Report for Control band

Conception:

The changes that occur due to plural extension or flexion of fingers are,

1. Skin Stretching at the fore arm.
2. Shape changes at the forearm muscles.
3. Blood pressure change.
4. Minute temperature change within the arm.
5. Stiffness change of the muscle.
6. Movement of the tendons inside the muscles.

The ideas could broadly be classified as capturing the signals externally or capturing the nerve signals directly.

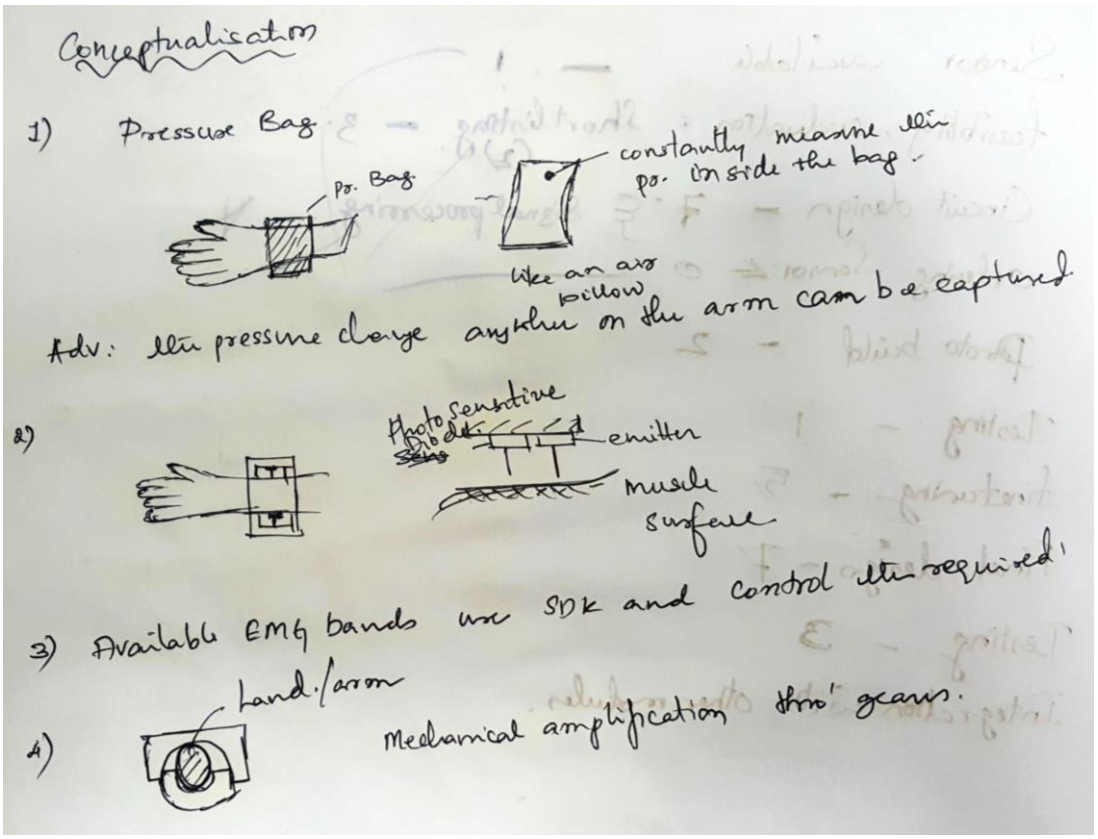
Capturing signals externally.

