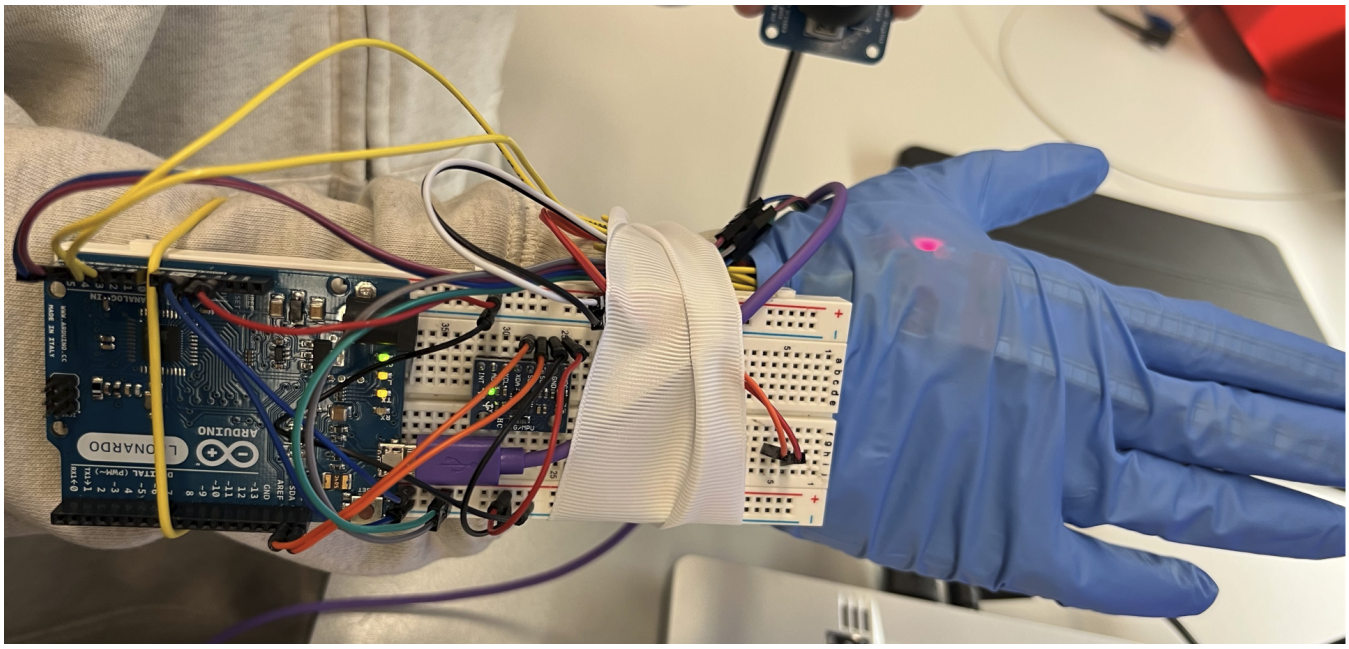
Glomo the Glove Mouse



Group 39

Sumanth Marakini Shivshankar & Arjun Makwana

Final Project, Fall 2021

ESE519: Real-Time and Embedded Systems

University of Pennsylvania

Table of Contents

Abstract

The following paper details the design process of a portable device which is able to sense hand gestures and convert them into a cursor motion on the computer screen like how a mouse would. A device integrated with buttons to perform the functionality of left and right clicks, selection and gaming modes (ADS in First Person Perspective gaming). It can further be improved by integrating a joystick capable of sensing the ‘W’, ‘A’, ‘S’, ‘D’ keys from the keyboard facilitating player movement for gaming.

A 3-axis Gyroscope (3 Degree of Freedom) of the MPU 6050 sensor is used to control cursor movement. A 3D gyroscope measures the angular velocity of its rotation in a reference frame along its three sensitivity axes. For an ideal sensor, these outputs are equal to the projections of the rotation angular velocity on the sensitivity, that is, intrinsic coordinate system axes. Then, each of the voltage values from the sensor is converted to digital numbers. These numbers are converted to final mouse movement values after reducing the noise and scaling the output to a range of values which can be mapped onto the mouse.

Motivation

A standard mouse has a dependency for a desk which may not be the most convenient option to move a cursor around the screen when you want a gaming experience. A conventional mouse not only has a dependency but also does not have additional features for gaming. A gaming mouse is expensive. A smart way to track motion is to use a 3D gyroscope. This is the idea behind Glomo. Using hand motion and gestures to move a mouse pointer on the computer screen with the option of hand held buttons for switching between perspectives/aim in gaming. The integration of the joystick for player movement, integrated with W,A,S,D keys provides a convenient stand alone device which can be used for general purpose as well as for gaming.
Many users who use the mouse on an everyday basis usually suffer from Carpal tunnel syndrome. Carpal tunnel syndrome is caused by increased pressure on the median nerve. The carpal tunnel is a narrow passageway surrounded by bones and ligaments on the palm side of your hand. When the median nerve is compressed, the symptoms can include numbness, tingling and weakness in the hand and arm. This usually occurs by prolonged usage of a standard mouse due to the posture with which it is held. Glomo addresses this issue with an idea that the mouse could be held in air without a specific posture, preventing the pressure on the median nerve.

