Motion Tracking Glove for Human-Machine Interaction: Inertial Guidance

by

Thilakshan Kanesalingam

Electrical and Biomedical Engineering Design Project
Department of Electrical and Computer Engineering
McMaster University
Hamilton, Ontario, Canada

Motion Tracking Glove for Human-Machine Interaction: Inertial Guidance

by

Thilakshan Kanesalingam

Electrical and Biomedical Engineering

Project Partner: Arefin Shamsil
Faculty Advisor: Prof. A. Patriciu
Course Instructor: Prof. T. Doyle

Electrical and Biomedical Engineering Project Report (4BI6)
submitted in partial fulfillment for the degree of
Bachelor of Engineering

McMaster University
Hamilton, Ontario, Canada
April 9, 2010

Abstract

Impairment of mobility is an issue that negatively affects a large percentage of the general population. A mobile robotic assistive device may be greatly beneficial to many people with mobility issues. The objective was to develop a glove that tracks basic movements of the human hand, such that it can control a distant robotic assistive device for people with limited mobility. In this project the distant robotic assistive device was implemented as a virtual model, controllable with the user's natural hand movements while wearing a glove that covers the hand and fingers. The glove focused on tracking the most distinct types of relative movements of the hand; orientation, position and finger flexion.

This work presents the complete implementation of the first two tracking features: orientation and position. An inertial measurement unit (IMU) was implemented with the use of an accelerometer and gyroscope, to measuring acceleration and angular velocity. The outputs from these sensors were sent to a microcontroller (µC). An algorithm was developed and programmed to the µC to translate the sensor data into information on orientation and position in measurements of attitude and displacement. This information was then used as input to a custom virtual simulation of an assistive device that follows the user's hand movements, thus establishing accurate inertial guidance.

Key Words: motion capture, inertial navigation, gyroscope, accelerometer, virtual simulation

Acknowledgments

I would like to express my gratitude to Dr. Patriciu for providing countless hours of assistance, and also to Dr. Doyle for his guidance throughout the year. I would also like to thank Tyler Ackland for his technical assistance in the lab.

I would also like to thank my colleague Arefin Shamsil for the assistance and constant support provided throughout this course of this project.

Thilakshan Kanesalingam

Table of Contents

List of Figures, Tables & Equations

1. Introduction

Biomedical engineering involves the development of methods and equipment that improve the healthcare and quality of life of individuals. Impairment of mobility is an issue that negatively affects a large percentage of the general population. This is especially relevant to the elderly and patients recovering from surgical or drug treatments due to injury or disease. Such circumstances result in a person living partially or completely immobilized and dependent upon others for basic tasks around their home, which they were previously able to perform on their own. Regaining independence is crucial to ones recovery and ability to cope, and an assistive device available at home can be a very valuable healing aid.

A mobile robotic assistive device may be greatly beneficial to many people with mobility issues. [2,3,4,5] An important feature of an assistive device is that it must be easy to learn to operate. If a device could take natural hand movements and convert them into instructions, it would meet the needs of many people who are impaired in mobility. A personal assistive device that can help someone fetch and retrieve objects around their home will allow for the advantage of performing many activities independently and with ease, as an able-bodied person would do.

Aside from consumer use as a mobile robot controller, a device to track natural hand movements also has applications for professionals. For instance, the device can be used in a medical surgical training program to allow a surgeon to train for surgery in the virtual environment. It can also be used by a surgeon as a controller for tele-robotic surgical machinery. This would allow for a more realistic feeling of subject and surgical tool contact rather than using a traditional joystick or pen controller. Additionally, a hand movement tracking device can be used by a therapist as a physiotherapeutic tool to monitor the progress of upper limb motor recovery.

The objective of this project is to develop a glove that tracks basic movements of the human upper limb, such that it can control a distant robotic assistive device for people with

limited mobility. In this project the distant robotic assistive device will be implemented as a virtual model, controllable with the user's natural hand movements while wearing a glove that covers the hand and fingers. The human arm has seven degrees of freedom. The glove will focus on tracking the most distinct types of movements of the hand; it will have three primary tracking features: hand orientation, relative hand position and finger flexion.

My work, as outlined in this report, involves the complete implementation of the first two features: hand orientation and relative hand position. This includes acquiring data from a wearable glove equipped with an accelerometer and a gyroscope, programming a microcontroller to translate the input into instructions that can be read on a computer, developing a wired communication interface with a computer, and developing a computer-based simulation of an assistive device that responds to hand movements using the glove. My project partner, Arefin Shamsil, is working on the complete implementation of the third feature: finger flexion.

An inertial measurement unit (IMU) was implemented with the use of an accelerometer and gyroscope, for measuring acceleration and angular velocity. The outputs from these sensors were sent to a microcontroller (µC). µC programming was done to translate this data into information on position and orientation in measurements of displacement and angle of rotation. This information was then used as input to a custom virtual model program to control a basic virtual hand on a computer such that it follows the user's hand movements, thus establishing accurate inertial guidance.

2. Literature Review

A number of papers are available that discuss the use of a glove for various tracking purposes. Many of these papers focus on very sophisticated and precise tracking abilities for simulation and assessment purposes.[1-6] The tracking devices are often very large and intricate, making them impractical for personal household use. Cost is also a very important factor to consider. A particular paper of interest has been published regarding the implementation of a communications protocol between a commercially available digital glove (5^{th} Dimension Technology) and robotic arm (OWI-007) via a network communication.[7] Although attention was made towards the cost benefit of using the existing technologies, it still remains too expensive to be considered a mainstream personal assistive device for common middle-class households. There lacks a low cost and wearable glove that can be used by an average consumer. This project stemmed from the need for a small wearable glove that can track distinguished hand movements enough to perform specific tasks on a remote object, namely a robotic assistive device.

Tracking the motion of the human hand and arm has been investigated by many researchers that have many applications and rehabilitation.[8-21] Inertial sensors are used widely for tracking arm movements and they have proven to be effective.[22-27] There are also a number of papers available regarding robotic arms, each with their specific purpose for implementation. One in particular focused on acting as an assistive device for eating food, for those affected by cerebral palsy.[28] Controlling a remote robotic assistive device with natural hand movements is the basis of this project and many robotic devices currently in development can be coupled with the low-cost tracking glove proposed here. A personal device may be more beneficial to some people than a service dog if cost is a limitation. Although extensive work has been done with robotic devices, very few papers focus on robotic assistive devices that are low in cost such that they can be considered an average household electronic item. A primary target of this project is to make the device low-cost and affordable for the average middle-class individual.

3. Methodology of Solution

Motion capture methods can be categorized into two main types of systems: optical and non-optical. Due to the need for a stand-alone product without external sensors or equipment, the non-optical route has been chosen in this project. In this category of systems, there remain a few more options: inertial, magnetic and mechanical motion capture techniques. Mechanical motion capture systems generally track the articulation of mechanical parts placed at body joint angles, therefore this method would not be ideal for hand tracking. Magnetic motion capture systems measure the relative magnetic flux of three orthogonal coils on a transmitter and receiver. This method proved to be too expensive to be considered. However, inertial motion capture technology is based on the use of inertial sensors and does not require external cameras, emitters or markers. In addition, there have been recent improvements in the performance of small and lightweight micro-machined electromechanical systems (MEMS), and thus inertial sensors are more widely used for the application of human motion capture.[29] Hence, the method chosen for tracking relative hand orientation and position was inertial guidance.

Inertial navigation is a self-contained navigation technique in which measurements provided by accelerometers and gyroscopes are used to track the position and orientation of an object relative to a known starting point, orientation and velocity.[29] An inertial measurement unit (IMU) typically consists of orthogonally placed accelerometers and gyroscopes, each measuring acceleration or angular velocity in one axis.

The purpose of this project was to develop a complete implementation of two tracking features of the human hand: relative hand orientation and position. The course of action included acquiring data from a wearable glove equipped with a multi-axis accelerometer and gyroscope and sending the data from these sensors to a microcontroller. The algorithm involved with translating data on angular velocity and acceleration into relative orientation and position was programmed into the microcontroller. A wired communication interface was then established with a computer and the desired information on orientation and position was sent to a computer for a visual output. Subsequently, a computer-based simulation of a virtual assistive device that responds to hand movements with the glove was developed. Upon completion, the work was

collaborated with Arefin Shamsil's implementation of the third tracking feature: finger flexion. The result is a wearable glove that tracks basic movements of the human upper limb, such that human-machine interaction is established using natural hand movements. A system block diagram of the methodology is shown in Figure 1.

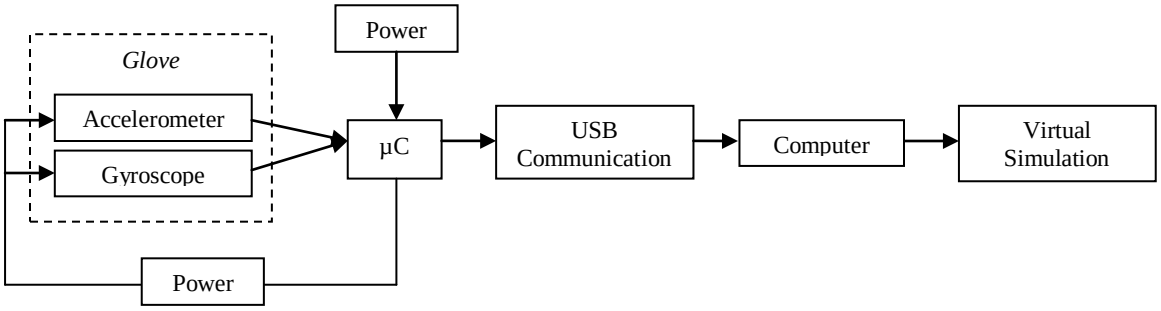

Figure 1: System Block Diagram

3.1 Goals

An important feature of an assistive device is that it must be easy to learn and operate. If a device could take natural hand movements and convert them into specific actions to perform specific tasks, it would meet the needs of many people. A personal assistive device that can help someone fetch and retrieve objects around their home will allow for the advantage of performing many activities independently and with ease, as an able-bodied person would do.

The primary goal of this project was to develop an assistive device for a person who is immobilized from the waist-down and is dependent upon others for basic tasks around their home. Regaining independence is crucial to a patient's recovery and ability to cope, and an assistive device available at the patient's home can be a very valuable healing aid. The human arm has seven degrees of freedom. The glove focused on tracking the most distinct types of movements of the hand. The final product, after collaboration with partner work, was able to track hand orientation, position, and finger flexion of the thumb and index finger.

Although the aim of this project was to control a virtual model of a remote robotic arm, the work presented here demonstrates that the fundamental principles behind human-machine interaction through inertial guidance have been accomplished and the method is accurate. Through some extension of the work, it is possible to program a mobile robotic assistive device such that it responds to movements from the gloves sensors.

In order for this design to be a viable solution as a personal assistive device, the primary focus was to minimize costs while maximizing performance. Due to this importance, the original plan for a wireless communication protocol was discarded and a wired serial protocol was deemed suitable. Also, due to the complexity of virtual modeling and the high cost of software licenses, focus was placed on using an open-source software distribution for the virtual model simulation.

3.2 Cost

The total cost for this part of the project was $111.42. This includes costs associated with components used for testing purposes throughout the scope of the project. The research and development cost was $15.06, while the cost of all that used in the final product was $96.36.