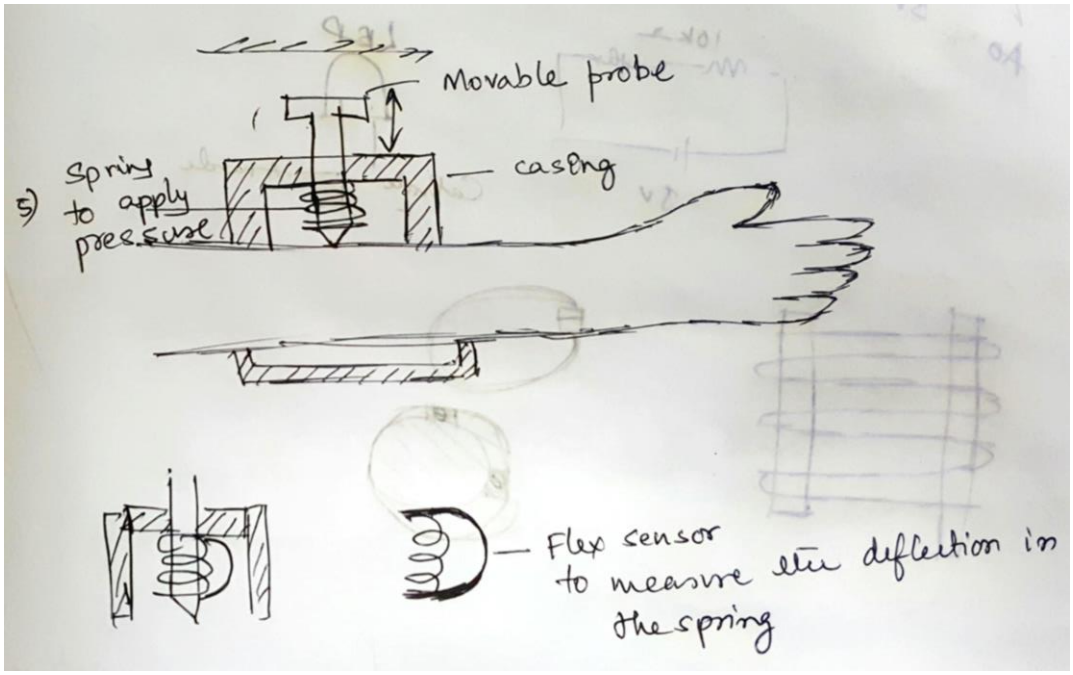
1. Pressure bag around the arm and measure the pressure change dynamically.
2. Laser infrared reading of the muscle change
3. Already available EMG bands – MyoBand
4. A band with one end attached to the upper arm and the other half to the lower arm and measure the change between them
5. Capacitive ring on the hand and envelope around and measure the capacitance change between the ring and the external enclosure
6. Conductive rubber cord stretch sensor worn around the fore arm.
7. Flex sensors used to capture the change in the shape of the muscle.
8. Proximity sensor and capacitive displacement sensor
9. Mechanical switching , activating an actual switch
10. Amplify the minute movements while the muscle expands by using gears and track the number of rotations in an encoder.
11. A grill of strain gauges on the forearm and supported in an enclosure
12. Using an LVDT sensor placed on the skin of the forearm and measure the movement in the sensor

Following a series of brainstorming we decided upon two easily available sensors which are inexpensive and also could be used with less tweaking

1. Stretch Sensor to detect the circumferential measurement of the forearm.
2. Flex sensor to measure the change in shape of the forearm.
3. Force Sensitive resistors (FSR) to be mounted at points to measure the local change in the forearm while in extension or flexion.

Drawings for few of the initial concepts.





List of sensors that could be used to capture the above said changes in the forearm.

Laser interferometers, Ultrasonic sensors, LVDT linear Measurement, Flex Sensors, Elastic sensors, Piezoelectric pressure sensor.

Trail with the Flex sensor.

The flex sensor is placed under the forearm directly onto the skin. And was interfaced to computer through arduino to measure the change in the readings of the sensor while the hand was in flexion or extension. Though the results were substantial, it took a considerable time for the sensor to comeback to original state i.e the hysteresis of the sensor was high.

Hence the idea of use of flex sensor was shelved to be explored for further results.

Trial with FSR.

Refer Arjun's report.

Trail with Stretch sensor (Conductive Rubber).

The initial test with the rubber worn over the forearm directly didn't give the necessary results as the rubber would dig into the skin, given the soft nature of the skin. Also the actual expansion wasn't captured in the sensor due to this. So there was a requirement for rethinking the design with the same sensor as the sensor reacted to the changes in the forearm but only not as expected.

The idea and design of Pads.

The problem of the conductive rubber digging into the skin when the hand is in flexion or extension was reduced when we introduced 'Pads' as described in the below figure. The integration of pads yielded positive results when tested with initial AdHoc prototype using foam board. The pad design itself went through a number of iterations and the transformation is as described below. The initial pad size was 25mm X 18mm. With screws too hold in place the conductive rubber. In the second iteration a snap fit parts were designed with dimensions 18 X 10 mm. This was done to make the band more compact and also to reduce the number of parts by introducing snap fit within the pads.

