DOI 10.17148/IJARCCE.2020.91202

Gesture Language Glove

Harrison Keats[1], Benjamin Lindquist[2], Dean Aslam[3]

Electrical and Computer Engineering Department, Michigan State University, MI 48824, USA [1,2,3]

Abstract: Gesture language recognition is a technology that allows humans and animals to interact with computers and hardware without the need for any mechanical actuation between them. Gestures can include anything from facial expressions, head or eye movements, hand and arm motion, or other bodily movements and can therefore encompass a wide range of control for complicated systems. As sensor technology becomes smaller, cheaper and more readily available, gesture language recognition becomes a more viable solution for projects and systems to employ. This paper presents a prototype gesture language recognition system composed of hobby-grade electronics, compares it against systems using higher-grade hardware or more advanced software algorithms and extrapolates what further technological advancement will do for gesture language recognition systems.

Keywords: Gesture Language, MicroElectroMechanical Systems, Inertial Measurement Unit

I. INTRODUCTION

Gestures can include anything from facial expressions, head or eye movements, hand and arm motion, or other bodily movements and can therefore encompass a wide range of control for complicated systems. Recent research shows that when humans and animals learn or communicate by watching a gesture, known as gesture language, neurons called mirror neurons become active in the brain of the watcher [1]. In living creatures, these mirror neurons help to establish the system of communication or language, but for a computerized system the needs and applications are different. One such application of gesture language recognition is the control of a video game using a glove controller. In 1989, Nintendo launched its famed but flopped "Power Glove" which allowed users to control their characters in video games using a gesture recognition system [2]. Although this commercially available product had failed, this motivated others to dive deep into the world of gesture language recognition and advance this form of technology into what we have today.

Gesture language recognition can be implemented either by direct sensing or indirect sensing. Direct sensing in this case refers to sensors which are worn by or interacted with by the operator such as accelerometers, gyroscopes, flex sensors or pressure sensors. Indirect sensing refers to image processing or range-finding sensors that observe a subject without physically contacting them at all. Many of these systems include the use of MEMS or Micro-electromechanical Systems. Micro-electromechanical systems are a technology used to create tiny integrated devices or systems that combine mechanical and electrical components. They are fabricated using integrated circuit (IC) batch processing techniques and can range in size from a few micrometers to millimeters. These devices (or systems) can sense, control and actuate on the micro scale and generate effects on the macro scale [3].

The purpose of this research is to expand on this idea of using a gesture language recognition system, in the form of a glove, to control the cursor of a mouse on a computer, adopting current affordable MEMS technology. This goal is achieved by monitoring change in analog signals produced by the bending of a flex sensor to simulate the click of a mouse, as well as the acceleration and gyroscopic values transmitted from a 9DOF IMU to control the movement of the mouse cursor. By using a flex sensor, we can set a threshold range and map this to a binary input to detect the physical movement of bending a finger and then produce the action of clicking a mouse in a virtual environment. From the 9DOF IMU, we can calculate acceleration in the x, y, and z dimensions to detect direction of the movement based on the initialized state of the IMU. Both sensors are connected to a Teensy 3.2 microcontroller equipped with a Bluetooth module, to allow remote programming of the microcontroller through an Android phone. Since the boom of gesture language recognition technology research, many other companies and researchers have developed similar systems for different applications. One such application is gesture to speech recognition, in which sign language is interpreted through the hand movements of someone wearing a smart glove [4]. Other applications include controlling a virtual environment such as in a 3D modeling software [5] or controlling the movements of a robotic arm using computer vision [6].

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 9, Issue 12, December 2020

DOI 10.17148/IJARCCE.2020.91202

II. PROTOTYPE HARDWARE, TECHNOLOGY AND SYSTEM INTEGRATION

The prototype created for this project is meant to exemplify multiple ways of sensing gesture language and transmitting that information for use by another system. Specifically, the user's hand motions and finger-flexing will be translated to mouse movement and clicking on a computer. In order to accomplish this, a breadboard was used to connect a flex sensor [7], an Inertial Measurement Unit or IMU [8], a microcontroller [9] and a Bluetooth communications module [10] to a small power supply. The flex sensor was given a length of wire such that when the breadboard was mounted on the user's hand, the flex sensor could be fitted to a finger. This allows for the motion capture of the hand by the IMU and the flection capture of the finger to be monitored and transmitted via Bluetooth to a computer. The resulting prototype is captured in Figure 1 below.

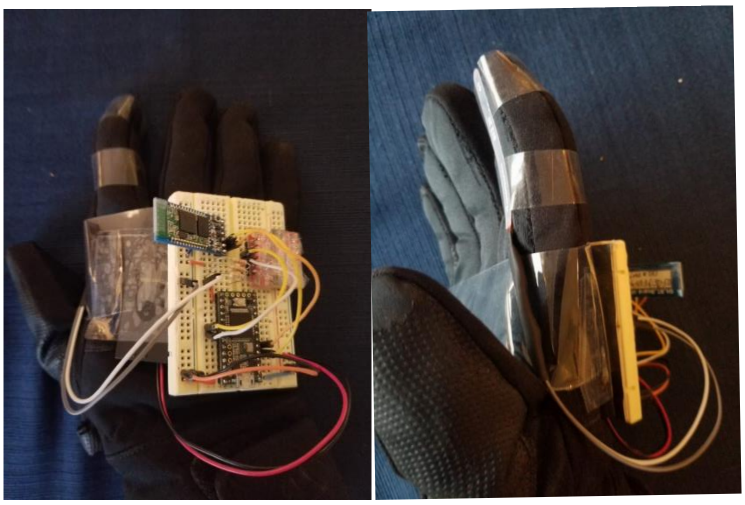


Figure 1: Completed Prototype

A. Microcontroller

The microcontroller utilized for the prototype is the Teensy 3.2 [9]. The Teensy was chosen as a low-cost solution that has a multitude of digital I/O pins, analog-to-digital converters and a corresponding programming extension for the Arduino IDE [11]. This IDE and extension were important for supporting open-source libraries and software applications for the Teensy that made interfacing with the other sensors and communications easier.

