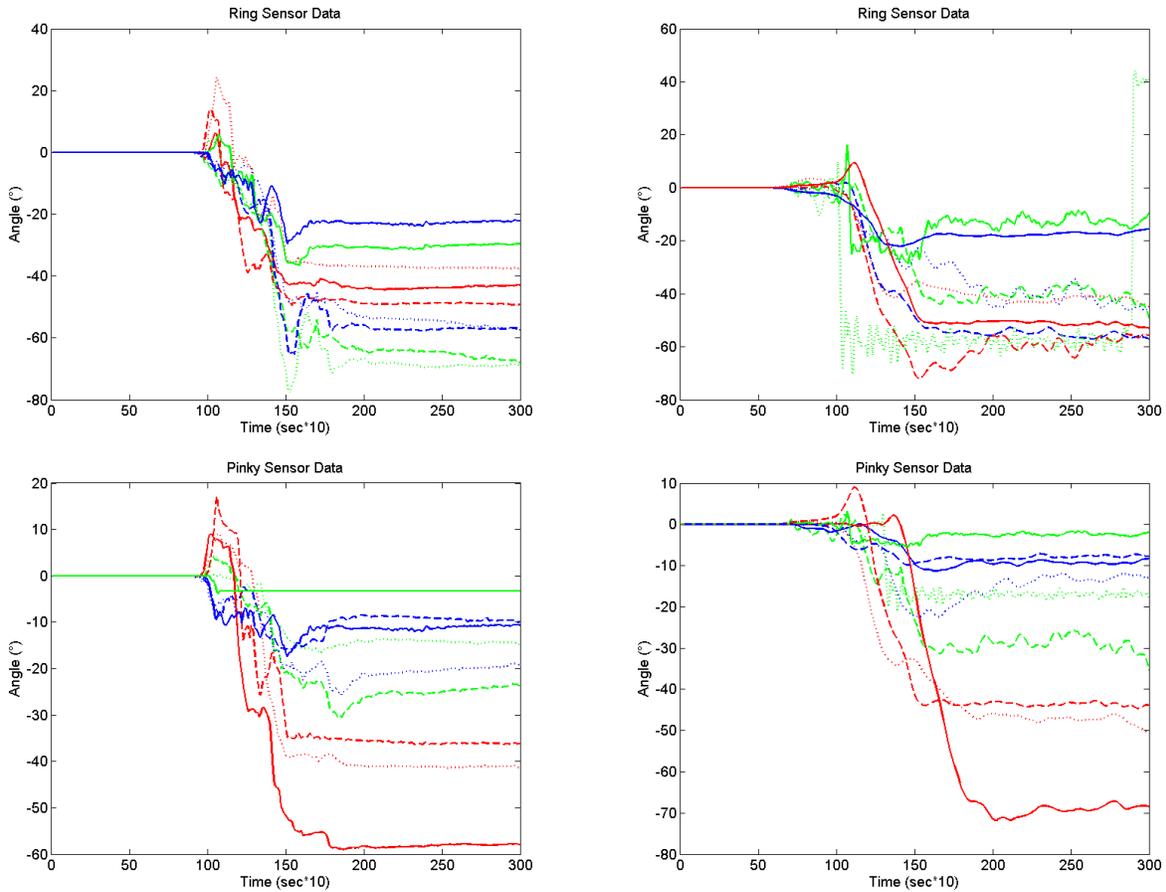


Fig. A.3. The left column of figures show the three Euler angles found from the test shown above all of the plots. The orientation is a two finger reach gesture and each plot is shown for each sensor. *Roll* angle is in blue, *pitch* is in red, and *yaw* angle is in green. Each plot shows the three repeated tests for the same orientation; the solid line corresponds to the first test, medium-dashed line is the second, and the smallest dashed line is the third test. Each angle is plotted against time. The time is in seconds*10 or tenths of seconds. The last five seconds of each test are not included in the plots as only the first 32 seconds are used in motion validation calculations.