Phase 1: Initial prototype in foam board

Figure depicting Initial prototype in foam board

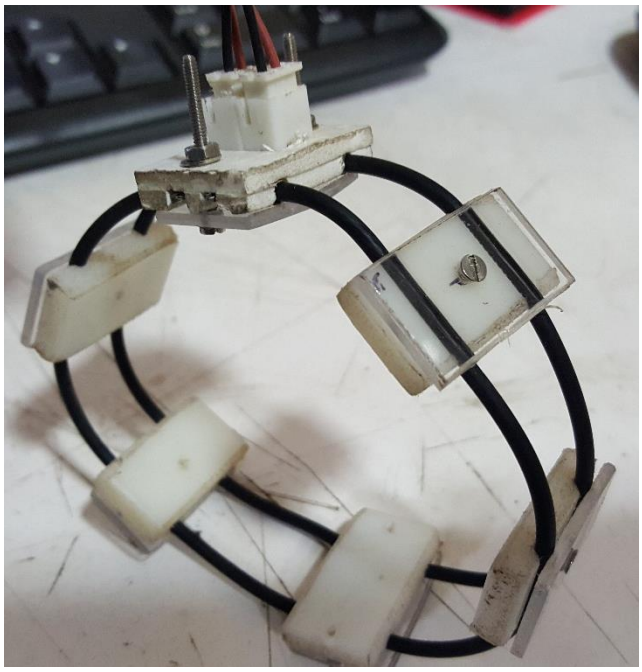
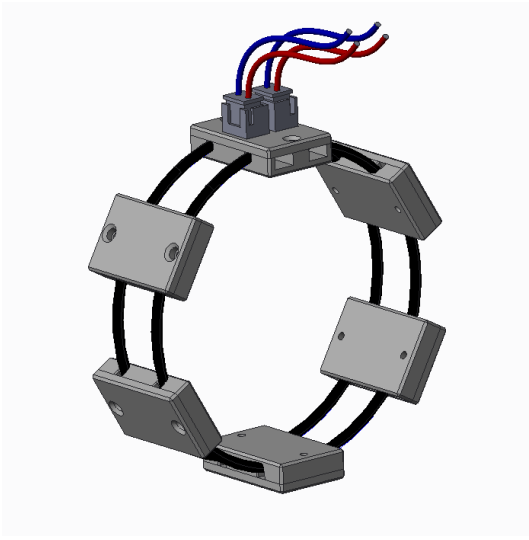


Figure depicting Initial prototype with sensors split.