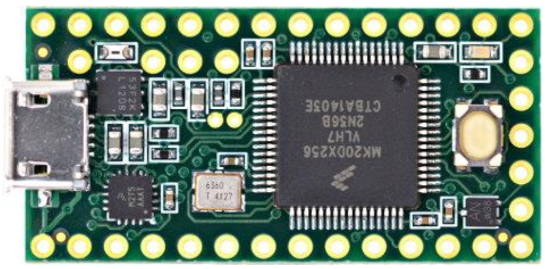


Figure 2: Teensy 3.2 [9]

B. Flex Sensor

The flex sensor used in the prototype is actually the same one used in the Nintendo Power Glove [1], manufactured by Spectra Symbol [7]. For this project a resistive flex sensor was desired to make data processing at the microcontroller simpler. The resistance seen across the terminals of the flex sensor, seen below in Figure 3, increases when the sensor is flexed. By hooking this sensor up in a voltage divider circuit, as seen below in Figure 4, an analog voltage corresponding to the flection of the sensor can be set up. The equation governing this relationship is captured below in Equation 1, where the value of R_{eq} was selected so that the voltage sensed at the microcontroller, V_{sense}, is half of the supply voltage, V_{dd}, when the sensor is unflexed or neutral.

Figure 3: Flex Sensor [7]

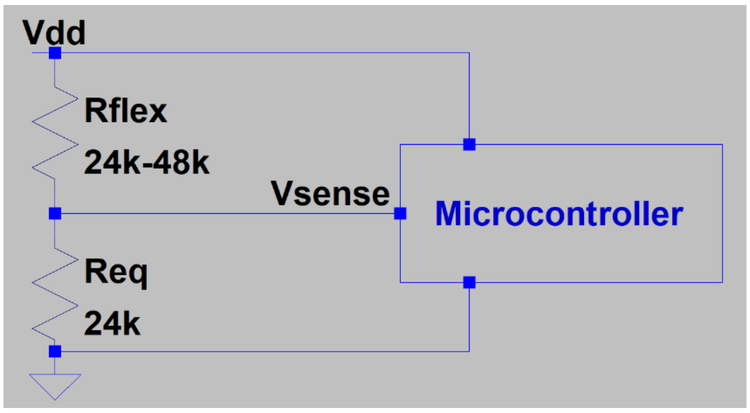


Figure 4: Flex Sensor Circuit

$$V_{sense} = V_{dd} * (R_{eq}) / (R_{eq} + R_{flex}) \quad (1)$$

C. IMU

For this prototype, an IMU was selected that exceeded the functionality needed for the prototype itself. The ICM-20948 Breakout Board by Sparkfun contains a triple-axis MEMS accelerometer, triple-axis MEMS gyroscope and more, which can be used to theoretically compute absolute position of the IMU relative to its starting position [8]. This IMU also has on-board digital filtering and data processing capabilities, separate from the processing power of the microcontroller [12]. A programming library already existed for the ICM-20948 which allowed the Teensy to easily parse the I2C communications received at the microcontroller from the breakout board and access individual axes of acceleration or gyroscope data for further processing [13].

Given that the application of this prototype is to parallel the motion of a computer mouse, only two axes are considered for acceleration. Also, to simplify programming, a restriction is placed on the user that their hand shall not turn or twist relative to the starting orientation. This allows the program to ignore gyroscope information as well and focus entirely on data from the two operative axes of the accelerometer.

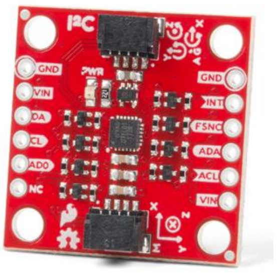


Figure 5: IMU Breakout Board [8]

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 9, Issue 12, December 2020

DOI 10.17148/IJARCCE.2020.91202

D. Bluetooth Module

An important part of this project is the remote transfer of sensor data from the mounted sensors to another system for remote use. To accomplish this, Bluetooth wireless communication was selected because the intended receiving systems, such as phone and computer, are usually equipped for Bluetooth. The HC05 module, seen in Figure 6 below, transmits UART-Serial communications over Bluetooth frequencies [10]. This is the same protocol by which market USB devices such as mice and keyboards communicate with computers or phones, and so testing a MEMS gesture language recognition system with this protocol improves its marketability.