Table 1: Equipment Costs

Item	**Supplier**	**Cost ($)**
SFE Ardupilot Sensor Board	Robot Shop	58.01
Arduino Duemilanove µC Board	Creatron Inc	33.89
USB Cable	Creatron Inc	4.46

Table 2: Research & Development Costs

Item	**Supplier**	**Cost ($)**
Breadboard Kit	Sayal Electronics	13.56
Voltage Regulator (LE33CZ-TR)	DigiKey	1.50

3.3 Time Line

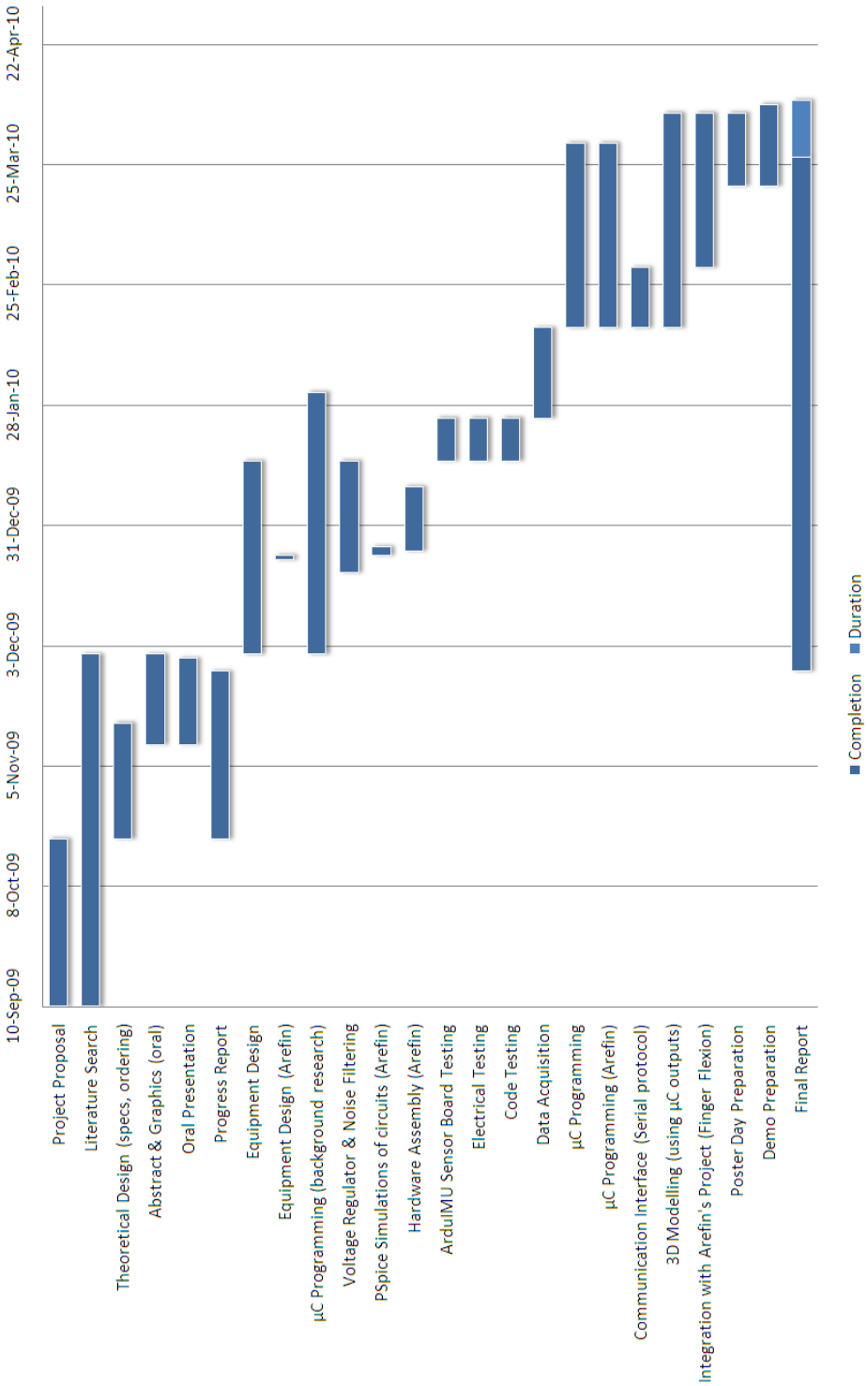

Figure 2: Tasks vs. Time graph for design and implementation of glove

3.4 Support Services

Much of the assembly required for this design was accomplished in a lab where soldering and basic electrical components were available. The IEEE student branch workshop at McMaster University had these facilities readily available for students.

Laboratory technicians were not required for the design and assembly of this project. All parts were assembled in the IEEE student branch workshop located in the Information Technology Building at McMaster University.

3.5 Safety

To eliminate many of the hazards associated with electrical equipment, this design employed the use of only low-voltage power supplies. An onboard battery module was used to house the battery supply. A power management system was included as a safety measure to control the power source. While working in the IEEE lab, safety procedures (appropriate grounding) and the use of safety equipment (goggles) were practiced to ensure safe execution of all project related tasks.

3.6 Approval

Work on this project was done with the help of Dr. Patriciu from the Department of Electrical and Computer Engineering at McMaster University. Details of the project were regularly discussed and Dr. Patriciu committed to guide in project-related matters and grade performance.

4. Design Procedures

The orientation of the hand was determined using a gyroscope embedded on the glove. A gyroscope measures orientation based on the principles of angular momentum (see Appendix A.1). A single-axis gyroscope was used as a yaw rate (z-axis) sensor. Relative hand position was used to control the movement of a distant robotic device. This was tracked using a triple-axis accelerometer embedded on the glove, as a x,y,z-axes acceleration sensor. An accelerometer measures the acceleration experienced relative to freefall (see Appendix A.2).

To simplify issues of calibration, a combination board with these two components installed (ArduIMU Sensor Board, GPS-09372) was chosen. This board was equipped with an ADXL335 accelerometer and LISY300AL gyroscope. All filtering components were embedded. Since there were no on-board regulators, a 3.3V voltage regulator (LE33CZ-TR) was required for testing purposes.

The chosen gyroscope was the LISY300AL, a single-axis ±300°/s analog output yaw rate gyroscope. This sensor required a 2.7 V to 3.6 V supply. The output from this sensor was an analog output voltage. A z-axis rate produced positive-going output for counterclockwise rotation.

The chosen accelerometer was the ADXL335, a triple-axis analog output accelerometer with a ±3g range. This sensor operated on a 1.8V to 3.6V supply, and provided an analog output voltage proportional to acceleration.

The chosen microcontroller board was the Arduino Duemilanove. The board is equipped with an ATmega328 microcontroller. The Arduino has 6 analog inputs, each of which provides 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts. It comes pre-programmed with a bootloader so that new code can be uploaded without the use of an external hardware programmer. The Arduino board also has an on-board FTDI chip which generates a 3.3 volt supply and this was used as the sensor’s power supply in the final design.

Four analog outputs from the sensor connect to the analog inputs on the Arduino. These outputs were then processed in the µC. After processing, the data was passed to a computer via a serial communication protocol. Once received on the computer, a real-time virtual model of a hand was created to display the output simulation.

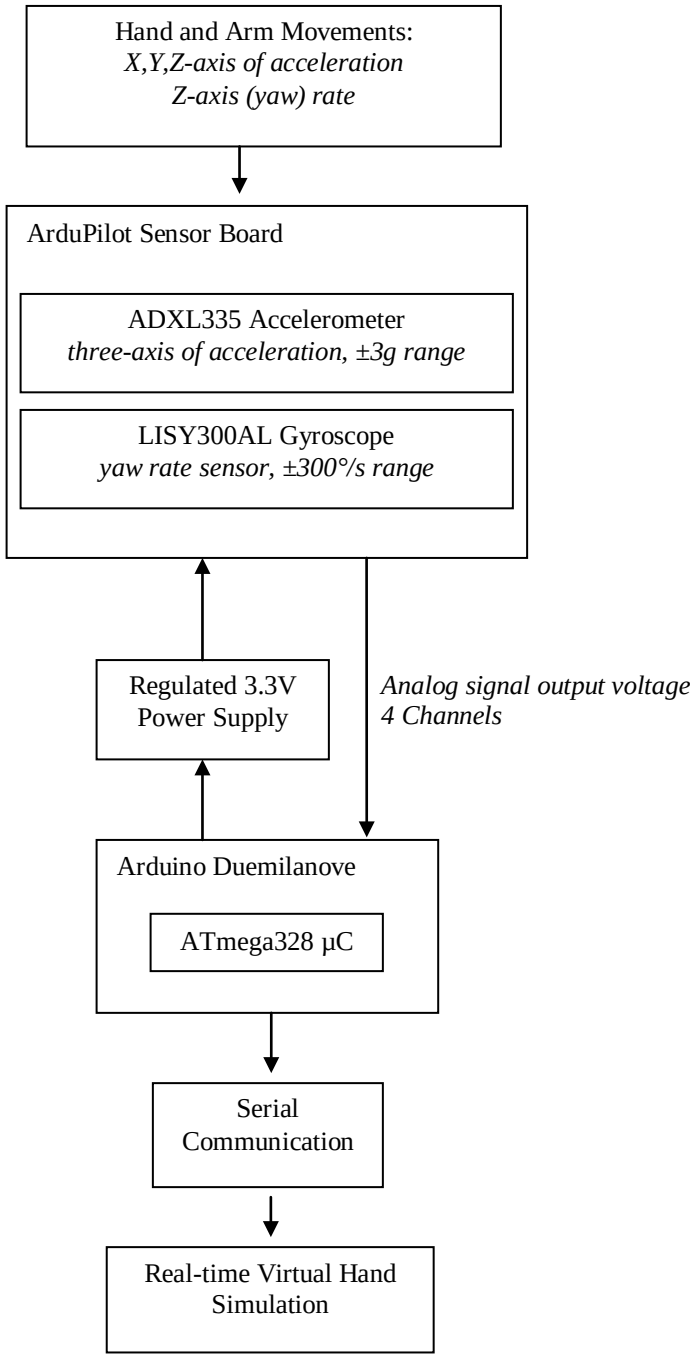

Figure 3: Detailed System Block Diagram

A row of right-angle break away headers were soldered to the sensor board such that it could easily be connected to a breadboard or a cable. Subsequently, a six-wire cable was created for easily connecting the sensor board to the Arduino. One wire was used for each of the power and ground connection, one for the gyroscope output and three for the accelerometer outputs. This allowed for a flexible range of motion while testing the sensor board.

The connection setup is shown in Figure 4.

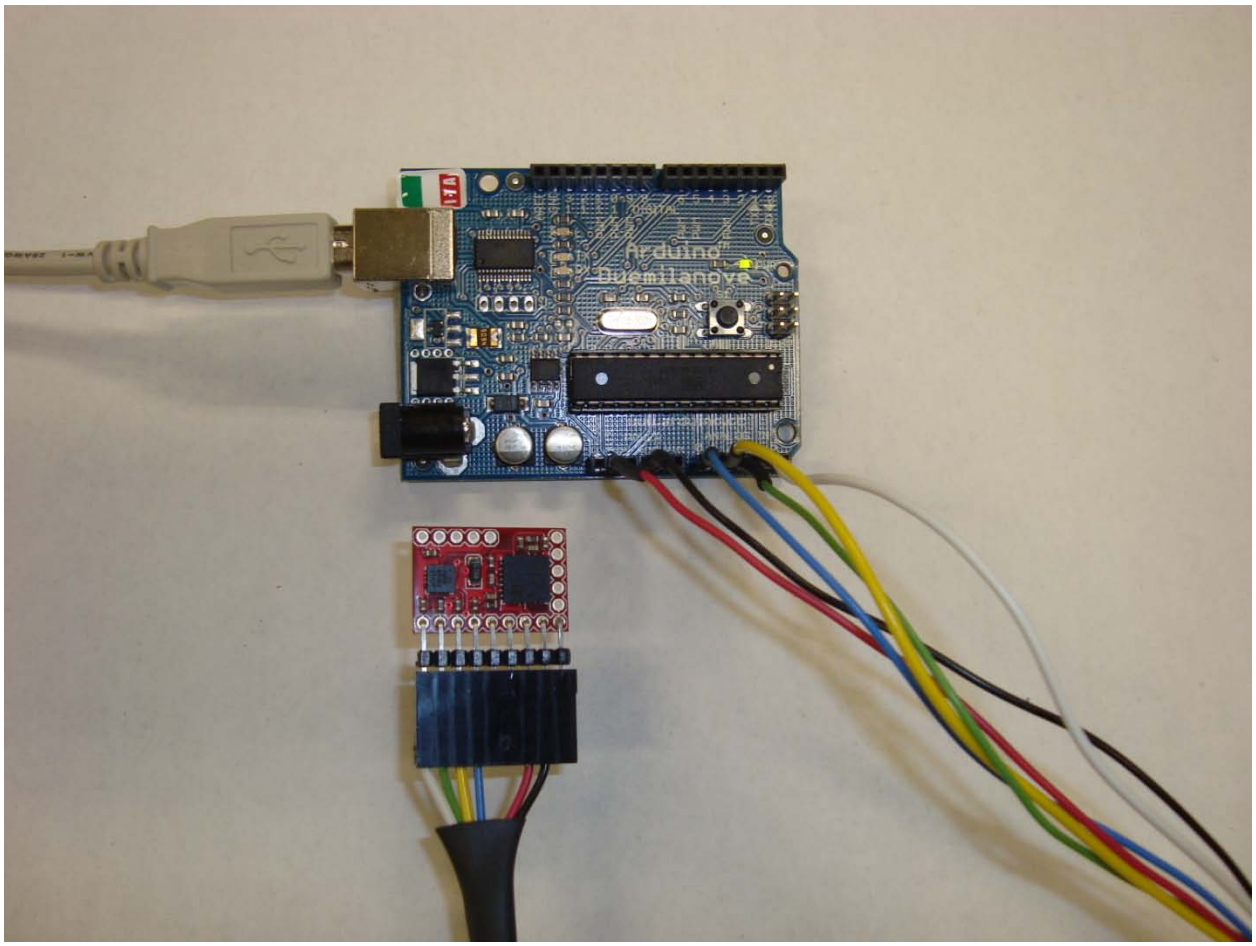

Figure 4: Connection Setup of IMU and µC

The design involved the interpretation of data given by the IMU, comprised of the gyroscope and accelerometer. Nearly all IMU's fall into one of two categories: stable platform systems and strapdown systems. The difference lies in the frame of reference in which the sensors operate. In this design the sensors were mounted onto a glove that the user wore. Therefore, the implementation required was that of a strapdown system in which the sensors outputted data in the base frame of reference rather than the global frame of reference. The strapdown system is shown in Figure 4. The algorithm for the strapdown system developed based on a technical report on inertial navigation by Oliver J Woodman.[29]