Phase 2: Prototype in nylon with two conductive sensors



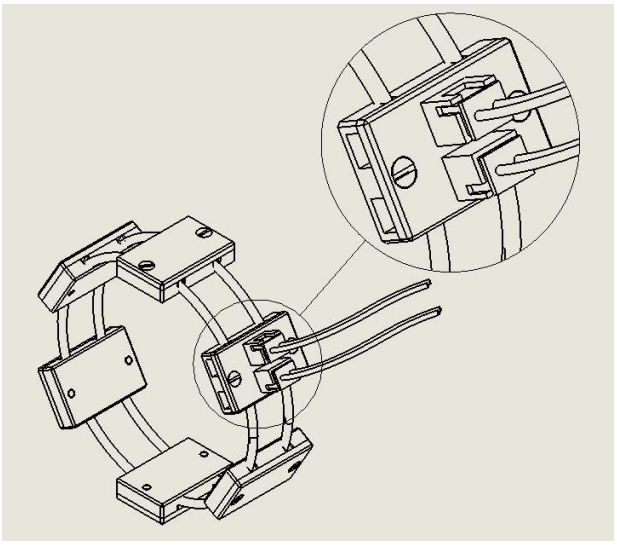


Figure Showing External connections

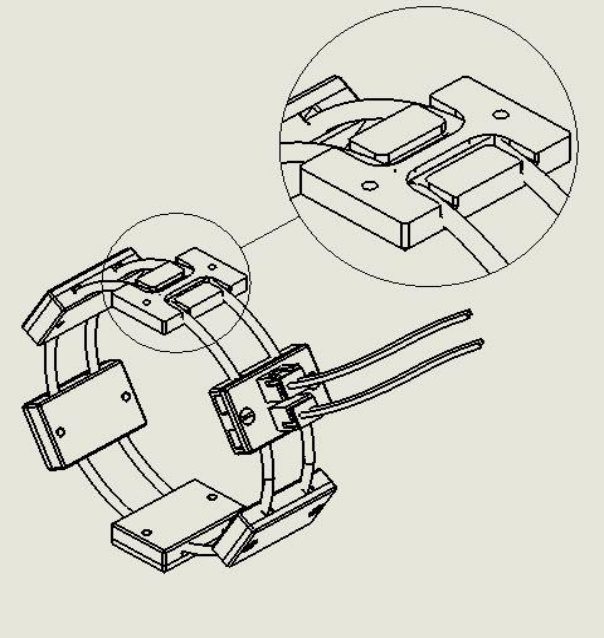


Figure depicting Pads with U looped sensors in place to connect two segment of conductive sensors.

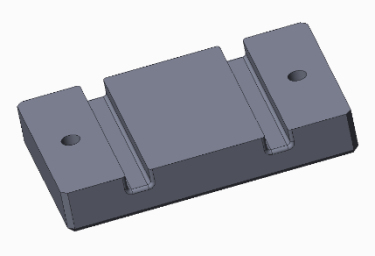
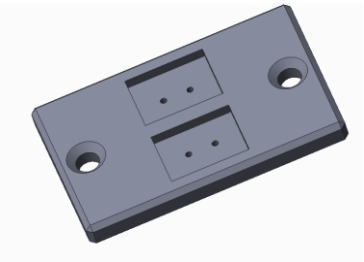
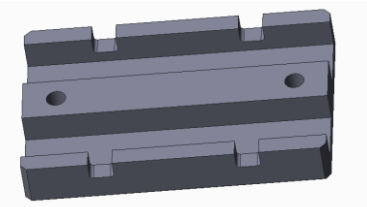


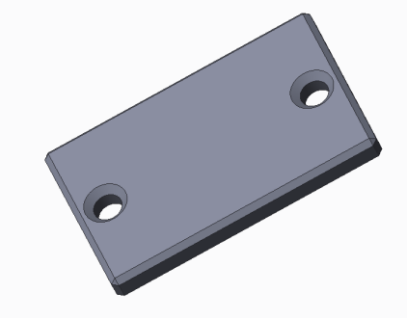
Figure 1general pad lower



Pad for external connections having a seat for plugs



Lower Pad for external connections where four ends of the conductive chord meet



Upper general pad

Two loops of the conductive sensor are in place on this pad

All the pads were redesigned for snap fitting so as to do away with the screws and reduce the weight. It also improved the design aesthetically and reduced complexity in assembling the parts. The dimensions of the pads were also reduced to make the design more compact.

The Figures below describe the CAD design and final model.