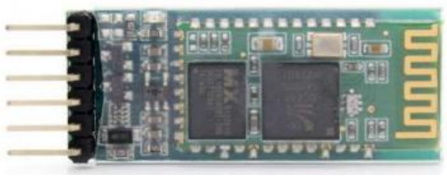


Figure 6: HC05 Bluetooth Module [10]

E. Power Supply

In order to support a wearable system, the power supply has to be small in size and light in weight while still meeting the power needs of the system. The Teensy runs at a nominal 5V but as input voltage regulation circuitry that allows for 6V supply voltage to be used. To meet these needs, CR2032 coin batteries were selected for their high voltage and energy density at their size. Two CR2032 batteries in series provide 6.0V and 220mAh or approximately 3 hours of operation for this prototype under constant use [14]. Putting these in a simple holder with a power switch, such as in Figure 7 below, allows for safe and easy use as part of the prototype [15].

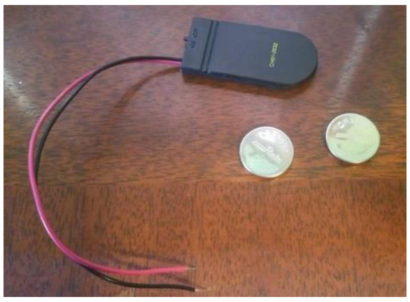


Figure 7: CR2032 Batteries [14] and Holder [15]

III. Prototype Software and Algorithms

In this prototype system, the only software being written is for the Teensy microcontroller. This program had to receive the data from the flex sensor, read as a digital value from an ADC pin, and the IMU, communicated over I2C, and send data to the receiving system that simulates a computer mouse, through Serial and Bluetooth protocols. In order to simplify this process, existing libraries, extensions and coding frameworks were used. The Teensy is firmware-boot loaded for Arduino language and can utilize the Teensyduino application extension for access to enhanced functionality [11]. Within the Teensyduino framework, code was written that addressed the flex sensor, initialization of the IMU and the processing of the IMU data to determine mouse movements. The final operational version of this code is contained in Appendix A, with excerpts contained in Appendix B of attempted methods of programming that proved unsuccessful and were not used.

A. Flex Sensing

The ADC built onto the Teensy gives 10 bits of resolution, so the program receives a data value from the flex sensor between 0 and 1023. For the purposes of this project, the flex sensor is considered to be either flexed or not flexed, resulting in either a mouse click or no mouse click. To reduce the 1024 possible digital values to these 2 states, binary mapping is done about a threshold value, flexT. The value of flexT that effectively establishes binary mapping for the prototype was experimentally determined to be at 650. Values above 650 mean that the sensor is unflexed and therefore unclicked, while values under 650 imply flection and result in a mouse click. This code is captured in Appendix A at comment “Flex Sensing”.

Vol. 9, Issue 12, December 2020
DOI 10.17148/IJARCCE.2020.91202

B. IMU Initialization

The ICM-20948 breakout board contains additional processing power and filtering options that must be configured during startup. The Teensy cannot directly access these but can perform configuration and initialization over I2C communication. A software library was written by the group that made the breakout board which contains functions and example configuration setups that were used in this program [13]. Specifically, the library was used to establish the generic initialization code which can be found in Appendix A at comment "IMU Initialization" as well as the configuration and initialization of the digital low-pass filter, or DLPF. The DLPF can be configured to any of eight different frequencies, where this low-pass frequency sets the rate at which data can safely be sampled and any higher-frequency noise will be blocked [12]. Based on this frequency, f_n, the sampling period can be set for the program, "step" measured in milliseconds, by Equations 2, 3 and 4 below. The code for this is captured in Appendix A at comment "DLPF Configuration and Initialization".

$$f_n \geq 2 * f_s \quad \text{in Hz} \tag{2}$$
$$f_s = 1000 / \text{step} \quad \text{in Hz} \tag{3}$$
$$\text{step} \geq 2000 / f_n \quad \text{in ms} \tag{4}$$

In order to model the motion of the IMU, its starting metrics must be known. What that meant for this project is that the operator's hand had to be unmoving during the startup of the system so that baseline values for not moving could be established. In the program, this manifests as a one-time loop of 100 measurements taken over one second that were averaged to provide the offset values from the accelerometer that corresponded to zero acceleration, or neutral motion. This can be found in Appendix A at comment "Startup Neutralization".

C. Trapezoidal Integration

In an ideal world, a mouse cursors movement's mirror the positional movements of the mouse itself, perhaps with scaling factor. Since the data collected from the IMU is acceleration based, position data could theoretically be calculated through double integration. In practice, however, it is impossible to implement true integrals in a computer program due to the discrete nature of computations. Therefore, the ideal Newtonian kinematic equations captured in Equation 5 below must be reimagined for computation. This was attempted by three different methods in this project.

$$v(t) = v_0 + \int_0^t a(t)\,dt \qquad x(t) = x_0 + \int_0^t v(t)\,dt \tag{5}$$

Trapezoidal integration is one such method. Similar to classical Riemann sums, trapezoidal integration takes discrete averages of acceleration values over a period of time and summarizes them in accordance with Equation 6 below. The same method can then be applied to the integration of velocity for position. For this project, trapezoidal integration was only taken for the velocity and the code for this can be found in Appendix B. Trapezoidal integration was not attempted for position because there is no opportunity in the system for error to occur between the velocity and position, so if the technique works to collect velocity data then the position should follow directly. The program implements trapezoidal integration over N steps, as in Equation 6, each step of time "step". While this method does come the closest to true integration to position, it also suffers most severely from error in the acceleration data. Since the IMU did suffer from substantial error and inconsistency in acceleration data.

$$v = v0 + \sum_n (a_n + a_{n-1}) / 2 * (t_n - t_{n-1}) \tag{6}$$

D. Trinary Accumulation

On close examination of the acceleration data coming in, statistical outliers were very common even when the sensor was held neutral. The extreme magnitude of these outliers would have a large effect on the velocity and larger impact on the position. This manifests as an instantaneous jump of the position to a very large number, sometimes too large to reasonably neutralize with normal movement. In order to eliminate this aspect of error, the magnitude factor was removed by the trinary accumulation method. The method is illustrated in Figure 8 below.