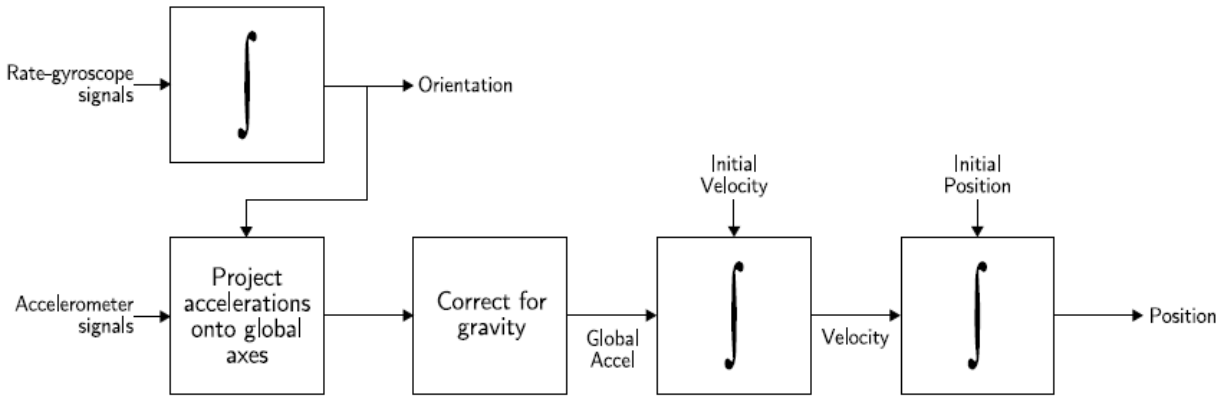

Figure 5: Strapdown Inertial Navigation Algorithm[29]

The gyroscope provides yaw rate angular velocity. The output from this sensor was integrated to obtain an angle of orientation. This orientation was then used to develop a rotation matrix that is used to project the acceleration signals from the accelerometer onto the global frame of reference. Next, the acceleration was normalized using a steady state value determined during a calibration routine of the sensor. Following this correction for gravity, the data was double integrated to get position. The algorithm for integration and filtering was developed based on a report by Kurt Seifert and Oscar Camacho.[30]

4.1 Gyroscope Integration

The first step was to determine orientation given the output from the gyroscope. The gyroscope outputs angular velocity. It is known that derivative of position gives velocity:

$$d\Theta = angular\ rate = gyroscope\ output$$

The inverse of this equation yields:

$$\int angular\ rate = \int gyroscope\ output = \Theta$$

Thus, integrating the gyroscope output gives the attitude angle.[31]

To perform this integration, the output from the sensor (0-5 V) was connected to an analog input on the microcontroller. The data was sent to a 10-bit ADC (analog to digital converter) and the result was a time stream of ADC resolution (0-1024) proportional to the angular velocity. As an example, the sensor was turned 90° counter-clockwise and back, then turned 90° clockwise then back. The ADC resolution as a function of time for this movement is shown in Figure 4.

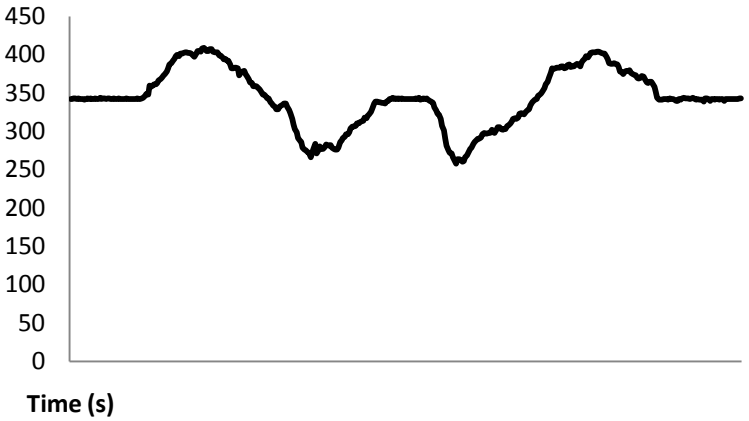

Figure 6: ADC Resolution vs. Time for Gyroscope

In order to integrate this data to obtain orientation, the graph must be normalized such that there are positive and negative values, otherwise the integration would sum infinitely. The data was normalized by subtracting the steady state value of the sensor at each sample. The ADC resolution was then converted to radians per second and the result is shown in Figure 5.

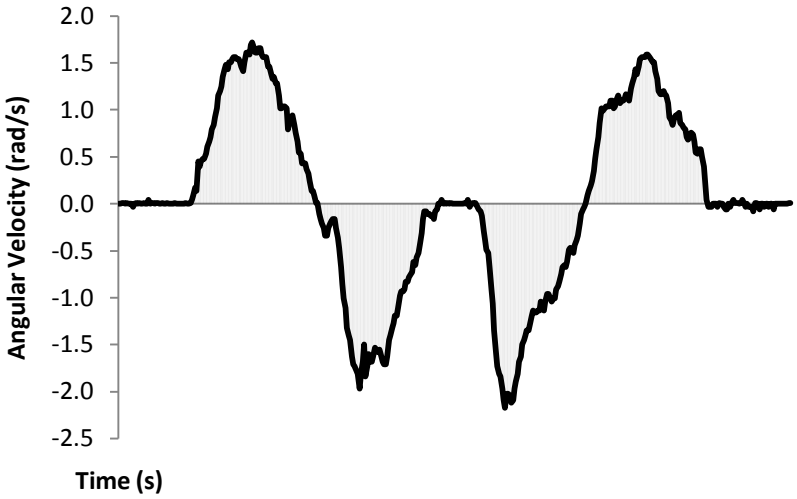

Figure 7: Angular Velocity vs. Time

The integral is defined as the area under the curve. Essentially, the integration is the sum of the areas comprised of rectangles with a very small width (Δx).

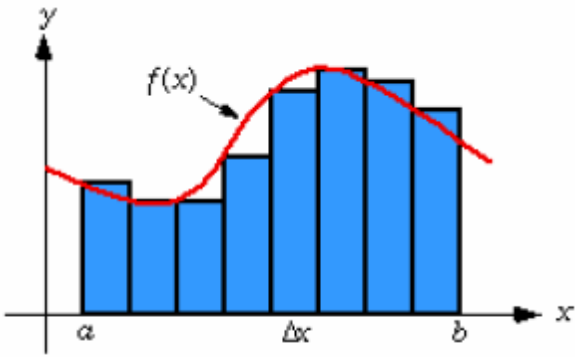

Figure 8: Integral as Area under the Curve[30]

To reduce errors due to sampling losses, a first order approximation (trapezoidal method) was used for all integrations. The area is taken as the combination of two smaller areas.

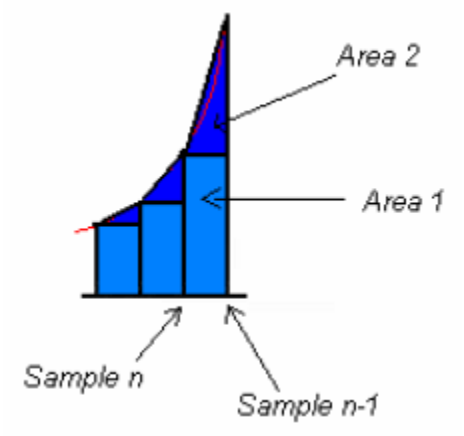

Figure 9: First Order Approximation[30]

The first area is the value of the previous sample given by a rectangle. The second area is a triangle formed between the previous sample and the current sample.

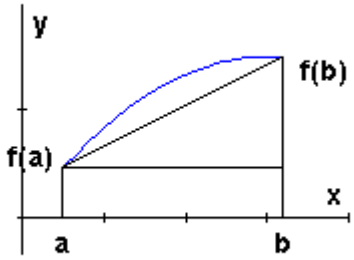

Figure 10: Trapezoidal Rule[32]

The integration is thus given by the following equation:

Equation 1: Trapezoidal Method of Integration

$$\int_a^b f(x)dx = (b-a)f(a) + \frac{1}{2}(b-a)[f(b) - f(a)]$$

Performing this integration on the data presented in Figure 5 yielded a plot of the angle in radians shown in Figure 9. The movement represented in this plot is identical to the angular movement exerted on the sensor; a counter-clockwise rotation of 90°, return to initial position, clockwise rotation of 90° and then a return to initial position.

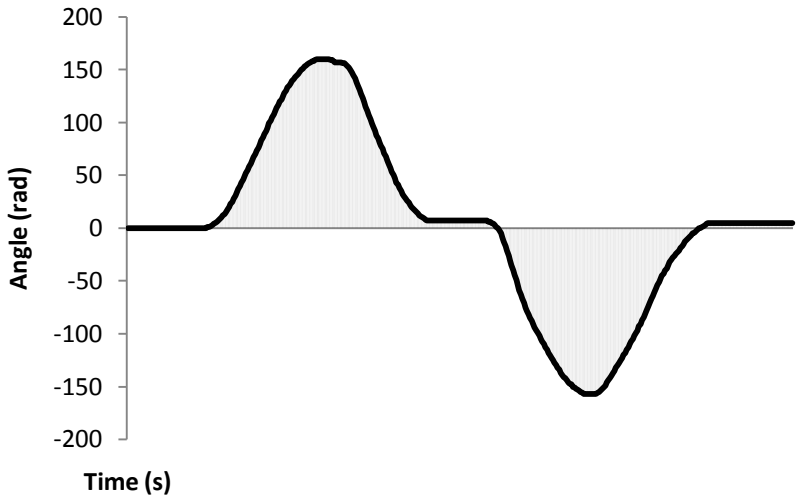

Figure 11: Angle vs. Time

4.2 Rotation Matrix Computation

The sensors in this design were mounted rigidly onto a glove which fit over the user's hand, thereby representing a strapdown system. However, this entailed that the output signals were measured in the body frame rather than the global frame (1) which could result in an incorrect representation of relative position. For example, if a user was to move the glove forward and backward in the x axis, rotate 90°, then perform the same movement along the same line; the sensor would track movement in two orthogonal axes, even though he user only moved along one axis.

To correct this, the acceleration vector had to be projected from the base frame of reference to the global frame of reference.

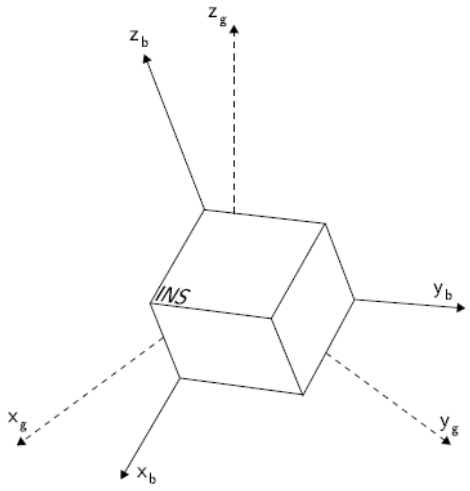

Figure 12: Body and Global frames of reference[29]

Once orientation was known by integrating the angular velocity provided by the gyroscope signal (section 4.1), the accelerometer signals were resolved into global coordinates using this known orientation. This was done by multiplying the acceleration vector by a rotation matrix for each sample. The projected acceleration was then integrated to obtain relative position in the global frame.

A rotation matrix had to be updated during each sample of the sensor data. Since a single axis gyroscope was used in this project and hence only one angle (yaw rate, rotation about the z axis) could be achieved, an 'attitude update' matrix was created using an elementary rotation about z.

Equation 2: Elementary rotation matrix about z

$$R_z = \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

This attitude update matrix is post multiplied to the old rotation matrix since the projection is onto the current frame.

Equation 3: Post multiplication of matrices

$$R_{global}^{t_{k+1}} = I \cdot R_{t_0}^{t_1} \cdot R_{t_1}^{t_2} \cdot R_{t_2}^{t_3} \cdot \ldots \cdot R_{t_k}^{t_{k+1}}$$

The resulted in a rotation matrix which was used to project acceleration from the base frame to the global frame.

4.3 Accelerometer Integration

Using the same first order approximation (trapezoidal method) technique as used on the gyroscope output (section 4.1), the accelerometer output was double integrated to obtain position. As an example, the sensor was moved from point A to B in a straight line. The ADC resolution as a function of time for this movement is shown in Figure 11.

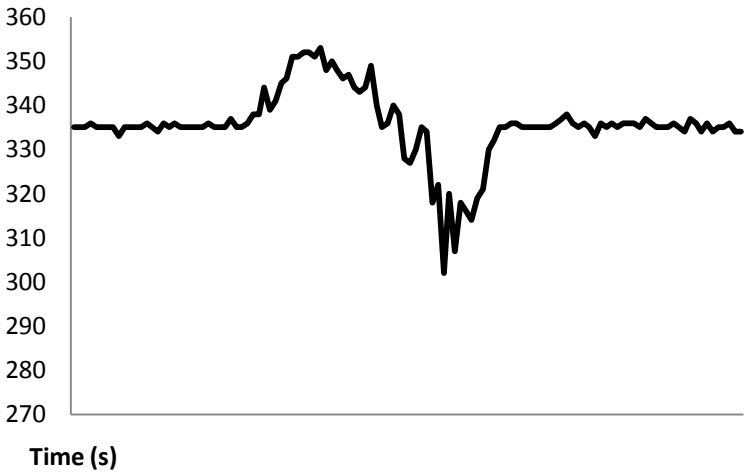

Figure 13: ADC Resolution vs. Time for Accelerometer

Following this, the data was normalized by subtracting the steady state value of the sensor at each sample. The ADC resolution was then converted to meters per second squared and the result is shown in Figure 12.