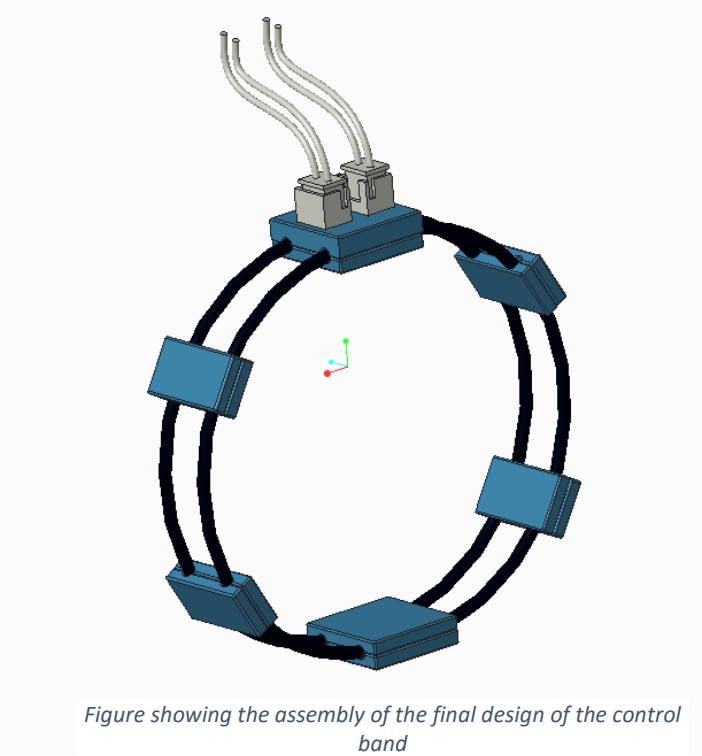


Figure showing the assembly of the final design of the control band

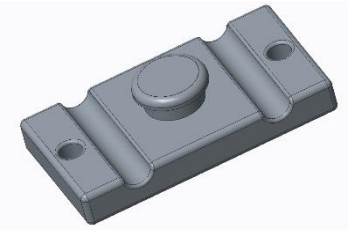


Figure showing Upper pad with snap fit projection

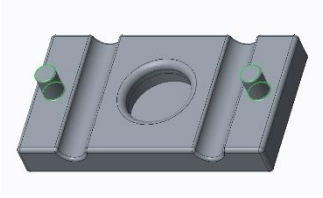


Figure showing Lower pad with snap fit depression

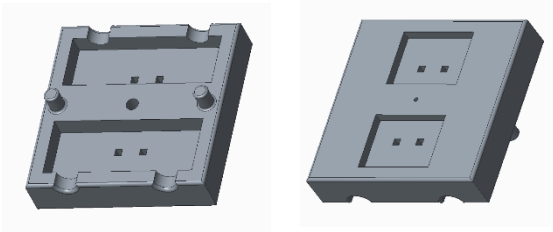


Figure showing pad for external connection with seating for plugs. From left Top and Bottom view

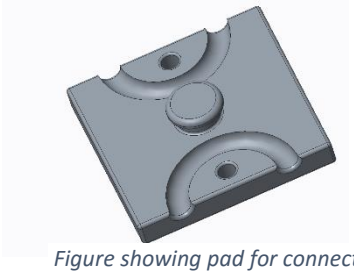
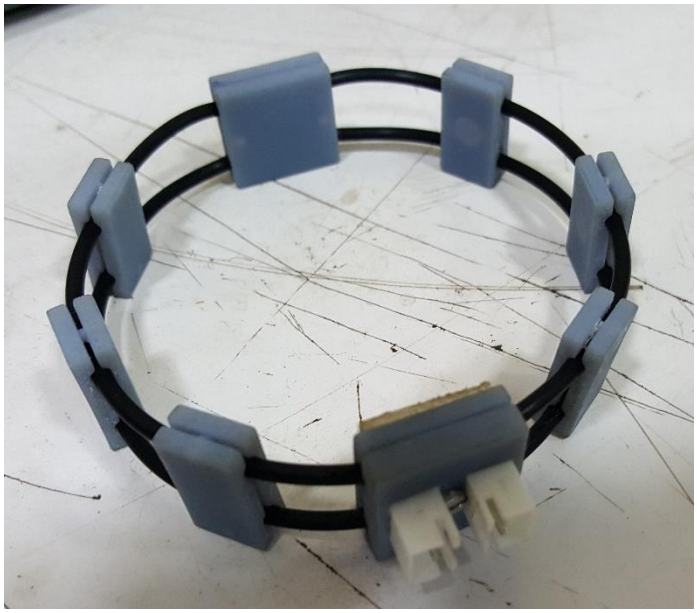
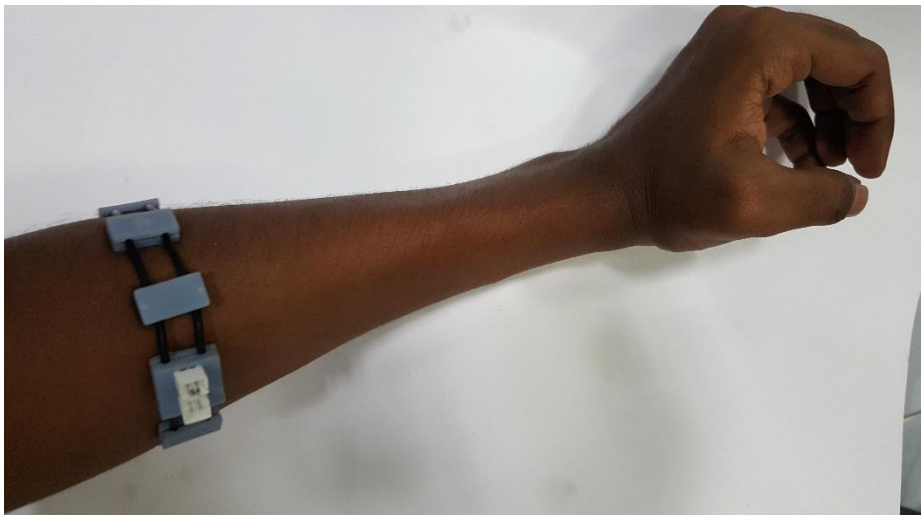