Vol. 9, Issue 12, December 2020

DOI 10.17148/IJARCCE.2020.91202

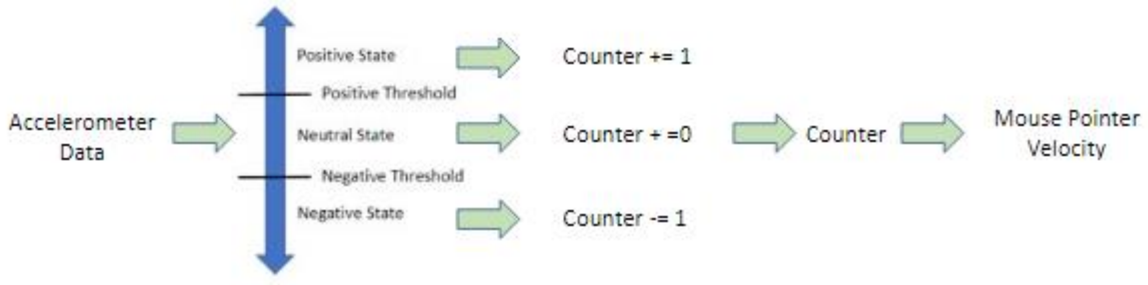
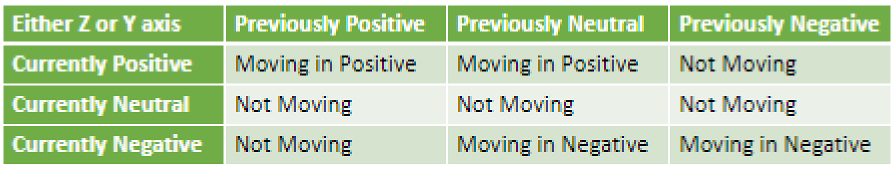


Figure 8: Trinary Accumulation Methodology

The trinary accumulation method is not a mathematical function but rather a three-state machine model, whereby the measured acceleration is categorized either as “positive”, “negative” or “neutral” based on experimentally determined thresholds. Acceleration data points were still taken by extensive average, just like in the trapezoidal method, but the state of each accelerometer data point had a set effect on a counter variable without regard for magnitude. After each point is counted, the sum value of the counter variable would correspond to the mouse velocity. However, with this method implemented as in Appendix B, error would still accumulate to a critical point where normal operation would become insignificant.

E. Trinary Logic

To further safeguard the mouse motion from the accumulation of error in the accelerometer data, a third method known as trinary logic was attempted. Trinary logic is a state system based around the same three state interpretation of accelerometer data points as trinary accumulation, but makes a logical decision based on the most recent data points instead of accumulating these conditions in a counter variable. In order to determine how the mouse should be moving at any given time, the state of acceleration at that time and the most recent previous time are considered. As seen in Figure 9 below, these result combinations result in either a positive, neutral or negative statement about motion in each direction, and that decision sets the velocity of the mouse cursor. This was the method ultimately settled on for the prototype, and so the code for its implementation can be found in Appendix A at comments “Managing current state and prev state” and “Setting motion commands”.



Either Z or Y axis	Previously Positive	Previously Neutral	Previously Negative
Currently Positive	Moving in Positive	Moving in Positive	Not Moving
Currently Neutral	Not Moving	Not Moving	Not Moving
Currently Negative	Not Moving	Moving in Negative	Moving in Negative

Figure 9: Trinary Logic Table

IV. RESULTS

The trapezoidal integration model was the closest model for the kinematics at play that was also computable, but because it used the raw numerical data from the accelerometer, it was also vulnerable to the imperfections of that hardware. When bad accelerometer data was interpreted by this method, the velocity and furthermore position gave this data equal weight to the actual motion of the system. Due to the summary nature of this model, bad data accumulates into either extremely negative or extremely positive values after each integration level, and the normal operation of the system loses the ability to return to neutral. In quantifiable terms, this would happen within thirty seconds with this model as composed of the code from Appendices A and B. This quantity will be used purely for comparing methods, since its specific value may vary with hardware and environment.

The trinary accumulation method allowed for more time before critical error accumulated, approximately two minutes. The trinary logic method, since no accumulation or summation occurs, does not grow unstable in this way. But while trinary state handling reduces the significance of bad accelerometer data on the system, it also reduces the sensitivity of the system to true and accurate accelerometer data. Additionally, an assumption was made in the trinary logic table seen in Figure 9 that when acceleration states reversed it meant that the system was slowing down or stopping motion instead of completely reversing direction of motion. Both patterns of motion would result in these changes in acceleration, but it is impossible to tell which pattern is present without integrating the acceleration data into a concise velocity tracker. By deciding that both patterns should result in stopping the mouse from moving, the full pattern of motion where the

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 9, Issue 12, December 2020

DOI 10.17148/IJARCCE.2020.91202

operator’s hand slows down, stops and then reverses is very uneven. The resulting movement is jerky and imprecise but has the advantage of never losing stability.