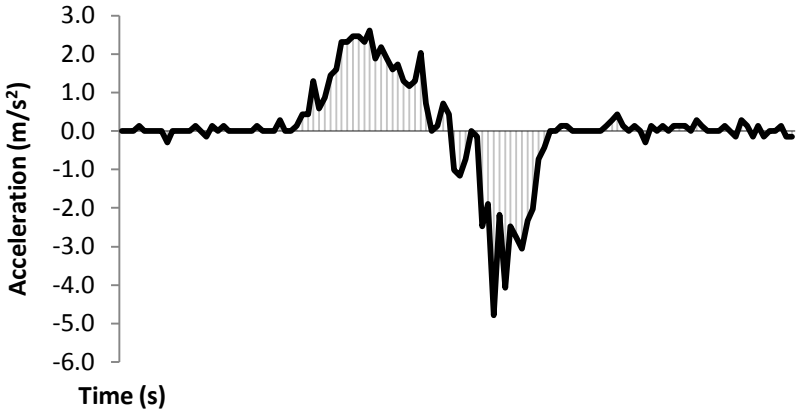

Figure 14: Acceleration vs. Time

Performing an integration of the data presented in Figure 12 yielded a plot of the velocity in meters per second shown in Figure 13.

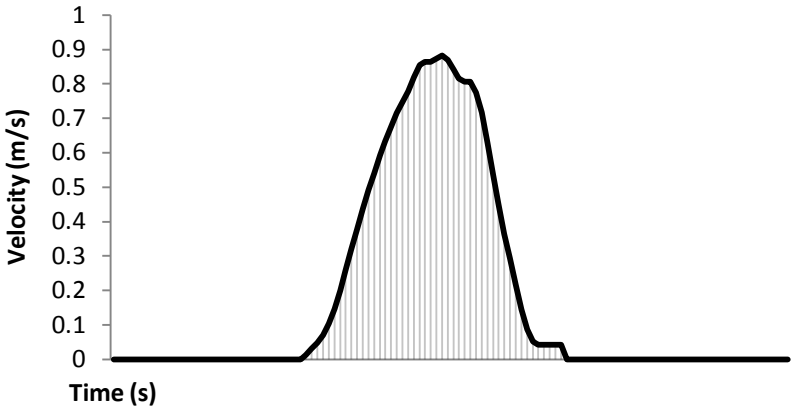

Figure 15: Velocity vs. Time

A second integration of this data yielded a plot of position in meters shown in Figure 14.

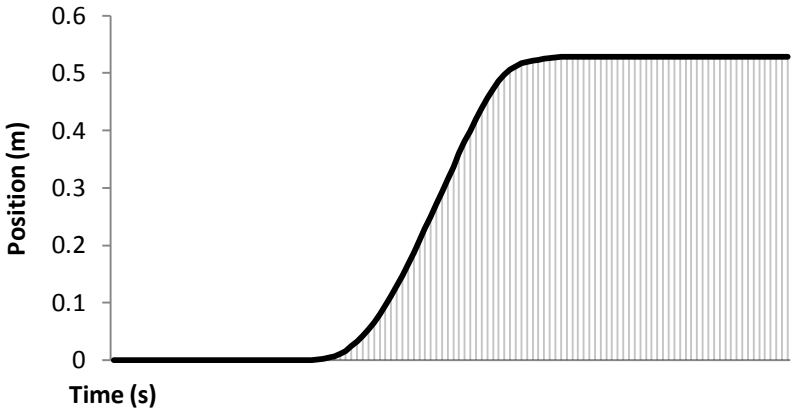

Figure 16: Position vs. Time

The movement represented in this plot was identical to the movement exerted on the sensor; movement from point A to point B in a straight line.

4.4 Code Explanation

The following sections give an analysis behind the design and implementation for programming the microcontroller. Source code is provided in Appendix A.3.

4.4.1 Calibration

The ADC counts from the µC are proportional and the steady state value for each sensor output is dependent on the initial orientation of the sensor. This is due to the intrinsic nature of the sensors sensitivity to a gravity component affecting a particular axis. To compute an accurate steady state value for each sensor output, after the µC has been programmed the samples from each output are read directly from the analog input pin on the µC and averaged.

```
do{ //accumulate samples
  ss_gyro_x = ss_gyro_x + analogRead(0);
  ss_accel_x = ss_accel_x + analogRead(1);
  ss_accel_y = ss_accel_y + analogRead(2);
  ss_accel_z = ss_accel_z + analogRead(3);
  count1++;
}while(count1!=500); //500 times

//average the samples
ss_gyro_x = ss_gyro_x/count1;
ss_accel_x = ss_accel_x/count1;
ss_accel_y = ss_accel_y/count1;
ss_accel_z = ss_accel_z/count1;
```

4.4.2 SI Unit Conversions

The µC, along with several other components on the Arduino board, are powered by a regulated 5V supply; this value is the reference voltage used for calculations. Each of the analog input pins provides 10 bits of resolution (i.e. 1024 different values).[33] To convert the ADC resolution into SI units of radians per second for the gyroscope output and meters per second

squared for the accelerometer outputs, the ADC resolution must first be converted to an ADC voltage resolution.

Equation 4: ADC resolution to ADC voltage resolution

$$ADC\ x\ \frac{Vref}{1024} = V$$

To convert from ADV voltage resolution to SI units the voltage must be divided by the sensitivity of the sensor given in the components data sheet. The gyroscope sensitivity is given in degrees per second so the final value must be converted to radians per second.

Equation 5: Degrees per second to Radians per second

$$\frac{(degrees)^\circ}{s} x \frac{\pi}{180^\circ} = radians/second$$

The accelerometer sensitivity is given in standard gravity and must be converted to meters per second squared using the nominal acceleration due to gravity at the Earth's surface at sea level which is 9.80665m/s^2.

Equation 6: Standard gravity to Meters per second squared

$$gx\frac{9.80665\ m}{s^2} = m/s^2$$

```
double make_radians_per_sec(double ADC_angular_rate){
  double Vref = 5; //arduino ADC 5V ref voltage
  double sensitivity = 0.0033; // 3.3mV/(degrees/sec)
sensitivity (from gyro data sheet)
```

```
  //convert ADC value to voltage and divide by sensitivity
  return ((ADC_angular_rate*Vref/1024)/sensitivity)*(3.14/180);
}

double make_metres_per_sec2(double ADC_acceleration){
  double Vref = 5; //arduino ADC 5V ref voltage
  double sensitivity = 0.33; // 0.33V/g sensitivity ratiometric
when Vs = 3.3V (from accel data sheet)

  //convert ADC value to voltage and divide by sensitivity
  //convert standard gravity g to m/s^2 by multiplying by
9.80665
  return ((ADC_acceleration*Vref/1024)/sensitivity)*9.80665;
}
```

4.4.3 Window of Discrimination

Very small movements from the gyroscope and accelerometer are considered unintentional error components. Error can accumulate once these values are integrated to obtain velocity. To reduce errors in measurement caused from this, the sensor's output data is checked to see if it falls within a 'window of discrimination' to separate valid data from invalid data.

```
double w_d = 0.2; //window of discrimination for no-movement
condition: 0.2 rad/s

//apply discrimination window
if ((x_angular_rate[1] <= w_d) && (x_angular_rate[1] >= -w_d))
{x_angular_rate[1] = 0;}
```

In the case of the accelerometer, another window is added to minimize errors due to tilt. Tilting the sensor will cause one of the axes to include a gravity component and thus acceleration will rapidly increase even though the sensor is not moving. To avoid this, a maximum acceleration value is set in a window of discrimination so that acceleration due to an unwanted gravity component is ignored.

```
double wd_max = 2; //window of discrimination for gravity
condition:     2 m/s^2

//if acceleration is very fast, its likely due to an unwanted
gravity component, so ignore it
if ((x_acceleration[1] >= wd_max) || (x_acceleration[1] <= -
wd_max)) {x_acceleration[1] = 0;}
```

4.4.4 Gyroscope Integration

Integration of the angular velocity is performed using a first order approximation. The trapezoidal method sums the area of rectangle and triangle formed under the angular velocity curve in one sample. Before integrating, angular velocity is obtained by first converting the ADC resolution to SI units of radians per second (section 4.4.2.), and a window of discrimination is applied (section 4.4.3). The attitude angle is converted from radians to degrees for serial output. The resulting angle is also used to create a rotation matrix that is used to project the acceleration from the base frame of reference to the global frame of reference.

```
//subtract the zero-rate level to obtain positive and negative
angular rate
//convert from proportional ADC counts to radians per second
x_angular_rate[1] = make_radians_per_sec(x_gyro) -
make_radians_per_sec(ss_gyro_x);

//apply discrimination window
if ((x_angular_rate[1] <= w_d) && (x_angular_rate[1] >= -w_d))
{x_angular_rate[1] = 0;}

//integrate using first order approximation (trapezoidal method)
// = rectangle area + triangle area
x_angle[1] = x_angle[0] + (x_angular_rate[0] +
((x_angular_rate[1] - x_angular_rate[0])/2))*dt;

double angle_vector[3] = {x_angle[1]-x_angle[0],0,0}; //angle
vector for integrated angular velocity
//if the angle is non-zero (with window) create a rotation
matrix
if (angle_vector[0] != 0) {UpdateRotationMatrix(angle_vector);}
```

4.4.5 Global Frame Projection

In order to project the acceleration from the base frame of reference to the global frame of reference, the current orientation from the gyroscope integration is used to develop a rotation matrix. A rotation matrix is any orthogonal matrix whose determinant is equal to 1 (Appendix A). The orientation is presented as an angle vector and when it is multiplied by the rotation matrix, the direction of the vector changes but not its magnitude. An 'attitude update' matrix can be created using an elementary rotation about z (section 4.2).

Equation 2: Elementary rotation matrix about z

$$R_z = \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

This attitude update matrix is post multiplied to the old rotation matrix since the projection is onto the current frame. The result is a rotation matrix which can be used to project acceleration from the base frame to the global frame.

```
//attitude update matrix is an elementary rotation matrix about
z
double attitude_update[3][3] = {{cos(angle[0]),-
sin(angle[0]),0},
                                {sin(angle[0]),cos(angle[0]),0},
                                {0,0,1}};

//Compute: Current Rotation Matrix = Old Rotation Matrix *
Attitude Update Matrix
//(multiplication of two 3x3 matrices)
for (i=0; i<3; i++) {
  for (j=0; j<3; j++) {
    sum=0;
    for (k=0; k<3; k++) {
      sum = sum + old_R[i][k]*attitude_update[k][j];
      current_R[i][j] = sum;
    }
  }
}
```

4.4.6 Accelerometer Integration

For the accelerometer sensor outputs, SI unit conversion and windowed mechanical filtering are performed in the same way as done for the gyroscope outputs. Trapezoidal integration is performed twice, once to obtain velocity and again to obtain position. The position in the x,y and z axes are transferred to a vector of acceleration in the base frame. This vector is multiplied with the updated rotation matrix (section 4.4.5) to effectively change the direction of the vector such that the acceleration is relative to a global frame of reference. For error reduction, the acceleration is sent to a function for sampling to determine is movement has ended. The final position is converted to cm for serial output.

```
//subtract the zero-rate level to obtain positive and negative
acceleration
//convert from proportional ADC counts to metres per second
squared
x_acceleration[1] = make_metres_per_sec2(x_accel) -
make_metres_per_sec2(ss_accel_x);

//apply discrimination window for no-movement condition
if ((x_acceleration[1] <= w_d) && (x_acceleration[1] >= -w_d))
{x_acceleration[1] = 0;}

//if acceleration is very fast, its likely due to an unwanted
gravity component, so ignore it
if ((x_acceleration[1] >= wd_max) || (x_acceleration[1] <= -
wd_max)) {x_acceleration[1] = 0;}

//move acceleration signal in the base frame of reference to a
new vector
base_accel[0] = x_acceleration[1];
base_accel[1] = y_acceleration[1];
base_accel[2] = z_acceleration[1];

//project acceleration into the global frame of reference:
matrix and vector multiplication
for (i=0; i<3; i++) {
  for (j=0; j<3; j++) {
    global_accel[i] = global_accel[i] +
current_R[i][j]*base_accel[j];
  }
```

```
}

//integrate using first order approximation (trapezoidal method)
// = rectangle area + triangle area
//double integrate each axis of acceleration to get position

//first x integration
x_velocity[1] = x_velocity[0] + (x_acceleration[0] +
((x_acceleration[1] - x_acceleration[0])/2.0))*dt;
//second x integration
x_position[1] = x_position[0] + (x_velocity[0] + ((x_velocity[1]
- x_velocity[0])/2))*dt;

//same for y

//same for z

//check for end of movement
Accel_Movement_End_Check();
```