Goals

A. Milestone 1

- ➢ Precisely read the gyroscope reading from the MPU6050 3 axis gyroscope sensor.
- ➢ Implement the motion of a mouse pointer on a screen by converting the voltage values from the MPU6050 sensor using the ADC of arduino leonardo board..
- ➢ Implement at least one of the gesture movements (movement in x or y direction) on the device being developed.
- ➢ Testing on different computers and wide monitor screens to ensure the correct functionality.

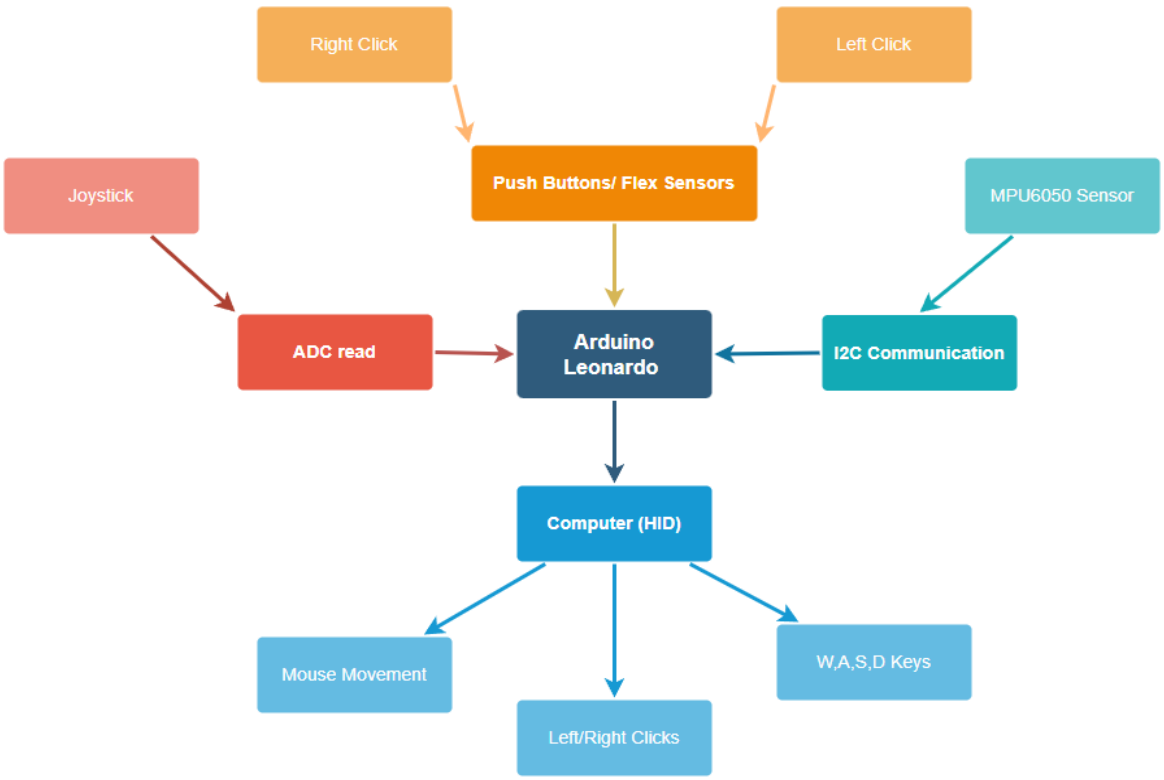
B. Final Demo

- ➢ Reading 3 axis gyroscope angular velocities in the form of voltage from MPU6050 sensor using I2C communication protocol. The analog values are converted using the ADC of the arduino microcontroller and the digital values are stored in a variable which continuously updates.
- ➢ Reading the values from the flex sensors to implement left and right clicks on the index finger, middle finger.
- ➢ Mapping the voltage values on to a specific range of values to implement the mouse movement using the mouse library of the arduino. The arduino leonardo has the HID(Human interface device) capabilities.
- ➢ Mapping the values to the mouse library, in turn providing the functionality of mouse motion on the computer screen, using hand gestures.
- ➢ Converting the change in the flex sensor values to implement left and right clicks.
- ➢ Mounting the device onto a glove to hide the wires and improve the aesthetic appearance of the device.
- ➢ Successful implementation of the left and right button functionality integrated into the device providing mouse clicks as an alternate to the flex sensors depending on the usage.
- ➢ Achieve a quick response time.
- ➢ Integration of a joystick capable of sensing the keys ‘W’, ’A’, ’S’, ’D’ and control motion of a player in a gaming environment.
- ➢ Integration of all the above functional into a compact device to provide the intended functionality for general purpose and for gaming.

Methodology

Block Diagram

The following block diagram represents the overview of the architecture of the system. The block diagram represents the input and output peripherals and the communication between the multiple devices.