Ultimately, this tradeoff is only necessary because of the limitations on accuracy with the IMU’s accelerometer. This IMU is not alone in its inability to handle live position-processing, nor is this deficiency lost on the industry. In 2012 an article was written by CHRobotics/Redshift Labs where they attempted to implement position tracking using their UM-7 3-axis accelerometer with a direct-integration algorithm. They put an emphasis on knowing the precise orientation of the sensor as error can be caused by the presence of external forces such as gravity or normal force on multiple axes [16]. This is handled by the individual axes’ startup neutralization in this project as well as the project operation guidelines, but in open practice these guidelines cannot be guaranteed. Ultimately, they were seeing meaningful position error at 10 seconds of operation and massive error after a minute. They concluded that “it is possible to use the accelerometers to estimate velocity and position, but the accuracy will be very poor so that in most cases, the estimates are not useful” [16].

V. FUTURE RESEARCH

The focus of this research was to create a glove controller to control the movement and input of a mouse using current low-cost MEMS technology. Although the desired results were acquired, many improvements upon this system could be made. One such improvement could be the use of a ‘Microcontroller Based Potentiometric Indicator System for Piezoresistive MEMS Sensor’ over a basic flex sensor. The system mentioned has two hardware subsystems, the first consisting of a sensor element and signal conditioning and the second being a liquid crystal display [17]. The software acquires analog data detected by a piezoresistive MEMS sensor and converts the data form with ADC, computes the digital data conversion into desired reading by using fuzzy logic controller [17]. By using these advanced computing techniques, the detectable ranges for differing ‘bent’ positions could be more accurately represented allowing for more robust gesture commands. Currently our system only utilizes the 3-axis accelerometer, which is used to detect the acceleration of the IMU in a certain direction to determine the direction and speed of the mouse movement. This method does not currently consider hand orientation and rotation, which could be improved upon for a more robust system. Another group of researchers designed a ‘Microcontroller-Based MEMS Rate Integrating Gyroscope Module with Automatic Asymmetry Calibration’. This system was designed to automatically calibrate the module on initialization, to accurately predict future movement based on the starting position of the hand [18]. The addition of this automatic calibration, along with the use of a gyroscope, to our design would greatly increase the accuracy of our detected movements based on the initialized values from the IMU from a resting state. Another improvement which could be made, would be the active filtering of accelerometer data processed from the IMU. As mentioned before, in the Trapezoidal integration method, erroneous data is compounded which can exponentially affect the calculated positioning of the accelerometer. By introducing a finite impulse response (FIR) filter [19], the noise obtained from the IMU can be greatly reduced, resulting in a smoother signal and minimizing the error detected when using the Trapezoidal integration method.

VI. CONCLUSION

At the time of this writing, gesture language technology development and research are both commonplace at a variety of modern tech companies and are implemented in many consumer products today as well. This paper presents the idea of utilizing cost-efficient MEMS technology to recognize the motion of a human hand to control a computer mouse and the bending of a finger to simulate the click of a mouse. Although the desired results of this project were obtained, many improvements could also be made. The Trapezoidal integration method which was used results in exponential errors due to the estimations during its calculations, which could possibly be reduced by the application of active filtering methods. The trinary accumulation method, used to determine the state of the flex sensor, is limited in its ability to detect more complex motion. The idea presented in this paper uses existing MEMS technology to explore the application of gesture language recognition in a standard computer-controlled environment. This idea can be easily implemented and further developed in other computer control applications.

VII. ACKNOWLEDGEMENT

Harrison Keats was responsible for the hardware assembly, programming, and simulation of this project, as well as writing and editing this paper. **Ben Lindquist** assisted with hardware, programming, and literature research as well as writing and editing this paper. **Dean Aslam** took the role of the overall mentor of the project, helping to consult the authors and provide feedback as we progressed through the project. The team all collaborated on this final paper, each taking different sections building it into one paper.