4.4.7 Movement End Check

A typical movement from point A to point B will result in an acceleration and deceleration component in an acceleration versus time plot. Since the integration of acceleration gives velocity, it is expected that the velocity should return to zero after reaching point B. However in a real world scenario, the area above and below zero of an acceleration versus time plot are not exactly zero, so the velocity will not necessarily become zero at the end of a movement. This results in a sloped position when performing the second integration. To avoid this, the acceleration is constantly read and if it is zero for more than five samples, the velocity if forced to zero, indicating the end of a particular movement.

```
//count the number of accel samples that equal zero
if (x_acceleration[1]==0) {countx_accel++;}
else {countx_accel = 0;}
//if this number exceeds 5, we can assume that velocity is zero
if (countx_accel >= 5) {
  x_velocity[1]=0;
```

```
  x_velocity[0]=0;
}

//same for y
if (y_acceleration[1]==0) {county_accel++;}
else {county_accel = 0;}
if (county_accel >= 5) {
  y_velocity[1]=0;
  y_velocity[0]=0;
}

//same for z
if (z_acceleration[1]==0) {countz_accel++;}
else {countz_accel = 0;}
if (countz_accel >= 5) {
  z_velocity[1]=0;
  z_velocity[0]=0;
}
```

4.5 Software Flowchart

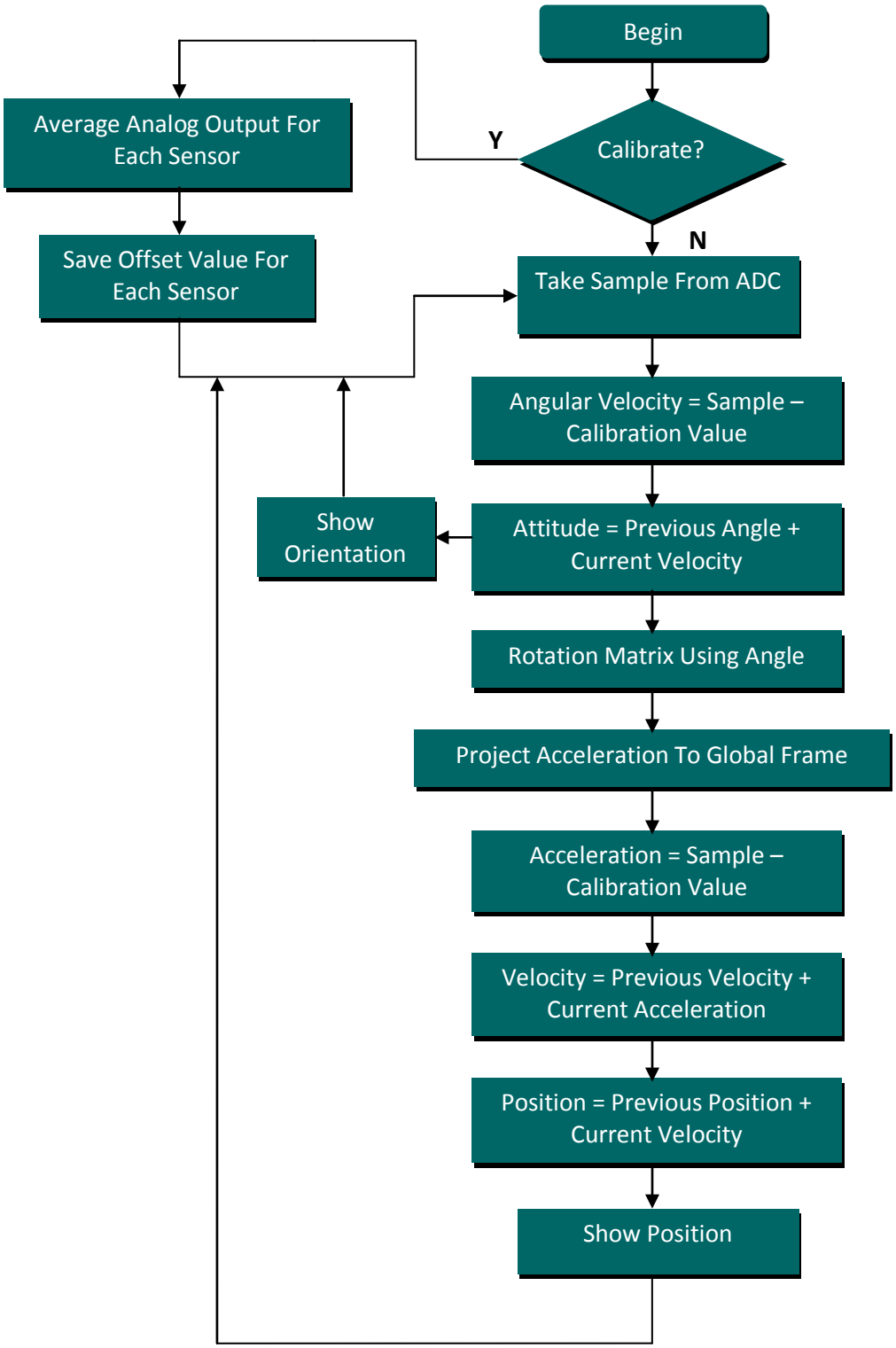

Figure 17: µC Flow Diagram

4.6 Communication with Computer and Virtual Simulation

A basic virtual simulation of a hand was created to display the functionality of the position and orientation tracking. The final outputs from the gyroscope and accelerometer are an angle of orientation and a displacement in the x, y and z direction. The Arduino Duemilanove is capable of communicating with a computer through the ATmega328's UART TTL (5V) serial communication. A Future Technology Devices International (FTDI) chip (FT232RL) on the board channels this serial communication over the USB interface. FTDI drivers were installed to provide a virtual com port on the computer. The processed data from the sensors is sent from the µC via a serial print command at 9600 bps to the computer on the virtual com port.

Processing is an open source programming language and environment for programming images, animation, and interactions.[34] Processing is capable of reading from the virtual com port with the use of the Processing Serial Library. A program was written to take values from the serial port; this now allows for the opportunity to program a virtual simulation based on the angle (rad) and x,y, z position (cm) outputs from the µC. To model a basic hand with two fingers, the Angle (rad) and X,Y position (cm) were used.

4.7 Code Explanation

The following sections give an analysis behind the design and implementation for the computer simulation of a hand. Source code is provided in Appendix A.4. This work combines both the processed data from this part of the project as well as the data on finger and thumb flexion from Arefin's work.

4.7.1 Serial Communication

In the µC programming (Arduino Code), each sensors output is sent via a serial print command at 9600 bps to the computer using the virtual com port. The values are comma-delimited and each sample set ends with a newline.

The virtual simulation program (Processing Code) reads ASCII values from the serial port at 9600 bps. The values are stored into variables which are later used to model the virtual hand.

Arduino Code:

```
//Initialize the serial port
Serial.begin(9600);

// Output values to serial port as an ASCII numeric string
Serial.print(angle, DEC);
Serial.print(",");
Serial.print(x_pos, DEC);
Serial.print(",");
Serial.print(y_pos, DEC);
Serial.print(",");
Serial.print(z_pos, DEC);
Serial.print(",");
Serial.print(analogRead(4), DEC); //index
Serial.print(",");
Serial.print(analogRead(5), DEC); // thumb
// Print a newline and carriage return in the end
Serial.println();
```

Processing Code:

```
println(Serial.list()); //list all available serial ports
myPort = new Serial(this, Serial.list()[0], 9600);
myPort.clear();
myPort.bufferUntil('\n'); // only generate a serialEvent() when
a newline is detected (end of sample)

String inString = myPort.readStringUntil('\n'); //get the ASCII
string
```

```
inString = trim(inString); //trim off any whitespace
float incomingValues[] = float(split(inString, ",")); //convert
to an array of floats
```

4.7.2 Screen Setup

The Processing language Application Programming Interface (API) includes functions that allow for the creation of many sophisticated structures. The screen is initialized as an 800 x 600 with a black background and a ball is drawn in the top centre.

```
void setup () {
  //initialize screen size and background
  size(800,600);
  background(0); // set initial background
  smooth(); //turn on anti aliasing

  //draw circle (equal width and height of ellipse)
  stroke(255);
  fill(255);
  ellipse(400, 100, 45, 45);
}
```

4.7.3 Modelling the Hand

The right hand will be modeled in two dimensions as a simple three line structure joined to produce a “] ” shape. . The top and bottom lines will represent the index finger and thumb, respectively. This simulation uses the flex sensor output that was developed by Arefin to control the finger flexion of the hand. It has a pivot point at the joint of the bottom and centre lines; this point represents the wrist joint of a human hand.

The hand will be initialized in the relaxed state and when a user fingers their index finger and thumb while wearing the glove, the centre line of the hand should decrease such that the

index finger and thumb come together, to mimic the users' flexion on-screen. This centre line represents the 'finger distance' and has a maximum value of 100 (no flexion) and a minimum value of 0 (fully flexed). The ADC resolution increases when the flex sensors are flexed, therefore these values must be mapped such the finger distance begins at 100 and is reduced to 0 as the flex sensors are flexed.

```
index = abs(map(incomingValues[4], 0, 750, -100, 0));
thumb = abs(map(incomingValues[5], 250, 655, -100, 0));
fd = (index + thumb)/2; //use average of thumb and index finger
flexion for finger distances
```

Using the variables containing the relative position (x,y in cm) and orientation (attitude in degrees), the hand is drawn as three lines. The centre line's length is variable based on the level of finger flexion, while the top and bottom lines are of fixed length. The orientation will control the angle of rotation at the wrist joint (the intersection of the bottom and centre lines). All three line structures are to move in unison with changes of x and y position. In this way, the hands orientation changes based on the angle. The hands position changes based on the x and y displacement. The x and y positions have an offset so that the hand is located at the bottom centre of the screen at program launch.

```
int x_offset = 450, y_offset = 500; //offsets to determine start
position of hand
angle = incomingValues[0];
x = -incomingValues[1];
y = incomingValues[2];

//outline colour
stroke(255);
//vertical line
line(x + x_offset,y + y_offset,x + x_offset - fd*sin(angle),y +
y_offset - fd*cos(angle));
//bottom line (thumb)
line(x + x_offset,y + y_offset,x + x_offset - fd*cos(angle),y +
y_offset + fd*sin(angle));
 //top line (index finger)
```

```
line(x + x_offset - fd*sin(angle),y + y_offset - fd*cos(angle),x
+ x_offset - fd*sin(angle)- fd*cos(angle),y + y_offset -
fd*cos(angle)+ fd*sin(angle));
```

4.7.4 Real-time Movement

At each new sample of values (at each serial read), the values of the position x and y are stored such that they can be accessed at the next iteration. At a current sample, the previous x and y positions are used to draw a black rectangle over the area that the hand was at the last sample. In this way, when a new hand is drawn based on new position and orientation information, the old one is covered; the result is the visual of a moving hand.

```
//if there is a change in position/orientation
  if ((angle != angle_old) || (x != x_old) || (y != y_old) ||
(fd != fd_old)) {

  //create a black box around the old position to clear the old
hand
  stroke(0); //outline colour
  fill(0); //fill colour
  rect(x_old + x_offset-200,y_old + y_offset-200,400,400);

  //the black box shouldn't cover the circle so draw the circle
again
  stroke(255);
  fill(255);
  ellipse(400, 100, 45, 45);
}
```

5. Results & Discussion

The final setup of the design is shown in Figure 17. The IMU has been attached to the glove just above the wrist such that the orientation of the hand relative to the wrist joint can be tracked using the yaw rate output of the gyroscope.

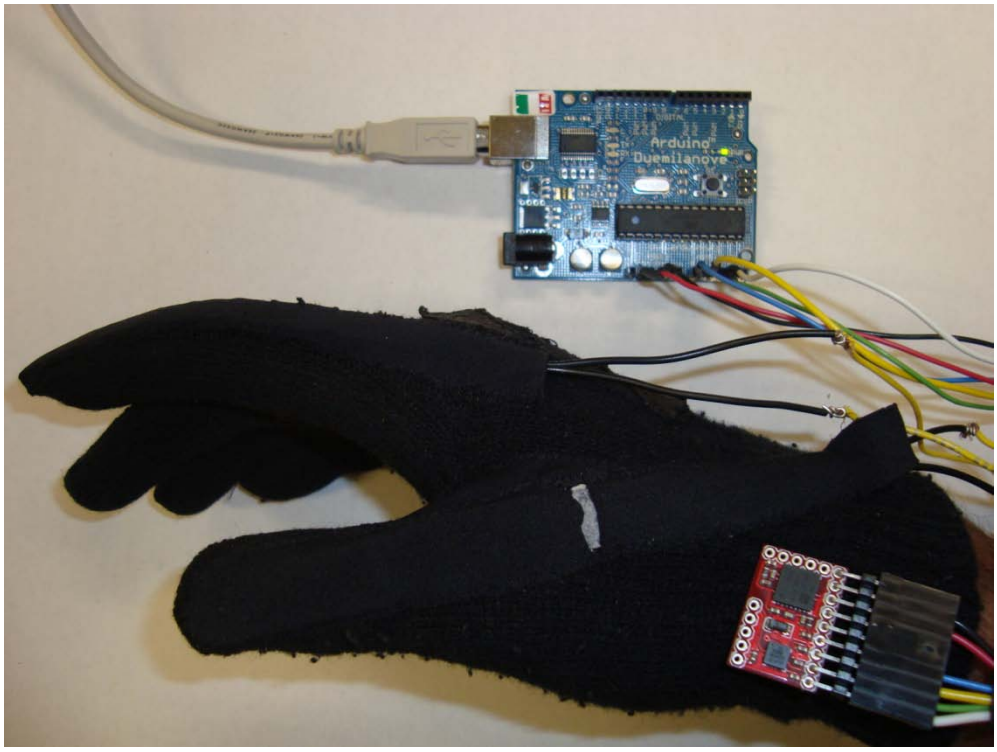

Figure 18: Final Design Setup

The microcontroller algorithm for tracking position and attitude relies on the manipulation of the acceleration and angular velocity outputs from the accelerometer and gyroscope. The gyroscope output is integrated and used to develop a rotation matrix to project the acceleration onto the global frame of reference. The acceleration outputs are then double integrated to achieve position.