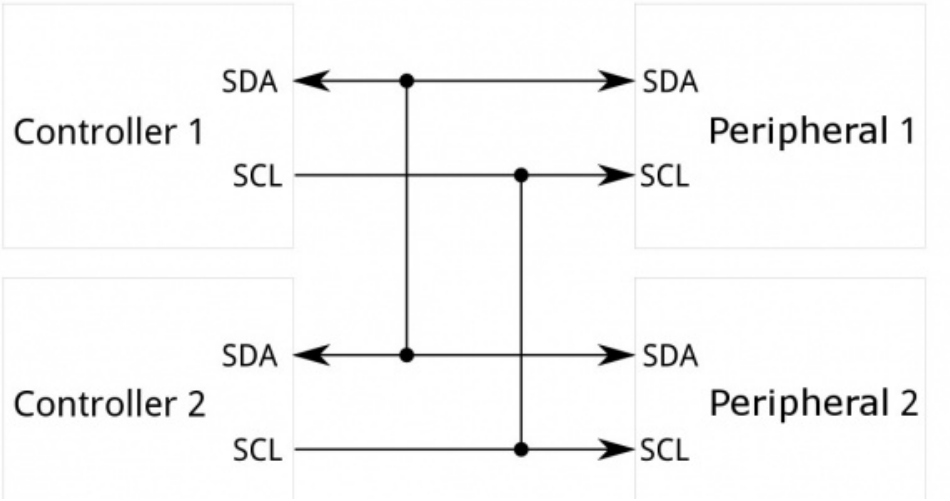


Block Diagram

I2C Communication

The Inter-Integrated Circuit (I2C) Protocol is a protocol intended to allow multiple peripheral digital integrated circuits to communicate with one or more controller chips. Like the Serial Peripheral Interface (SPI), it is only intended for short distance communications within a single device. Like Asynchronous Serial Interfaces (such as RS-232 or UARTs), it only requires two signal wires to exchange information. These two lines are the SDA and the SCL lines.

The following high level diagram represents the overview of the I2C communication protocol:



I2C Communication between two devices

The Serial ports (UART) cannot be used for the implementation because serial ports are asynchronous (no clock data is transmitted) and the devices using them must agree ahead of time on a data rate. The two devices must also have clocks that are close to the same rate, and will remain so--excessive differences between clock rates on either end will cause garbled data.

Asynchronous serial ports require hardware overhead. The UART at either end is relatively complex and difficult to accurately implement in software if necessary. At least one start and stop bit is a part of each frame of data, meaning that 10 bits of transmission time are required for each 8 bits of data sent, which eats into the data rate.

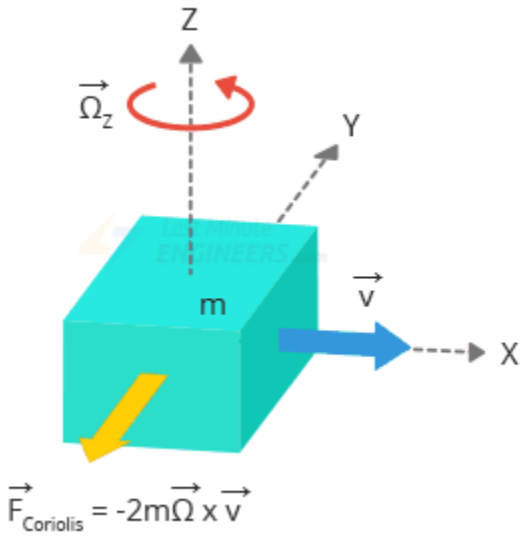
I2C can support a multi-controller system, allowing more than one controller to communicate with all peripheral devices on the bus. Data rates fall between asynchronous serial and SPI. Most I2C devices can communicate at 100kHz or 400kHz. There exists an overhead with I2C. For every 8 bits of data to be sent, one extra bit of metadata (the "ACK/NACK" bit) must be transmitted. The hardware required to implement I2C is more complex than SPI, but less than asynchronous serial. It can be fairly trivially implemented in software.

The MPU6050 uses the I2C protocol to communicate with the Arduino microcontroller.

MPU6050

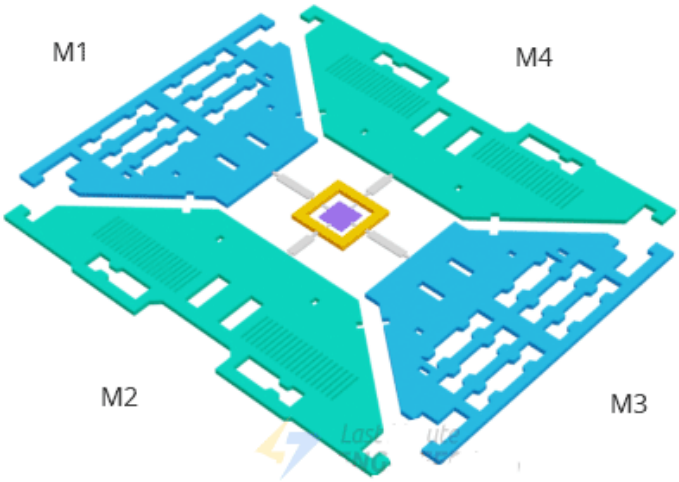
MPU6050 houses a 3 axis MEMS gyroscope. While accelerometers measure linear acceleration, MEMS gyroscopes measure angular rotation. To do this they measure the force generated by what is known as The Coriolis Effect.