REFERENCES

1. A. N. Shete1 and K. D. Garkal, "Mirror neurons and their role in communication", Int J Res Med Sci., 4(8), 3097-3101, 2016; DOI: http://dx.doi.org/10.18203/2320-6012.ijrms20162265
2. Center, S. (2017, February 13). Success Born of Failure: The Nintendo Power Glove. Retrieved November 12, 2020, from https://invention.si.edu/success-born-failure-nintendo-power-glove
3. *Prime Faraday Technology Watch An Introduction to MEMS (Micro-Electromechanical Systems).* https://www.lboro.ac.uk/microsites/mechman/research/ipm-ktn/pdf/Technology_review/an-introduction-to-mems.pdf
4. P. Telluri, S. Manam, S. Somarouthu, J. M. Oli and C. Ramesh, "Low cost flex powered gesture detection system and its applications," *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*, Coimbatore, India, 2020, pp. 1128-1131, doi: 10.1109/ICIRCA48905.2020.9182833. https://ieeexplore-ieee-org.proxy2.cl.msu.edu/document/9182833
5. M. B. H. Flores, C. M. B. Siloy, C. Oppus and L. Agustin, "User-oriented finger-gesture glove controller with hand movement virtualization using flex sensors and a digital accelerometer," *2014 International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*, Palawan, 2014, pp. 1-4, doi: 10.1109/HNICEM.2014.7016195. https://ieeexplore-ieee-org.proxy2.cl.msu.edu/document/7016195
6. X. Zhang and X. Wu, "Robotic Control of Dynamic and Static Gesture Recognition," *2019 2nd World Conference on Mechanical Engineering and Intelligent Manufacturing (WCMEIM)*, Shanghai, China, 2019, pp. 474-478, doi: 10.1109/WCMEIM48965.2019.00100. https://ieeexplore-ieee-org.proxy2.cl.msu.edu/document/9004188
7. Industries, Adafruit. "Short Flex Sensor." *Adafruit Industries Blog RSS*, www.adafruit.com/product/1070.
8. #589719, Member, et al. "SparkFun 9DoF IMU Breakout - ICM-20948 (Qwiic)." *SEN-15335 - SparkFun Electronics*, www.sparkfun.com/products/15335.
9. "Teensy USB Development Board." *PJRC*, www.pjrc.com/store/teensy32.html.
10. "Bluetooth Module HC-05 , Serial TTL 'Most Popular.'" *RAM Electronics*, 4 Oct. 2018, ram-e-shop.com/product/kit-bluetooth-hc05/.
11. "Using USB Mouse." *PJRC*, www.pjrc.com/teensy/td_mouse.html.
12. "ICM-20948: TDK." *InvenSense*, invensense.tdk.com/products/motion-tracking/9-axis/icm-20948/.
13. Sparkfun. "Sparkfun/SparkFun_ICM-20948_ArduinoLibrary." *GitHub*, github.com/sparkfun/SparkFun_ICM-20948_ArduinoLibrary.
14. "CR2032-HE8." *CR2032-HE8|Micro Batteries|Batteries|Murata Manufacturing Co., Ltd.*, www.murata.com/en-us/products/productdetail?partno=CR2032-HE8.
15. "122-0120-GR Eagle Plastic Devices: Mouser." *Mouser Electronics*, www.mouser.com/ProductDetail/Eagle-Plastic-Devices/122-0120-GR?qs=F5EMLAvA7IAn58ST31xThQ==.
16. "Using Accelerometers to Estimate Position and Velocity." *CH Robotics*, 2012, www.chrobotics.com/library/accel-position-velocity.
17. M. F. Abdullah, M. A. Adnan, M. G. A. Aziz and N. K. Madzhi, "Development of Microcontroller Based Potentiometric Indicator System for Piezoresistive MEMS Sensor," 2013 Fifth International Conference on Computational Intelligence, Communication Systems and Networks, Madrid, 2013, pp. 155-159, doi: 10.1109/CICSYN.2013.61. https://ieeexplore-ieee-org.proxy2.cl.msu.edu/document/6571358
18. R. Gando, D. Ono, S. Kaji, H. Ota, T. Itakura and Y. Tomizawa, "A Compact Microcontroller-Based MEMS Rate Integrating Gyroscope Module with Automatic Asymmetry Calibration," 2020 IEEE 33rd International Conference on Micro Electro Mechanical Systems (MEMS), Vancouver, BC, Canada, 2020, pp. 1296-1299, doi: 10.1109/MEMS46641.2020.9056317. https://ieeexplore-ieee-org.proxy2.cl.msu.edu/document/9056317
19. S. Chikhalikar, O. Khandekar and C. Bhattacharya, "Design of Real-Time Acquisition and Filtering for MEMS-based Accelerometer Data in Microcontroller," 2018 IEEE Electron Devices Kolkata Conference (EDKCON), Kolkata, India, 2018, pp. 15-18, doi: 10.1109/EDKCON.2018.8770488. https://ieeexplore-ieee-org.proxy2.cl.msu.edu/document/8770488

Appendix A – Final Code

```
#include <ICM_20948.h>
#include <Mouse.h>
#define SERIAL_PORT Serial
#define WIRE_PORT Wire
#define AD0_VAL 1

ICM_20948_I2C myICM;
int flexT = 650;
float Aystart = 0;  //Startup Neutralization values
float Azstart = 0;
float Aypos = 5;    //Y-axis thresholds
float Ayneg = -5;
float Azpos = 5;    //Z-axis thresholds
float Azneg = -5;
int Ypstate = 0;    //Previous state variables for axes
int Zpstate = 0;
int ms = 5;         //Pixels per dt
float time = 0;


void setup() {
  Serial.begin(9600);
```