Due to the fact that a single-axis gyroscope was used, a rotation matrix can only bring acceleration in the x-y plane into the global frame of reference. Hence, motion is restricted to two dimensions. This is the biggest limitation in the design, as it was originally expected to be able to track movement of the hand in three-dimensional space. Due to budgeting constraints, it was not feasible to purchase another 6DOF sensor board. However, the µC programming has been designed in such a way that additional sensors can be added on with minimal adjustments. The algorithm developed can be extended to higher DOF sensors with ease.

Due to the inability to bring develop a complete rotation matrix to project all axis of acceleration to the global frame, an unwanted component of gravity affects the horizontal acceleration when the sensor is tilted. In a strapdown algorithm, the calibration routine is run to determine a steady state value that is used to normalize the sensor data. This is effectively subtracting 1g from the (global) vertical acceleration signal (z-axis) to remove acceleration due to gravity before integration. A tilt error of α will cause a component of the acceleration due to gravity to be projected onto the horizontal axes with magnitude $g \cdot \sin(\alpha)$. This component causes residual bias in the acceleration signals. There will also be a component of magnitude $g \cdot (1-\cos(\alpha))$ on the (global) vertical axis however this is much less severe. For a small α, $\cos(\alpha) \rightarrow 1$ and $\sin(\alpha) \rightarrow \alpha$ therefore the error in position caused by a small tilt error is prominent on the global x-y plane.[29]

The position and attitude can be simulated in real-time using a serial communication protocol as discussed in section 4.7.1. The sensor outputs can be demonstrated in graphical form. A modified version of a graphing program[35] in Processing was used to test the sensor outputs from the µC. Figure 18 shows a real-time response to different movements.

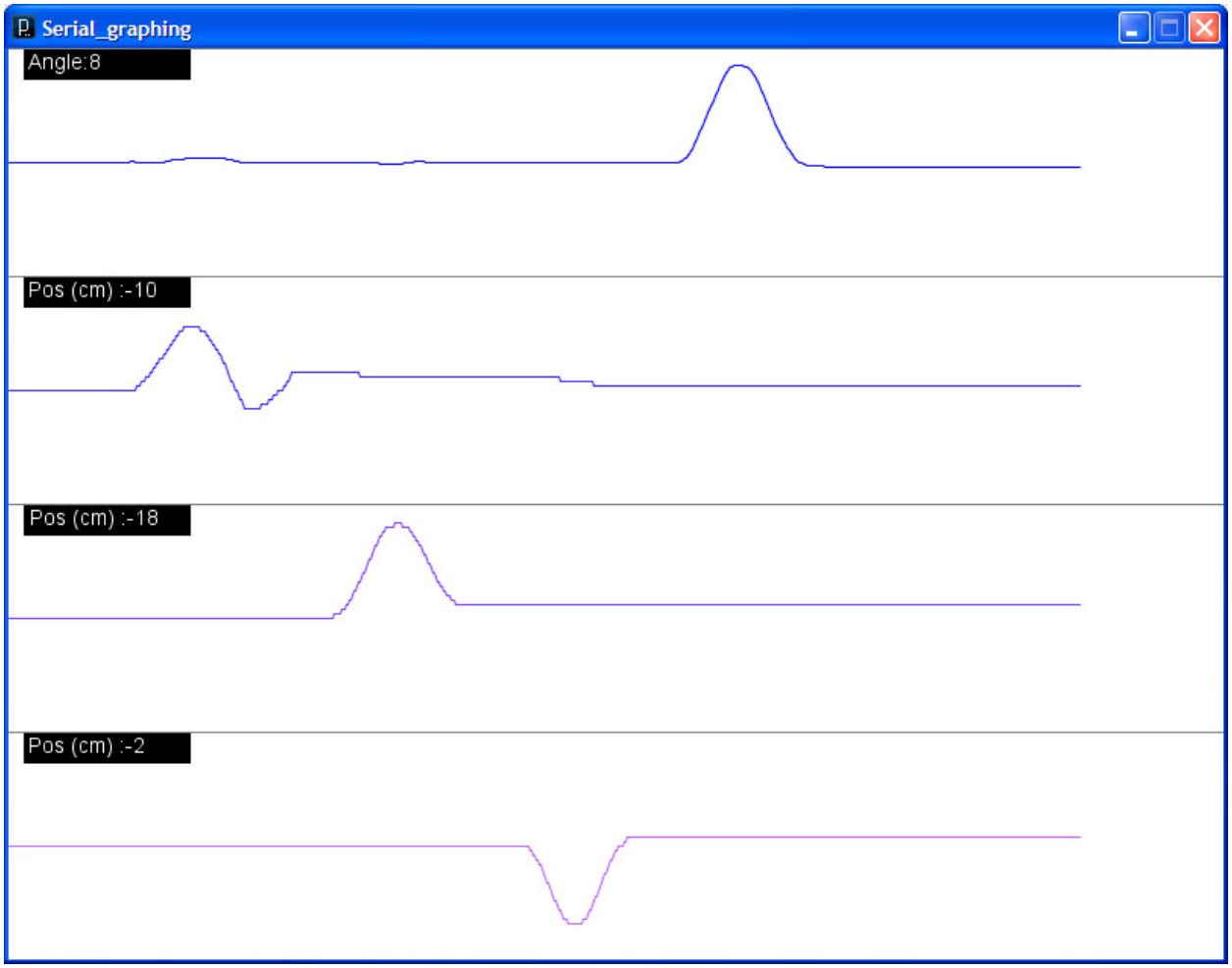

Figure 19: Real-time graphing of Sensor Outputs

This graph displays a series of movements by the sensor then returning the sensor to its initial state. The sensor was first moved in the positive x-axis direction then in the positive y-axis direction. This was followed by lifting the sensor up and bringing it back down on the z-axis. and finally a 90° clockwise rotation and return to initial position. The magnitudes of the values in the graph of Figure 18 are inverted and are proportional.

A simulation of a virtual hand that can reach for a ball (position tracking), turn about the wrist (attitude tracking) and grasp and release (finger flexion) has been developed using Processing as discussed in section 4.6 and 4.7. Figure 19 shows a screenshot of the simulation.

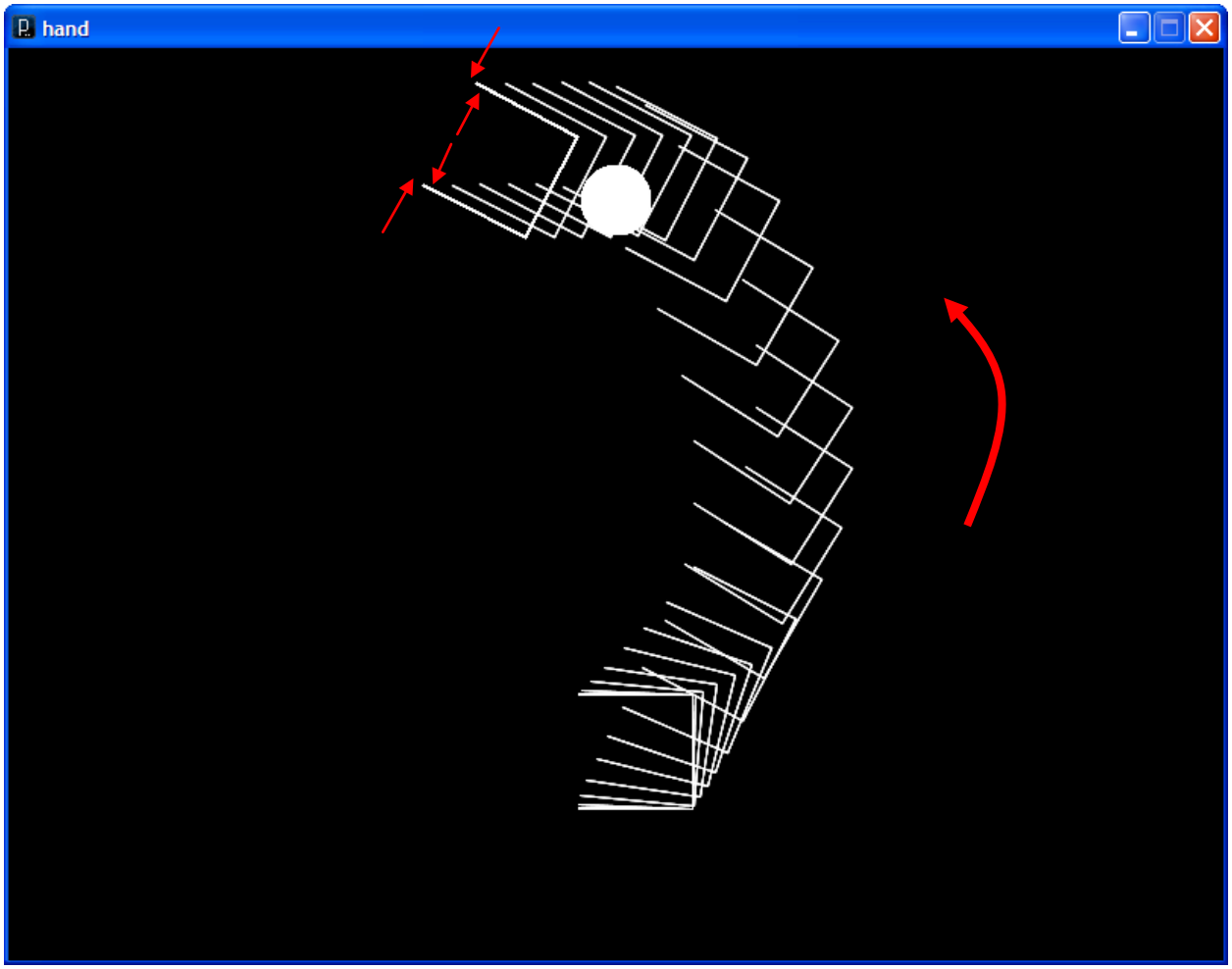

Figure 20: Virtual Hand Simulation

In this screenshot, the old position of the hand was not removed as discussed in section 4.7.4 in order to show the movement. The hand is able to change its attitude at the wrist while maintaining positional accuracy in the x-y global frame; thus the rotation matrix computation is shown to be accurate. Finger flexion is not demonstrated in this screenshot, however flexing the fingers brings the top and bottom lines closer together and reduces the height of the centre line; thus mimicking a grasp and release motion.

6. Conclusions & Recommendations

In this design project I was successfully able to complete the implementation of two tracking features of the hand: orientation and relative position. This included acquiring data from a wearable glove equipped with an accelerometer and a gyroscope, microcontroller programming to analyze the data, developing virtual serial communication interface with a computer, and developing a virtual simulation of an assistive device that responds to hand movements using the glove. There are three primary areas where this design can be improved.

6.1.1 Increase DOF

As stated in section 5, the biggest limitation in the design was that position tracking cannot be performed in three-dimensional space due to the missing attitude angles required in the rotation matrix computation. A triple-axes gyroscope would solve this problem. The μC programming needs only minimal changes to incorporate extra degrees of freedom from sensors. This would allow for a user to move their hand freely in space rather than restricting motion to a horizontal plane.

6.1.2 Reduce Integration Drift

A recognized issue that occurs with the use of accelerometers and gyroscopes is integration drift, whereby over time the integration of inertial sensor data leads to errors and increased position uncertainty. The accumulation of data from sensor noise, offsets, orientation and errors will cause a ‘drift’ from its true value and result in uncertainty. Software filtering

techniques (section 4.4.3 and 4.4.7) have been incorporated into the µC programming to reduce the error; however more sophisticated software filtering techniques exist. One widely used method is Kalman filtering.[36] The basic idea behind Kalman filtering is using an iterative approach to observe values over time that contain inaccuracies and produce values that tend to be closer to the true values of the measurements. It requires some kind of input that can be turned into a prediction of the desired output and a second input that is a good approximation of the real world value of the predicted one.

6.1.3 Enhance Virtual Modeling

The program of choice for the virtual simulation was Processing due to its free distribution and similarity with the Arduino µC programming language. If the DOF of the sensor board was increased to six and the sensor could track motion in three dimensions, more sophisticated virtual models can be developed for different purposes. Advanced programming in Processing can allow for incorporation with other modelling software's such as 3D Autodesk MAYA. This would allow for advanced simulation capabilities such as a virtual surgery simulation.

A.1 Gyroscopes

A gyroscope measures orientation based on the principles of angular momentum. In this design project a single-axis gyroscope was used as a yaw rate (z-axis) sensor. Several types of gyroscopes exist as explained in (29). The comparisons are summarized below.