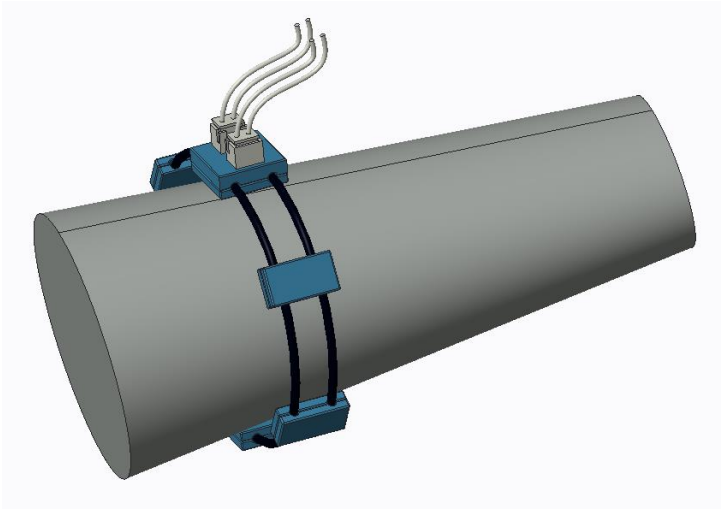
Figure showing pad for connecting two segments of the conductive rubber.



Actual prototype of the final design



Person wearing the band to control the Prosthetic hand



The electronics.

At first only the Stretch sensor was connected in a voltage divider circuit and interfaced with arduino through analog input A0 pin to know if the resistance change in the stretch sensor was considerable enough. Given the positive results we went ahead with zeroing on the stretch sensor.

Phase 1: Reading the resistance values from the sensor.

Firstly the Arduino IDE was set up and with the help of maker plot we were able to plot the values of the resistance on the scale 0-1023 for different configurations of the hand, Flexion and extension. The graph plotted had a lot of noise but resulted in a very desirable curve i.e a peak occurred when in flexion and the graph subsided to lower limits when in extension. It was concluded from this phase that the values needed to be reduced of noise.

On the similar lines, in the Program for the control band, the filters, Exponential Moving Average and Median filter were adopted. The exponential moving average gave better results than the median filter (median filter is generally used in image processing to estimate the nearby values based on the median of the stream of values). Through the adjustment of the alpha value in the EMA we arrived at the alpha as 0.9 such that the noise was reduced but at an expense of the response.

Phase 2: Categorising the curve.

The idea of control through 'Clicks' was first pondered on. A 'click' is defined by a flexion and extension subsequently in a short interval (about 0.5 sec). The logic that was thought about

a. A single click would mean to run the motor forward till it approaches the object.
b. A double click would mean to run the motor reverse entirely.

This was tried through printing out statements but this didn't suffice our needs of the device being intuitive rather needed the training to the wearer to remember how to control the prosthetic hand

Phase 3: Idea of intuitive control of the hand.

Since the curve was now filtered of the spikes and random values this resulted in a curve that was predictable enough and smooth. The curve raised as the fingers went through flexion and fell when the fingers were in extension.

Hence, the idea of intuitive control was conceived based on the gradient of the slope. Since the curve raised w.r.t the extent of flexion and fell with the extent of extension, a positive gradient meant the fingers were closing and a negative gradient meant the fingers were opening and a zero gradient ideally meant the fingers were at rest.

Hence algorithm was built to calculate the slope at every 200ms interval with 3 averages in between the interval to make the value more resilient to random errors.