Coriolis Effect tells us that when a mass (m) moves in a particular direction with a velocity (v) and an external angular rate (Ω) is applied (Red arrow); the Coriolis Effect generates a force (Yellow arrow) that causes a perpendicular displacement of the mass. The value of this displacement is directly related to the angular rate applied.



Coriolis Effect

The MEMS sensor is composed of a proof mass (M1, M2, M3 and M4) as shown in the diagram below which is kept in a continuously oscillating movement so that it reacts to the coriolis effect. They move inward and outward simultaneously in the horizontal plane.
There are three modes depending on which axis the angular rotation is applied. The Roll, Pitch and Yaw.



MEMS sensor

The Roll :
When an angular rate is applied along the X-axis, due to the coriolis effect, then M1 and M3 will move up and down out of the plane. This causes the roll angle to change hence it is called Roll Mode.

The Pitch:
When an angular rate is applied along the Y-axis, then M2 and M4 will move up and down. This causes the pitch angle to change hence it is called Pitch Mode.

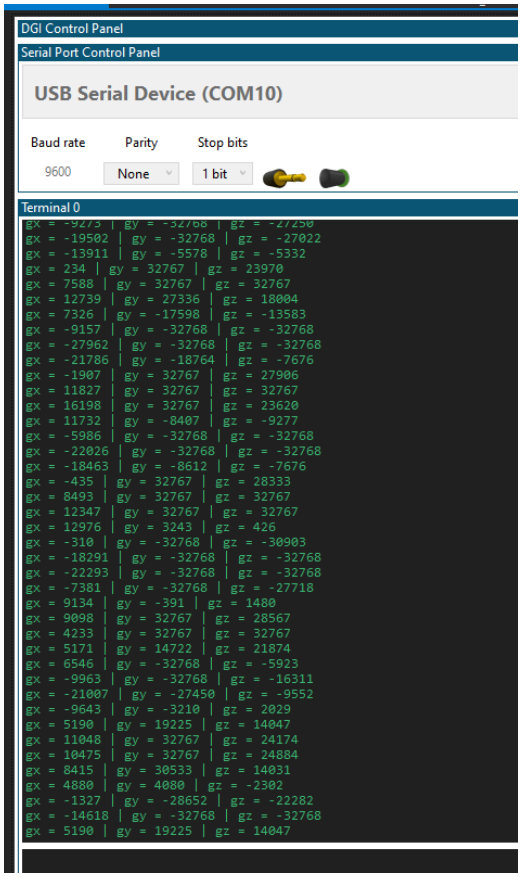
The Yaw:
The following snippet shows the code for reading the values from the MPU6050 sensor using I2C protocol. Another figure also shows the values being read from the sensor.

The MPU6050 module comes with an on-board LD3985 3.3V regulator, so you can use it with a 5V logic microcontroller like Arduino without worry. It consumes less than 3.6mA during measurements and only 5µA during idle. This low power consumption allows the implementation in battery driven devices. The MPU6050 has three 16-bit analog-to-digital converters that simultaneously sample the 3 axis of movement (along X, Y and Z axis).

The MPU6050 can measure angular rotation using its on-chip gyroscope with four programmable full scale ranges of ±250°/s, ±500°/s, ±1000°/s and ±2000°/s. We are using the ±1000°/s range for our application. It supports two separate I2C addresses: 0x68 and 0x69.

The ADO pin determines the I2C address of the module. This pin has a built-in 4.7K pull-down resistor. Therefore, when you leave the ADO pin unconnected, the default I2C address is 0x68 and when you connect it to 3.3V, the line is pulled HIGH and the I2C address becomes 0x69HEX.

The following image shows the reading of gyroscope values using the I2C communication.



Gyroscope readings

The gx, gy and gz values as seen in the Gyroscope readings represent the roll pitch and yaw respectively.