Mechanical

A conventional gyroscope consists of a spinning wheel mounted on two gimbals which allow it to rotate in all three axes. An effect of the conservation of angular momentum is that the spinning wheel will resist changes in orientation. When a mechanical gyroscope is subjected to a rotation the wheel will remain at a constant global orientation and the angles between adjacent gimbals will change. The main disadvantage of mechanical gyroscopes is that they contain moving parts, which cause friction, which in turn causes the output to drift over time.

Optical

A fiber optic gyroscope (FOG) uses the interference of light to measure angular velocity. A FOG consists of a large coil of optical fiber. To measure rotation two light beams are fired into the coil in opposite directions. If the sensor is undergoing a rotation then the beam travelling in the direction of rotation will experience a longer path to the other end of the fiber than the beam travelling against the rotation. These are beneficial because unlike mechanical gyroscopes, optical gyros contain no moving parts and require only a few seconds to start-up.

MEMS (Micro-Machined Electromechanical Systems)

MEMS sensors built using silicon micro-machining techniques have low part counts and are relatively cheap to manufacture. MEMS gyroscopes make use of the Coriolis effect by using vibrating elements. Many vibrating element geometries exist, such as vibrating wheel and tuning fork gyroscopes. The simplest geometry consists of a single mass which is driven to vibrate along a drive axis. When the gyroscope is rotated a secondary vibration is induced along the perpendicular sense axis due to the Coriolis force. The angular velocity can be calculated by measuring this secondary rotation.

A.2 Accelerometers

An accelerometer measures the acceleration experienced relative to freefall. In this design project a triple-axis accelerometer was used as a x,y,z-axes acceleration sensor. Several types of linear accelerometers exist as explained in (29). The comparisons are summarized below.

Mechanical

A mechanical accelerometer consists of a mass suspended by springs. The displacement of the mass is measured using a displacement pick-off, giving a signal that is proportional to the force acting on the mass in the direction of the input axis. Newton's second law (F=ma) is then used to calculate the acceleration acting on the device.

Solid State

Solid-state accelerometers can be sub-categorized to surface acoustic wave, vibratory, silicon and quartz devices. An example of a solid-state accelerometer is the surface acoustic wave (SAW) accelerometer. A SAW accelerometer consists of a cantilever beam which is resonated at a particular frequency. A mass is attached to one end of the beam which is free to move, while the other end is rigidly attached to the case. An acceleration applied along the input axis causes the beam to bend. This results in the frequency of the surface acoustic wave to change proportionally to the applied strain. Acceleration can be measured by measuring this change in frequency.

MEMS (Micro-Machined Electromechanical Systems)

Micro-machined silicon accelerometers use the same principles as mechanical and solid state sensors. There are two main classes of MEMS accelerometer: the first class consists of mechanical accelerometers (i.e. devices which measure the displacement of a supported mass) and the second class consists of devices which measure the change in frequency of a vibrating element caused by a change of tension, as in SAW accelerometers.

A.3 Source Code: Microcontroller Algorithm

```
/*

  Gyroscope and Accelerometer Integration
  By: Thilakshan Kanesalingam

  April 4, 2010
  4BI6 Biomedical Design, McMaster University

  Description:

  Takes raw ADC counts from the analog inputs of the Arduino from
  a single axis gyroscope and triple axis accelerometer and prints
  the angle of rotation (in radians) and x,y,z position (in cm) to
  the serial port at 9600 bps. Values are comma-delimited and each
  sample set ends with a newline.

  The gyroscope data provides angular velocity in the x axis. The
  value from each sample is normalized and integrated (trapezoidal
  method) to give the angle of rotation (orientation).

  The angle of rotation is used to generate a rotation matrix about
  the z-axis. This rotation matrix is applied to the raw ADC counts
  from the triple axis accelerometer to project acceleration in the
  x,y,z direction from the base frame of reference to the global
  frame of reference.

  The acceleration (x,y,z) in the global frame of reference is
  normalized and double integrated (trapezoidal method) to obtain
  position in the x,y,z direction.

*/

//initialize functions
void Initialize_Globals();
void Calibrate(void);
double make_radians_per_sec(double ADC_angular_rate);
double make_metres_per_sec2(double ADC_acceleration);
void Gyroscope(double x_gyro);
void UpdateRotationMatrix(double angle[3]);
void Accelerometer(double x_accel, double y_accel, double z_accel);
void Accel_Movement_End_Check(void);

//initialize global variables
double old_R[3][3] = {{1,0,0},{0,1,0},{0,0,1}};
double current_R[3][3] = {{0}};

double x_angular_rate[2];
double x_angle[2];

double x_acceleration[2], y_acceleration[2], z_acceleration[2];
double x_velocity[2], y_velocity[2], z_velocity[2];
double x_position[2], y_position[2], z_position[2];
```

```
double ss_gyro_x, ss_accel_x, ss_accel_y, ss_accel_z;
int countx_gyro, countx_accel, county_accel, countz_accel;

double start_time = 0;
double dt = 0; //time step between samples

float angle, x_pos, y_pos, z_pos;

void setup() {
  Initialize_Globals(); //initialize global variables
  Calibrate(); //calibrate
  Serial.begin(9600); //initialize the serial port
}

void loop() {
   //start time is the number of ms since the Arduino board began running the
current program
  start_time = millis();

  Gyroscope(analogRead(0));
  Accelerometer(analogRead(1), analogRead(2), analogRead(3));

  //output values to serial port as an ASCII numeric string
  Serial.print(angle, DEC);
  Serial.print(",");
  Serial.print(x_pos, DEC);
  Serial.print(",");
  Serial.print(y_pos, DEC);
  Serial.print(",");
  Serial.print(z_pos, DEC);
  Serial.print(",");
  Serial.print(analogRead(4), DEC); //flex sensor reading: index finger
  Serial.print(",");
  Serial.print(analogRead(5), DEC); //flex sensor reading: thumb
  Serial.println(); //print a newline and carriage return in the end

  //delay 1ms before next read
  delay(1);

  //define time step as the difference between current time and start time
(since program launch) in seconds
  dt = (millis() - start_time)/1000;
}

void Gyroscope(double x_gyro) {

  double w_d = 0.2; //window of discrimination for no-movement condition: 0.2
rad/s

  //subtract the zero-rate level to obtain positive and negative angular rate
  //convert from proportional ADC counts to radians per second
  x_angular_rate[1] = make_radians_per_sec(x_gyro) -
make_radians_per_sec(ss_gyro_x);

  //apply discrimination window
  if ((x_angular_rate[1] <= w_d) && (x_angular_rate[1] >= -w_d))
{x_angular_rate[1] = 0;}
```

```
  //integrate using first order approximation (trapezoidal method)
  // = rectangle area + triangle area
  // = (b-a)*f(a) + 0.5*(b-a)[f(b)-f(a)]
  x_angle[1] = x_angle[0] + (x_angular_rate[0] + ((x_angular_rate[1] -
x_angular_rate[0])/2))*dt;

  //current velocity sent to previous velocity
  x_angular_rate[0] = x_angular_rate[1];

  double angle_vector[3] = {x_angle[1]-x_angle[0],0,0}; //angle vector for
integrated angular velocity
  //if the angle is non-zero (with window) create a rotation matrix
  if (angle_vector[0] != 0) {UpdateRotationMatrix(angle_vector);}

  //current angle sent to previous angle
  x_angle[0] = x_angle[1];

  //final angle as float variable for serial output as radians
  //multiply by (180/3.14) for angle in degrees
  angle = x_angle[1];
}

void UpdateRotationMatrix(double angle[3]){ // Update R as each new sample
becomes available

  double sum; //summation variable used in matrix multiplication
  int i, j, k; //counters
  //attitude update matrix is an elementary rotation matrix about z
  double attitude_update[3][3] = {{cos(angle[0]),-sin(angle[0]),0},
                                  {sin(angle[0]),cos(angle[0]),0},
                                  {0,0,1}};

  //Compute: Current Rotation Matrix = Old Rotation Matrix * Attitude Update
Matrix
  //(multiplication of two 3x3 matrices)
  for (i=0; i<3; i++) {
    for (j=0; j<3; j++) {
      sum=0;
      for (k=0; k<3; k++) {
        sum = sum + old_R[i][k]*attitude_update[k][j];
        current_R[i][j] = sum;
      }
    }
  }

  //Send Current Rotation Matrix to Previous Rotation Matrix: old_R =
current_R
  for (i=0; i<3; i++) {
    for (j=0; j<3; j++) {
      old_R[i][j] = current_R[i][j];
    }
  }
}

void Accelerometer(double x_accel, double y_accel, double z_accel) {
```

```
  double w_d = 0.5; //window of discrimination for no-movement condition: 50
cm/s^2
  double wd_max = 2; //window of discrimination for gravity condition: 2
m/s^2
  double base_accel[3] = {0}; //base frame of reference
  double global_accel[3] = {0}; //global frame of reference
  int i=0, j=0; //counters

  //subtract the zero-rate level to obtain positive and negative acceleration
  //convert from proportional ADC counts to metres per second squared
  x_acceleration[1] = make_metres_per_sec2(x_accel) -
make_metres_per_sec2(ss_accel_x);
  y_acceleration[1] = make_metres_per_sec2(y_accel) -
make_metres_per_sec2(ss_accel_y);
  z_acceleration[1] = make_metres_per_sec2(z_accel) -
make_metres_per_sec2(ss_accel_z);

  //apply discrimination window for no-movement condition
  if ((x_acceleration[1] <= w_d) && (x_acceleration[1] >= -w_d))
{x_acceleration[1] = 0;}
  if ((y_acceleration[1] <= w_d) && (y_acceleration[1] >= -w_d))
{y_acceleration[1] = 0;}
  if ((z_acceleration[1] <= w_d) && (z_acceleration[1] >= -w_d))
{z_acceleration[1] = 0;}

  //if acceleration is very fast, its likely due to an unwanted gravity
component, so ignore it
  if ((x_acceleration[1] >= wd_max) || (x_acceleration[1] <= -wd_max))
{x_acceleration[1] = 0;}
  if ((y_acceleration[1] >= wd_max) || (y_acceleration[1] <= -wd_max))
{y_acceleration[1] = 0;}
  if ((z_acceleration[1] >= wd_max) || (z_acceleration[1] <= -wd_max))
{z_acceleration[1] = 0;}

  //move acceleration signal in the base frame of reference to a new vector
  base_accel[0] = x_acceleration[1];
  base_accel[1] = y_acceleration[1];
  base_accel[2] = z_acceleration[1];

  //project acceleration into the global frame of reference: matrix and
vector multiplication
  for (i=0; i<3; i++) {
    for (j=0; j<3; j++) {
      global_accel[i] = global_accel[i] + current_R[i][j]*base_accel[j];
    }
  }

  //move projected acceleration back
  x_acceleration[1] = global_accel[0];
  y_acceleration[1] = global_accel[1];
  z_acceleration[1] = global_accel[2];

  //integrate using first order approximation (trapezoidal method)
  // = rectangle area + triangle area
  // = (b-a)*f(a) + 0.5*(b-a)[f(b)-f(a)]
  //double integrate each axis of acceleration to get position
```

```
  //first x integration
  x_velocity[1] = x_velocity[0] + (x_acceleration[0] + ((x_acceleration[1] -
x_acceleration[0])/2.0))*dt;
  //second x integration
  x_position[1] = x_position[0] + (x_velocity[0] + ((x_velocity[1] -
x_velocity[0])/2))*dt;

  //same for y
  y_velocity[1] = y_velocity[0] + (y_acceleration[0] + ((y_acceleration[1] -
y_acceleration[0])/2.0))*dt;
  y_position[1] = y_position[0] + (y_velocity[0] + ((y_velocity[1] -
y_velocity[0])/2))*dt;

  //same for z
  z_velocity[1] = z_velocity[0] + (z_acceleration[0] + ((z_acceleration[1] -
z_acceleration[0])/2.0))*dt;
  z_position[1] = z_position[0] + (z_velocity[0] + ((z_velocity[1] -
z_velocity[0])/2))*dt;

  //current accel sent to previous accel
  x_acceleration[0] = x_acceleration[1];
  y_acceleration[0] = y_acceleration[1];
  z_acceleration[0] = z_acceleration[1];

  //same for velocity
  x_velocity[0] = x_velocity[1];
  y_velocity[0] = y_velocity[1];
  z_velocity[0] = z_velocity[1];

  //check for end of movement
  Accel_Movement_End_Check();

  //actual position sent back to previous position
  x_position[0] = x_position[1];
  y_position[0] = y_position[1];
  z_position[0] = z_position[1];

  //final position as float variable for serial output in cm
  x_pos = x_position[1]*100;
  y_pos = y_position[1]*100;
  z_pos = z_position[1]*100;
}

//initial values for global variables
void Initialize_Globals(){

  x_angular_rate[0] = 0;
  x_angle[0] = 0;

  x_acceleration[0] = 0;
  y_acceleration[0] = 0;
  z_acceleration[0] = 0;
  x_velocity[0] = 0;
  y_velocity[0] = 0;
  z_velocity[0] = 0;
  x_position[0] = 0;
  y_position[0] = 0;
```

```
  z_position[0] = 0;

  angle = 0;
  x_pos = 0;
  y_pos = 0;
  z_pos = 0;

  ss_gyro_x = 0;
  ss_accel_x = 0;
  ss_accel_y = 0;
  ss_accel_z = 0;

  countx_gyro = 0;
  countx_accel = 0;
  county_accel = 0;
  countz_accel = 0;
}

void Calibrate(void) {

  unsigned int count1;
  count1 = 0;

  do{ //accumulate samples
    ss_gyro_x = ss_gyro_x + analogRead(0);
    ss_accel_x = ss_accel_x + analogRead(1);
    ss_accel_y = ss_accel_y + analogRead(2);
    ss_accel_z = ss_accel_z + analogRead(3);
    count1++;
  }while(count1!=500); //500 times

  //average the samples
  ss_gyro_x = ss_gyro_x/count1;
  ss_accel_x = ss_accel_x/count1;
  ss_accel_y = ss_accel_y/count1;
  ss_accel_z = ss_accel_z/count1;
}

double make_radians_per_sec(double ADC_angular_rate){

  double Vref = 5; //arduino ADC 5V ref voltage
  double sensitivity = 0.0033; // 3.3mV/(degrees/sec) sensitivity (from gyro
data sheet)

  //convert ADC value to voltage and divide by sensitivity
  return ((ADC_angular_rate*Vref/1024)/sensitivity)*(3.14/180);
}

double make_metres_per_sec2(double ADC_acceleration){

  double Vref = 5; //arduino ADC 5V ref voltage
  double sensitivity = 0.33; // 0.33V/g sensitivity ratiometric when Vs =
3.3V (from accel data sheet)

  //convert ADC value to voltage and divide by sensitivity
  //convert standard gravity g to m/s^2 by multiplying by 9.80665
  return ((ADC_acceleration*Vref/1024)/sensitivity)*9.80665;
```

```
}

void Accel_Movement_End_Check(void){

  //count the number of accel samples that equal zero
  if (x_acceleration[1]==0) {countx_accel++;}
  else {countx_accel = 0;}
  //if this number exceeds 5, we can assume that velocity is zero
  if (countx_accel >= 5) {
    x_velocity[1]=0;
    x_velocity[0]=0;
  }

  //same for y
  if (y_acceleration[1]==0) {county_accel++;}
  else {county_accel = 0;}
  if (county_accel >= 5) {
    y_velocity[1]=0;
    y_velocity[0]=0;
  }

  //same for z
  if (z_acceleration[1]==0) {countz_accel++;}
  else {countz_accel = 0;}
  if (countz_accel >= 5) {
    z_velocity[1]=0;
    z_velocity[0]=0;
  }
}
```