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 9, Issue 12, December 2020

DOI 10.17148/IJARCCE.2020.91202

```
  pinMode(14,INPUT);
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);
  Mouse.begin();
  Mouse.screenSize(1920, 1080);  // configure screen size
  Mouse.moveTo(960,540);         // Center the mouse

  //IMU Initialization
  WIRE_PORT.begin();
  WIRE_PORT.setClock(16000000);
  bool initialized = false;
  while(!initialized){
    myICM.begin(WIRE_PORT, AD0_VAL);
    SERIAL_PORT.print( F("Initialization of the sensor returned: ") );
    SERIAL_PORT.println( myICM.statusString() );
    if( myICM.status != ICM_20948_Stat_Ok ){
      SERIAL_PORT.println( "Trying again..." );
      delay(500);
    }
    else{
      initialized = true;
    }
  }

  //DLPF Configuration and Initialization
  ICM_20948_dlpcfg_t myDLPcfg;
  myDLPcfg.a = acc_d246bw_n265bw;   //Set DLPF for Accelerometer to Nyquist BW of 265Hz
  myDLPcfg.g = gyr_d361bw4_n376bw5; //Set DLPF for Gyroscope
  myICM.setDLPFcfg( (ICM_20948_Internal_Acc | ICM_20948_Internal_Gyr), myDLPcfg );  //Check config
  if( myICM.status != ICM_20948_Stat_Ok){
    SERIAL_PORT.print(F("setDLPcfg returned: "));
    SERIAL_PORT.println(myICM.statusString());
  }
  ICM_20948_Status_e accDLPEnableStat = myICM.enableDLPF( ICM_20948_Internal_Acc, true );
  ICM_20948_Status_e gyrDLPEnableStat = myICM.enableDLPF( ICM_20948_Internal_Gyr, true );
  SERIAL_PORT.print(F("Enable DLPF for Accelerometer returned: "));
  SERIAL_PORT.println(myICM.statusString(accDLPEnableStat));
  SERIAL_PORT.print(F("Enable DLPF for Gyroscope returned: "));
  SERIAL_PORT.println(myICM.statusString(gyrDLPEnableStat));
  SERIAL_PORT.println();
  SERIAL_PORT.println(F("Configuration complete!"));

  //Startup Neutralization
  //Over 1 second, take an average neutral value for acceleration in Z and Y
  //Requires user to hold IMU still for 1 second on startup/reset
  int i;
  int k = 100;
  float aysum = 0;
  float azsum = 0;
  for (i=0;i<k;i++){
    while(!myICM.dataReady()){}
    myICM.getAGMT();
    aysum += myICM.accY();
    azsum += myICM.accZ();
    delay(10);
  }
  Aystart = aysum / k;
  Azstart = azsum / k;
}
```

Vol. 9, Issue 12, December 2020

DOI 10.17148/IJARCCE.2020.91202

```
void loop() {
  while(!myICM.dataReady()){}
  myICM.getAGMT();


  float step = 10;  //Step size in milliseconds
  float N = 1;   //Number of steps of 'step' milliseconds
  time = time + step * N / 1000; //Time in seconds
  //Frequency of sampling then is
  float i = 0;    //Step counter
  float prevAy = myICM.accY();
  float prevAz = myICM.accZ();
  float Aysum = 0;
  float Azsum = 0;

  for(i=0;i<N;i++){
    delay(step);
    while(!myICM.dataReady()){}
    myICM.getAGMT();
    Aysum += 0.5 * (myICM.accY() + prevAy) - Aystart; //Average acceleration added over each N step within dt
    Azsum += 0.5 * (myICM.accZ() + prevAz) - Azstart; //Average acceleration added over each N step within dt
    prevAy = myICM.accY();
    prevAz = myICM.accZ();
  }
  float Az = Azsum / N; //Average acceleration over 0.5 second
  float Ay = Aysum / N; //Average acceleration over 0.5 second

  //Limit Finding
//  if(Az > Azpos){
//    Azpos = Az;
//  }
//  else if(Az < Azneg){
//    Azneg = Az;
//  }
//  if(Ay > Aypos){
//    Aypos = Ay;
//  }
//  else if(Ay < Ayneg){
//    Ayneg = Ay;
//  }

  //Managing current state and prev state
  int Z = 0;
  int Y = 0;
  if(Az > Azpos){
    Z = 1;
  }
  else if(Az < Azneg){
    Z = -1;
  }
  if(Ay > Aypos){
    Y = 1;
  }
  else if(Ay < Ayneg){
    Y = -1;
  }

  //Setting motion commands
  //Zpos = yneg for mouse
  int ymov = 0;
```

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 9, Issue 12, December 2020
DOI 10.17148/IJARCCE.2020.91202

```
if(Zpstate == 1 && Z == 1){
  //Moving in Zpos
  ymov = -1;
}
else if(Zpstate == 1 && Z == 0){
  //Not moving
}
else if(Zpstate == 1 && Z == -1){
  //Not moving
}

else if(Zpstate == 0 && Z == 1){
  //Moving in Zpos
  ymov = -1;
}
else if(Zpstate == 0 && Z == 0){
  //Not moving
}
else if(Zpstate == 0 && Z == -1){
  //Moving in Zneg
  ymov = 1;
}

else if(Zpstate == -1 && Z == 1){
  //Not moving
}
else if(Zpstate == -1 && Z == 0){
  //Not moving
}
else if(Zpstate == -1 && Z == -1){
  //Moving in Zneg
  ymov = 1;
}

//Ypos = xneg for mouse
int xmov = 0;
if(Ypstate == 1 && Y == 1){
  //Moving in Ypos
  xmov = -1;
}
else if(Ypstate == 1 && Y == 0){
  //Not moving
}
else if(Ypstate == 1 && Y == -1){
  //Not moving
}

else if(Ypstate == 0 && Y == 1){
  //Moving in Ypos
  xmov = -1;
}
else if(Ypstate == 0 && Y == 0){
  //Not moving
}
else if(Ypstate == 0 && Y == -1){
  //Moving in Yneg
  xmov = 1;
}

else if(Ypstate == -1 && Y == 1){
```