The following snippet shows the initialization of the I2C communication.

```
void init_i2c()
{
            PORTD &= ~(1 << 0); //Port D0 SCL
            PORTD &= ~(1 << 1); //Port D1 SDA
            TWBR = ((( F_CPU / SCL_CLOCK ) - 16) / 2);
            TWSR = 0;
```

```
            TWCR = ( 1 << TWEN ); // enable i2c bus
}
```

The following code snippet shows the reading of the gyroscope values into a buffer and storing it. These values are sent to the microcontroller when it is requested for.

```
void mpu6050_read_gyro_ALL(int16_t * buff){

      uint8_t tmp[2];

      mpu6050_read_gyro_X(tmp);
      buff[0] = (tmp[0]<<8)|(tmp[1]);
      mpu6050_read_gyro_Y(tmp);
      buff[1] = (tmp[0]<<8)|(tmp[1]);
      mpu6050_read_gyro_Z(tmp);
      buff[2] = (tmp[0]<<8)|(tmp[1]);
}
```

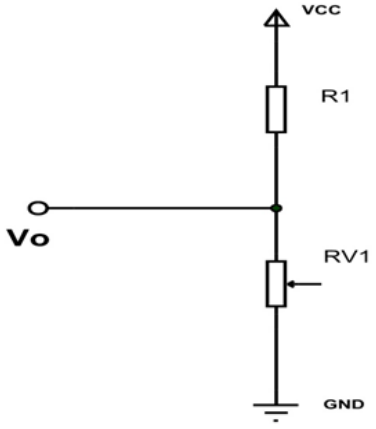
Flex Sensor

The Flex Sensor patented technology is based on resistive carbon elements. As a variable printed resistor, the Flex Sensor achieves a great form factor on a thin flexible substrate. When the substrate is
bent, the sensor produces a resistance output correlated to the bend radius, the smaller the radius, the higher the resistance value.
This property of the flex sensor is used to read the resistance that is output from the sensor when it is bent. By using the flex sensor on the index and the middle finger, we can assign the bending of the sensor on the index finger as a left click and the bending of the finger on the middle finger as the right click.
Flex sensor is a two terminal device. The Flex sensor does not have polarized terminals like diodes. So there is no positive and negative.

The diagram below shows the angle of flex on the flex sensors.

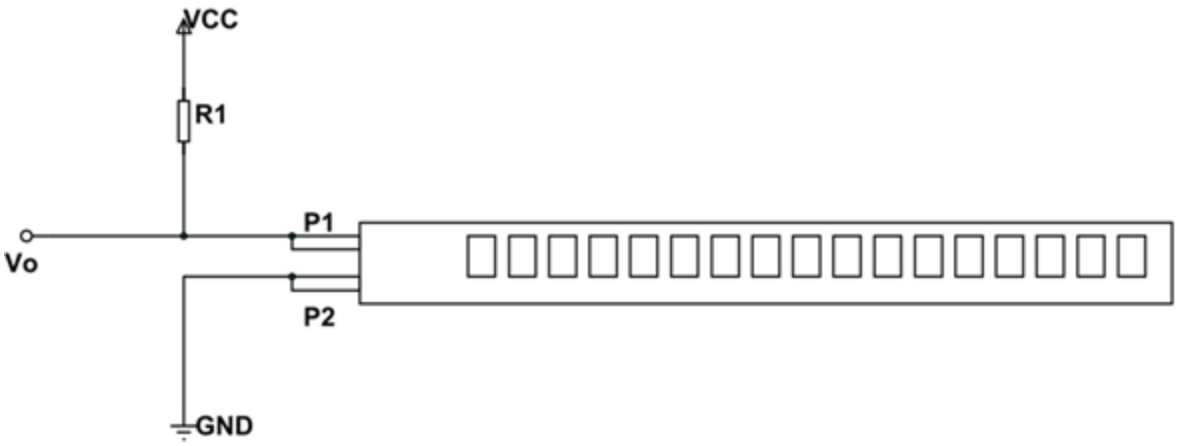
Different Flex angles

The flex sensor converts the resistance parameter into a voltage. This can be treated as a voltage divider network as shown in the diagram below.



Voltage divider network

In this resistive network we have two resistances. Vo is the voltage at the midpoint of the Voltage divider circuit and is also the output voltage. Vo is also the voltage across the variable resistance (RV1). So when the resistance value of RV1 is changed the output voltage Vo also changes. So we will have resistance change in voltage change with the voltage divider circuit.