A.4 Source Code: Virtual Simulation

```
/*

  Hand Simulation
  By: Thilakshan Kanesalingam

  April 4, 2010
  4BI6 Biomedical Design, McMaster University

  Description:

  Takes ASCII values from the serial port at 9600 bps. Values should
  be comma-delimited and each sample set should end with a newline.
  Angle (rad) and X,Y position (cm) are used to draw three lines
  representing a hand. The hands orientation changes based on the
  angle. The hands position changes based on the X,Y displacement.

*/

import processing.serial.*;
Serial myPort;
int maxSensors = 6; //the Arduino has a maximum of 6 possible inputs
float angle_old=0, x_old=0, y_old=0, fd_old=0; //previous values used to
cover old drawing

void setup () {
  println(Serial.list()); //list all available serial ports
  myPort = new Serial(this, Serial.list()[0], 9600);
  myPort.clear();
  myPort.bufferUntil('\n'); // only generate a serialEvent() when a newline
is detected (end of sample)
  size(800,600);
  //size(1000, 676); // office
  //size(640, 376); //food

  //PImage b;
  //b = loadImage("office.jpg");
  //b = loadImage("food.jpg");
  //background(b);
  background(0); // set initial background
  smooth(); //turn on anti aliasing
}

void draw () {
  //keeps the program running
}

void serialEvent (Serial myPort) {
  String inString = myPort.readStringUntil('\n'); //get the ASCII string

  if (inString != null) { //if it's not empty
    inString = trim(inString); //trim off any whitespace
```

```
    float incomingValues[] = float(split(inString, ",")); //convert to an
array of floats
    float angle=0, x=0, y=0; //angle, position (x,y)
    float thumb=0, index=0, fd=0; //thumb and index flexion (proportional
values), fd is finger distance used to draw hand
    int x_offset = 450, y_offset = 500; //offsets to determine start position
of hand

    //draw circle (equal width and height of ellipse)
    stroke(255);
    fill(255);
    ellipse(400, 100, 45, 45);

    if (incomingValues.length <= maxSensors && incomingValues.length > 0) {
      //loop through each array element (each sensor output)
      for (int i = 0; i < incomingValues.length; i++) {

        angle = incomingValues[0];
        x = -incomingValues[1]*25; //amplify 25x
        y = incomingValues[2]*25; //amplify 25x

        //re-map thumb and index finger flexion to values between 0 and 100
        //100 corresponds to no flexion and 0 to complete flexion
        //the ADC count from the fingers increase when flexed; the reverse is
required
        index = abs(map(incomingValues[4], 0, 750, -100, 0));
        thumb = abs(map(incomingValues[5], 250, 655, -100, 0));
        fd = (index + thumb)/2; //use average of thumb and index finger
flexion for finger distances

        //outline colour
        stroke(255);
        //vertical line
        line(x + x_offset,y + y_offset,x + x_offset - fd*sin(angle),y +
y_offset - fd*cos(angle));
        //bottom line (thumb)
        line(x + x_offset,y + y_offset,x + x_offset - fd*cos(angle),y +
y_offset + fd*sin(angle));
        //top line (index finger)
        line(x + x_offset - fd*sin(angle),y + y_offset - fd*cos(angle),x +
x_offset - fd*sin(angle)- fd*cos(angle),y + y_offset - fd*cos(angle)+
fd*sin(angle));

        //if there is a change in position/orientation
        if ((angle != angle_old) || (x != x_old) || (y != y_old) || (fd !=
fd_old)) {

          //create a black box around the old position to clear the old hand
          stroke(0); //outline colour
          fill(0); //fill colour
          rect(x_old + x_offset-200,y_old + y_offset-200,400,400);

          //the black box shouldn't cover the circle so draw the circle again
          stroke(255);
          fill(255);
          ellipse(400, 100, 45, 45);
```

```
            //set old values to current values
            angle_old = angle;
            x_old = x;
            y_old = y;
            fd_old = fd;

          }
        }
      }
    }
  }
```

References

1. Ji-Hwan Kim, Nguyen Duc Thang, Tae-Seong Kim, “3-D hand motion tracking and gesture recognition using a data glove”. Industrial Electronics, 2009. ISIE 2009. IEEE International Symposium on 5-8 July 2009 Page(s):1013 – 1018

2. R. Ozawa, N. Ueda, “Supervisory control of a multi-fingered robotic hand system with data glove”. Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on
Oct. 29 2007-Nov. 2 2007 Page(s):1606 – 1611

3. Chou Wusheng, Wang Tianmiao, Hu Lei, “Design of data glove and arm type haptic interface”. Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2003. HAPTICS 2003. Proceedings. 11th Symposium on 22-23 March 2003 Page(s):422 – 427

4. Mostafa, D.G.M, “Anthropomorphic interface for robot arm programming through a data glove”.
Industrial Electronics, 1994. Symposium Proceedings, ISIE '94., 1994 IEEE International Symposium on 25-27 May 1994 Page(s):326 – 328

5. JunJie Zhang, Jiangcheng Fang, “Research of Mechanical Arm Control Based on Data Glove”.
Information Science and Engieering, 2008. ISISE '08. International Symposium on Volume 1, 20-22 Dec. 2008 Page(s):188 – 191

6. Xiaoling Lv, Minglu Zhang, Feng Cui, Xiaoli Zhang, “Teleoperation of Robot Based on Virtual Reality”. Artificial Reality and Telexistence--Workshops, 2006. ICAT '06. 16th International Conference on Nov. 29 2006-Dec. 1 2006 Page(s):400 – 403

7. M. Romero Huertas, J. Raymundo Marcial Romero, H.A. Montes Venegas, “A robotic arm telemanipulated through a digital glove.” Electronics, Robotics and Automotive Mechanics Conference, 2007. CERMA 2007 25-28 Sept. 2007 Page(s):470 - 475

8. H. Zhou, H. Hu, N. Harris, “Wearable inertial sensors for arm motion tracking in home-based rehabilitation”. IOS press 2005.

9. H. Zhou, H. Hu, “Upper limb motion estimation from inertial measurements”. International Journal of Information Technology, 2007. 13(1).

10. O. Takesi, M. Honda, “Trajectory Formation in Sequential Arm Movements”. IEEE International Conference on Systems Man and Cybernetics, 1992.

11. M. Boonstra, R. Slikke, N. Keijsers, R. Lummel, M. Malefijt, N. Verdonschot, “The accuracy of measuring the kinematics of rising from a chair with acceelerometers and gyroscopes”. Journal of Biomechanics, 2006. 39: 354-358.

12. H. Luinge, P. Veltink, “Inclination Measurement of Human Movement Using a 3-D Accelerometer With Autocalibration”. IEE Transactions on Neural Systems and Rehabilitation Engineering, 2004. 12(1): 1534-4320.

13. J. Kim, N. Thang, H. Suh, T. Rasheed, T. Kim, “Forearm Motion Tracking with Estimating Joint Angles from Inertial Sensor Signals”.

14. Benbasat, J. Paradiso, “An Inertial Measurement Framework for Gesture Recognition and Applications”.

15. S. Won, N. Parnian, F. Golnaraghi, W. Malek, “A Quaternion-Based Tilt Angle Correction Method for a Hand-Held Device Using an Inertial Measurement Unit”. 2008.

16. J. Yang, E. Choi, W. Chang, W. Bang, S. Cho, J. Oh, J. Cho, D. Kim, “A Novel Hand Gesture Input Device Based on Inertial Sensing Technique”. Conference of the IEEE Industrial Electronics Society, 2004.

17. W. Ang, P. Khosla, C. Riviere, “Design of All-Accelerometer Inertial Measurement Unit for Tremor Sensing in Hand-held Microsurgical Instrument”.

18. H. Zhou, H. Hu, “Kinematic model aided inertial motion tracking of human upper limb”. International Conference on Information Acquisition, 2005.

19. H. Zheng, N.D. Black, N.D. Harris, “Position-sensing technologies for movement analysis in stroke rehabilitation”. Med. Biol. Eng. Comput., 2005. 43: 413-420.

20. H. Zhou, H. Hu, “Human motion tracking for rehabilitation – A survey”. Biomedical Signal Processing and Control, 2008. 3: 1-18.

21. Y. Uno, M. Kawato, R. Suzuki, “Formation and Control of Optimal Trajectory in Human Multijoint Arm Movement”. Biol. Cybern., 1989. 61:89-101.

22. G. Lu, L. Shark, G. Hall, U. Zeshan, “Dynamic Hand Gesture Tracking and Recognition for Real-time Immersive Virtual Object Manipulation”. International Conference on CyberWorlds, 2009.

23. S. Guo, G. Song, Z. Song, “Development of a Self-assisted Rehabilitation System for the Upper Limbs Based on Virtual Reality”. IEEE International Conference on Mechatronics and Automation, 2007.

24. F. Cutolo, C. Mancinelli, S. Patel, N. Carbonaro, M. Schmid, A. Tognetii, D. De Rossi, P. Bonato, “A Sensorized Glove for Hand Rehabilitation”.

25. H. Zhou, H. Hu, “Inertial motion tracking of human arm movements in stroke rehabilitation”. IEEE International Conference on Mechatronics and Automation, 2005.

26. A.S. Ali, M.G. Madariaga, D.C. McGeary, W.R. Pruehsner, J.D. Enderle, “The assistive robotic arm.” Bioengineering Conference, 2007. NEBC '07. IEEE 33rd Annual Northeast 10-11 March 2007 Page(s):291 - 292.

27. F. Cutolol, C. Mancinelli, S. Patel, N. Carbonaro, M. Schmid, A. Tognettil, D. De Rossil, P. Bonato, "A Sensorized Glove for Hand Rehabilitation".

28. Woodman, Oliver J. *Intro to Inertial Navigation*

29. Kurt Seifert, Oscar Camacho. *Implementing Positioning Algorithms Using Accelerometers.* 2007. AN3397

30. Pycke, Tom. [Online] [Cited: April 1, 2010.] http://tom.pycke.be/mav/70/gyroscope-to-roll-pitch-and-yaw

31. Numerical Integration. [Online] [Cited: April 1, 2010.] http://www.ugrad.math.ubc.ca/coursedoc/math101/notes/techniques/numerical.html

32. Arduino Duemilanove. [Online] [Cited: April 9, 2010.] http://www.arduino.cc/en/Main/ArduinoBoardDuemilanove.

33. Processing. [Online] [Cited: April 1, 2010.] http://processing.org/

34. Igoe Interview. [Online] [Cited: April 1, 2010.] http://processing.org/exhibition/features/igoe/

35. Pycke, Tom. MAV Blog. [Online] April 1, 2010. http://tom.pycke.be/mav/71/kalman-filtering-of-imu-data.

Vitae

Name: Thilakshan Kanesalingam

Education: McMaster University (2005-2010)

Degree: Electrical and Biomedical Engineering Co-op

Internship: Princess Margaret Hospital (Dept. of Radiation Physics)