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 9, Issue 12, December 2020

DOI 10.17148/IJARCCE.2020.91202

```
   //Not moving
  }
  else if(Ypstate == -1 && Y == 0){
   //Not moving
  }
  else if(Ypstate == -1 && Y == -1){
   //Moving in Yneg
   xmov = 1;
  }
  Ypstate = Y;
  Zpstate = Z;
  Mouse.move(xmov * ms, ymov * ms);


  //Flex Sensing
  int flexval = analogRead(14);
  if (flexval > flexT) {       //Unflexed
   digitalWrite(LED_BUILTIN, LOW);
  }
  else {                      //Flexed
   digitalWrite(LED_BUILTIN, HIGH);
   Mouse.click();
  }


  //Debug Outputs
  Serial.print("Time: ");
  Serial.println(time);
//  Serial.print("AccY: ");
//  Serial.println(Ay);
//  Serial.print("AccZ: ");
//  Serial.println(Az);
//  Serial.print("Aypos: ");
//  Serial.println(Aypos);
//  Serial.print("Azpos: ");
//  Serial.println(Azpos);
//  Serial.print("Ayneg: ");
//  Serial.println(Ayneg);
//  Serial.print("Azneg: ");
//  Serial.println(Azneg);
//  Serial.print("Y: ");
//  Serial.println(Y);
//  Serial.print("Z: ");
//  Serial.println(Z);
  Serial.print("Xmov: ");
  Serial.println(xmov);
  Serial.print("Ymov: ");
  Serial.println(ymov);
}
```

Appendix B – Code Excerpts

Trapezoidal Integration

```
//Global declarations
float Aystart = 0;
float Azstart = 0;
float Vy = 0;
float Vz = 0;
float Y = 0;
float Z = 0;
float t = 0;
```

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 9, Issue 12, December 2020

DOI 10.17148/IJARCCE.2020.91202

```
void loop() {
  while(!myICM.dataReady()){}
  myICM.getAGMT();

  float step = 5;  //Step size in milliseconds
  float N = 100;   //Number of steps of n milliseconds
  float dt = step * N / 1000; //Time in seconds
  //Frequency of sampling then is
  float i = 0;    //Step counter
  float prevAy = myICM.accY();
  float prevAz = myICM.accZ();
  float Aysum = 0;
  float Azsum = 0;


  for(i=0;i<N;i++){
    delay(step);
    while(!myICM.dataReady()){}
    myICM.getAGMT();
    Aysum += 0.5 * (myICM.accY() + prevAy) - Aystart; //Average acceleration added over each N step within dt
    Azsum += 0.5 * (myICM.accZ() + prevAz) - Azstart; //Average acceleration added over each N step within dt
    prevAy = myICM.accY();
    prevAz = myICM.accZ();
  }

  Vy += dt * Aysum / N; // Velocity as of time (t) [seconds * mg]
  Vz += dt * Azsum / N; // Velocity as of time (t) [seconds * mg]
  Y += dt * Vy; // Position as of time (t) [seconds * seconds * mg]
  Z += dt * Vz; // Position as of time (t) [seconds * seconds * mg]

  //After the loop, V and Y are correct for t = +N*0.01 = +0.5 seconds
  //So every half second this algorithm will determine the velocity and position of the sensor
  //***Relative to its starting position and with assumed 0 velocity on startup
  t += dt;
}
```

Trinary Accumulation

```
//Global Declarations
float Aystart = 0;  //Startup Neutralization values
float Azstart = 0;
float Aypos = 5;    //Y-axis thresholds
float Ayneg = -5;
float Azpos = 5;    //Z-axis thresholds
float Azneg = -5;
int Y = 0;
int Z = 0;
float time = 0;
void loop() {
  while(!myICM.dataReady()){}
  myICM.getAGMT();


  float step = 10;  //Step size in milliseconds
  float N = 50;   //Number of steps of n milliseconds
  time = time + step * N / 1000; //Time in seconds
  //Frequency of sampling then is
  float i = 0;    //Step counter
  float prevAy = myICM.accY();
  float prevAz = myICM.accZ();
  float Aysum = 0;
  float Azsum = 0;

  for(i=0;i<N;i++){
    delay(step);
    while(!myICM.dataReady()){}
```

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 9, Issue 12, December 2020
DOI 10.17148/IJARCCE.2020.91202

```
  myICM.getAGMT();
  Aysum += 0.5 * (myICM.accY() + prevAy) - Aystart; //Average acceleration added over each N step within dt
  Azsum += 0.5 * (myICM.accZ() + prevAz) - Azstart; //Average acceleration added over each N step within dt
  prevAy = myICM.accY();
  prevAz = myICM.accZ();
}
float Az = Azsum / N; //Average acceleration over 0.5 second
float Ay = Aysum / N; //Average acceleration over 0.5 second

//Limit Finding
if(Az > Azpos){
  Azpos = Az;
}
else if(Az < Azneg){
  Azneg = Az;
}
if(Ay > Aypos){
  Aypos = Ay;
}
else if(Ay < Ayneg){
  Ayneg = Ay;
}

//Accumulating positivity or negativity of acceleration over time
if(Az > Azpos){
  Z = Z + 1;
}
else if(Az < Azneg){
  Z = Z - 1;
}
if(Ay > Aypos){
  Y = Y + 1;
}
else if(Ay < Ayneg){
  Y = Y - 1;
}
```