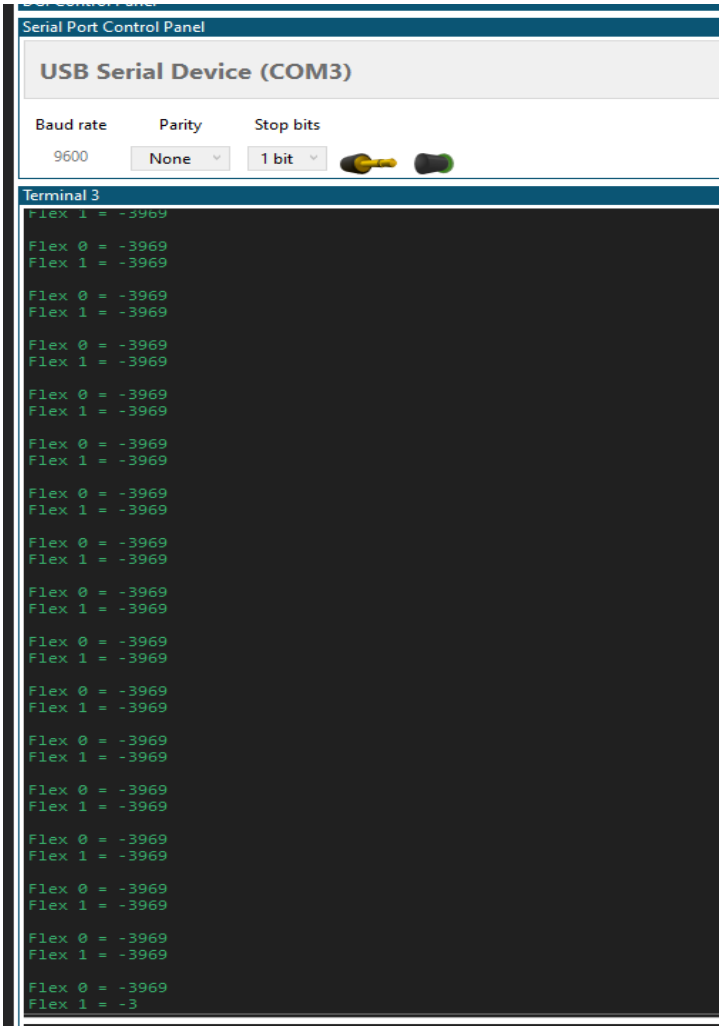


Flex Sensor Circuit Diagram

As shown in the figure above, R1 here is a constant resistance and flex sensor which acts as a variable resistance. Vo being output voltage and also the voltage across the flex sensor.

Here,
Vo = VCC (Rx / (R1+Rx)).
Rx - Flex Sensor resistance

Now, when the Flex sensor is bent, the terminal resistance increases. This increase also appears in the voltage divider circuit. With that the drop across the Flex sensor increases so does Vo. So with increase in bent Flex sensor, Vo voltage increases linearly. We convert this voltage using an ADC built into the arduino. The ADC value is mapped to the mouse.press library with a condition of left or right click according to the read value. The following is a screenshot of the flex values being read.



Flex Sensor readings

Joystick

The Arduino Leonardo consists of 12 analog inputs, labeled from A0 to A11, all of these pins can also be used as digital I/O pins. Each of these analog pins has an inbuilt ADC of resolution of 2^10 bits (so it will give 1024 values). We use two of these pins, the A0 and A1 pins to read the analog value of the Joystick X and Y.

These analog values are converted into digital values using the inbuilt ADC of the analog pins (1024 values). The digital values are read and mapped such that each of the ‘W’, ‘A’, ‘S’, ‘D’ functions can be assigned according to the deviation in the movement of the joystick respectively. These values are assigned to a key using the Keyboard.h library which enables us to interact with the computer keyboard and maps the said values to the required functionality as

defined by us. The following snippet shows the implementation and usage of the keyboard library to interact with the system.

```
void adc_call(){

 PRR0 &= ~(1<<PRADC);

  //Select Vref = AVcc
  ADMUX |= (1<<REFS0);
  ADMUX &= ~(1<<REFS1);

  //Setup ADC Clock div by 128 -> 125KHz
  ADCSRA |= (1<<ADPS0);
  ADCSRA |= (1<<ADPS1);
  ADCSRA |= (1<<ADPS2);

  //Setting Conversion Bit to 0
  ADCSRA &= ~(1<<ADIF);

  //Set to auto trigger
  ADCSRA |= (1<<ADATE);

  //Disable Input Buffer on ADC PIN
  DIDR0 |= (1<<ADC0D);

  //Enable ADC
  ADCSRA |= (1<<ADEN);
  //Start Conversion
  ADCSRA |= (1<<ADSC);

}
```

```
void loop() {

 while(1){
    while(!(ADCSRA & (1<<ADIF)));
      if(key_press == 0){
      xPosition = ADC;
      Serial.println("Value of X is = ");
      Serial.print(xPosition);
      Serial.println(" ");
      ADMUX = 0x46; //read from A2
      _delay_ms(1000);
      key_press = 1;
      ADCSRA |= (1<<ADEN);
      ADCSRA |= (1<<ADSC);
    }
     if(key_press == 1){
      yPosition = ADC;
      Serial.println("Value of Y is = ");
      Serial.print(yPosition);
      Serial.println(" ");
      ADMUX = 0x45; //read from A2
      _delay_ms(1000);
      key_press = 0;
      ADCSRA |= (1<<ADEN);
      ADCSRA |= (1<<ADSC);

  }
```

Snapshots of the Joystick Code

```
if (xPosition < 480 && (yPosition>100 && yPosition<900))
 {
  Keyboard.releaseAll();
  Keyboard.press('a'); //for a
  Keyboard.releaseAll();
 }

 if (xPosition > 530 && (yPosition>100 && yPosition<900))
 {
  Keyboard.releaseAll();
  Keyboard.press('d'); //for d
  Keyboard.releaseAll();
 }

if (yPosition > 530 && (xPosition>230 && xPosition< 760))
{
  Keyboard.releaseAll();
  Keyboard.press('w'); //for w
  Keyboard.releaseAll();
}

if (yPosition < 480 && (xPosition>230 && xPosition< 760))
{
  Keyboard.releaseAll();
  Keyboard.press('s'); //for s
  Keyboard.releaseAll();
}
```

Keyboard mapping of joystick input

Push Buttons

Push buttons are very simple to integrate using the digital pins of the Arduino Leonardo. It is integrated using the method of Polling where the button pin is continuously read for a digital input.

As soon as there is an input, left or right button click, the microcontroller is triggered. The Mouse.press() function from the Mouse.h library enables us to interact with the computer facilitating the left and right clicks. Mouse_left is passed as an argument to the Mouse.press() function which enables the left click functionality.

The same is implemented for a second button to facilitate the right click. The following snippet shows the simple initialization of the registers of the arduino for the input and output of a button press.

```
void Initialize()
{
        DDRB &= ~(1 << DDB0); //PB0 Left click input
        DDRB &= ~(1 << DDB1); //PB1 Right click input
        DDRB |= (1 << DDB5);  //PB5 Mouse click left
        DDRB |= (1 << DDB6);  //PB6 Mouse click right

        //timer prescaling by 8
        TCCR1B &= ~(1 << CS10);
        TCCR1B |= (1 << CS11);
        TCCR1B &= ~(1 << CS12);

        //setting up the timer
        TCCR1A &= ~(1 << WGM10);
        TCCR1A &= ~(1 << WGM11);
        TCCR1B &= ~(1 << WGM12);
        TCCR1B &= ~(1 << WGM13);

        //Clear input capture
        TIFR1 |= (1 << ICF1);

        TCCR1B &= ~(1 << ICES1);

        sei();
}
```

The following code snippet shows the reading of the ports for the for the button input.

```
void loop()
{

    while (1)
    {
    while(!(TIFR1 & (1 << ICF1)));
    PORTB |= (1 << PB5);
    TIFR1 |= (1 << ICF1);
    TCCR1B |= (1 << ICES1);

    while(!(TIFR1 & (1 << ICF1)));
    PORTB &= ~(1 << PB5);
    TIFR1 |= (1 << ICF1);
    TCCR1B &= ~(1 << ICES1);

    while(!(TIFR1 & (1 << ICF1)));
    PORTB |= (1 << PB6);
```

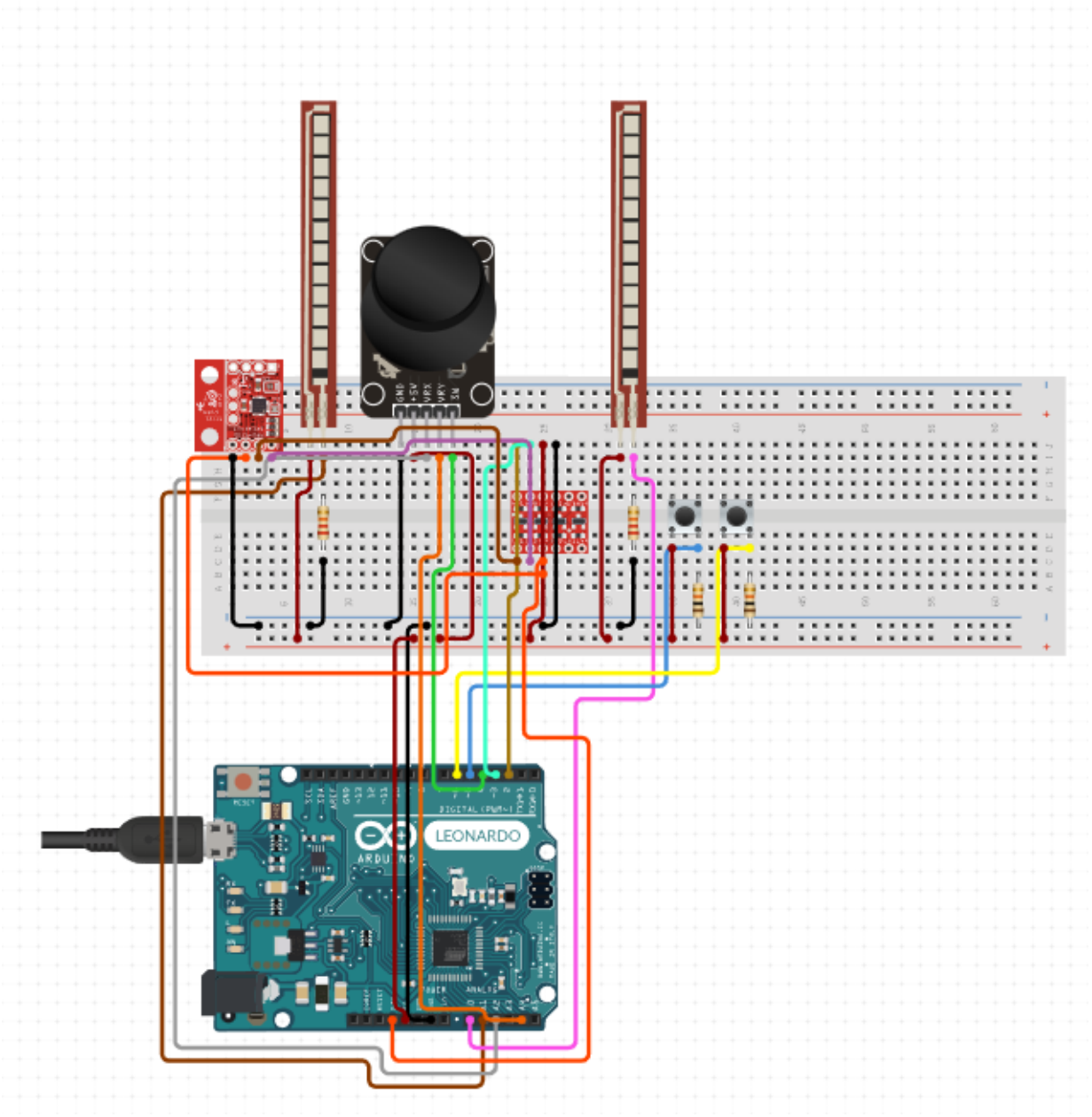
```
    TIFR1 |= (1 << ICF1);
    TCCR1B |= (1 << ICES1);

    while(!(TIFR1 & (1 << ICF1)));
    PORTB &= ~(1 << PB6);
    TIFR1 |= (1 << ICF1);
    TCCR1B &= ~(1 << ICES1);
    }
}
```

The code snippet above shows the reading of pins 5 and 6 for the button input.

Circuit Diagram

The following circuit diagram represents the connections between the various sensors and the Arduino Leonardo board. This is only the graphical representation of the system. The system is integrated on a glove to make it aesthetically presentable. The circuit diagram shows the Joystick, flex sensors, MPU6050 gyroscope and the push buttons connected to the Arduino.



Circuit Diagram

Results

The final product is capable of the functionality as described in section B of Goals. The Glomo is a gesture controlled, joystick integrated device capable of solving the problems of a standard mouse with an extra functionality for gaming.

Overall, the results of our project met our expectations outlined in our project proposal. Our final prototype is able to effectively control mouse movement and clicking with user hand gestures. We are pleased with the integration of both our final hardware and software.

The final prototype created from our 4 week project serves as an additional proof-of-concept to similar existing projects. Because of this, we believe that this design has much room to grow. Primarily, we believe that a smaller wearable unit than a breadboard based design would be more practical for everyday use. Our project is built with a small budget in mind and with easily usable, but not necessarily the smallest, parts. With more time, a custom PCB could be used to simplify the physical construction. Additionally, higher resolution cursor movement could be attractive for commercialization.

Conclusion

The 4 week project gave us an insight of the various sensors which could be used in daily applications easing human lives. We learnt how multiple electronic sensors can interact to bring together a whole system essential for communicating with the digital world. We also explored multiple ways of tackling the same problem which gave us a deeper understanding of the protocols and the concepts used.

We implemented the device using an analog read of the pins and ADXL335 sensor initially and then improved our design by shifting the design to a more reliable and precise sensor, MPU6050. We encountered a couple of challenges with the faulty hardware and spent a lot of time debugging for errors when all we had to do was replace the sensor. We initially wrote parts of the code for different sensors and tested them out using hardware methods like using a simple LED to test if the device is working. We encountered issues while integrating all the parts of the code together. It was quite challenging to solve the mapping issue as we have to manually calibrate the motion of the mouse. We could have made the device to operate wirelessly using a HC05 bluetooth module if given some more time.

Overall, it was a good learning experience to use the knowledge of embedded systems design on an application that we created from scratch. The next step in this project is to make the device wireless, more compatible by building a custom PCB for integration of the sensors.

References

https://lastminuteengineers.com/mpu6050-accel-gyro-arduino-tutorial/
https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/f2012/as986_hl525/as986_hl525/index.htm
https://microcontrollerslab.com/flex-sensor-arduino-tutorial/
https://create.arduino.cc/projecthub/albani-carlo-amaducci-lorenzo-ferraiuolo-lorenzo/gesture-glove-e64871
https://github.com/arduino-libraries/Mouse
https://github.com/arduino-libraries/Keyboard
https://create.arduino.cc/projecthub/MissionCritical/mpu-6050-tutorial-how-to-program-mpu-6050-with-arduino-aee39a
https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf
https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

Appendix A

External Libraries used from the arduino library manager:

Mouse.h
Mouse.cpp
Keyboard.h
Keyboard.cpp

Arduino IDE was used as some of the device libraries like Mouse and Keyboard libraries are not compatible with the Atmel studio IDE by Microchip. These libraries have multiple dependencies and access the driver of the mouse and the keyboard. Writing these libraries on our own would require a huge amount of time.

Although arduino also provides other libraries such as the I2C and the ADC, we have put in the effort to implement these libraries on our own and not use the inbuilt libraries.

I, Sumanth Marakini Shivshankar, certify that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this final exercise.

I, Arjun Makwana, certify that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this final exercise.