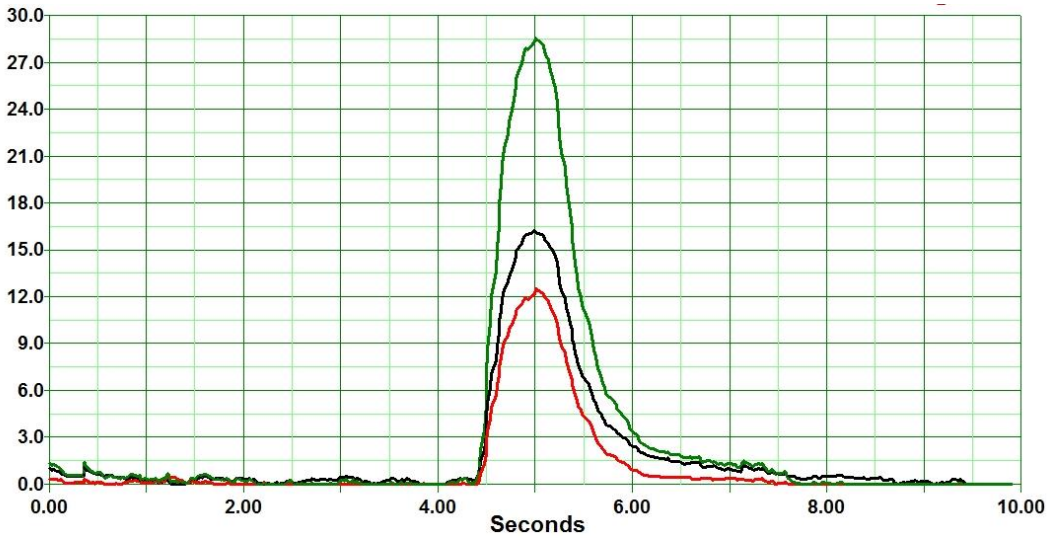


Figure shows the plot of the sensor values wrt time. Channel A- Black, Channel B - Red, A+B is Green

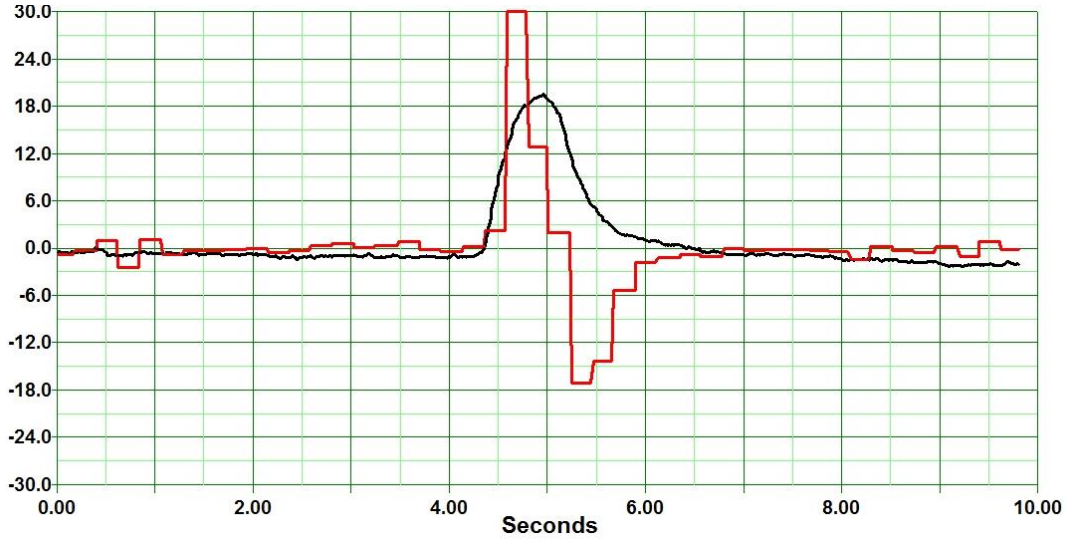


Figure shows the plot of the derivative of the sensor values

Phase 4: Integration of motor.

The motor was controlled by an IC L293D, which could reverse the polarity of the motor based on the digital input from the arduino. This allowed the bidirectional movement of the motor based on the sensor values. Now the motor to go to the desired position was defined by the sensor value and the direction is decided by the gradient of the sensor value.

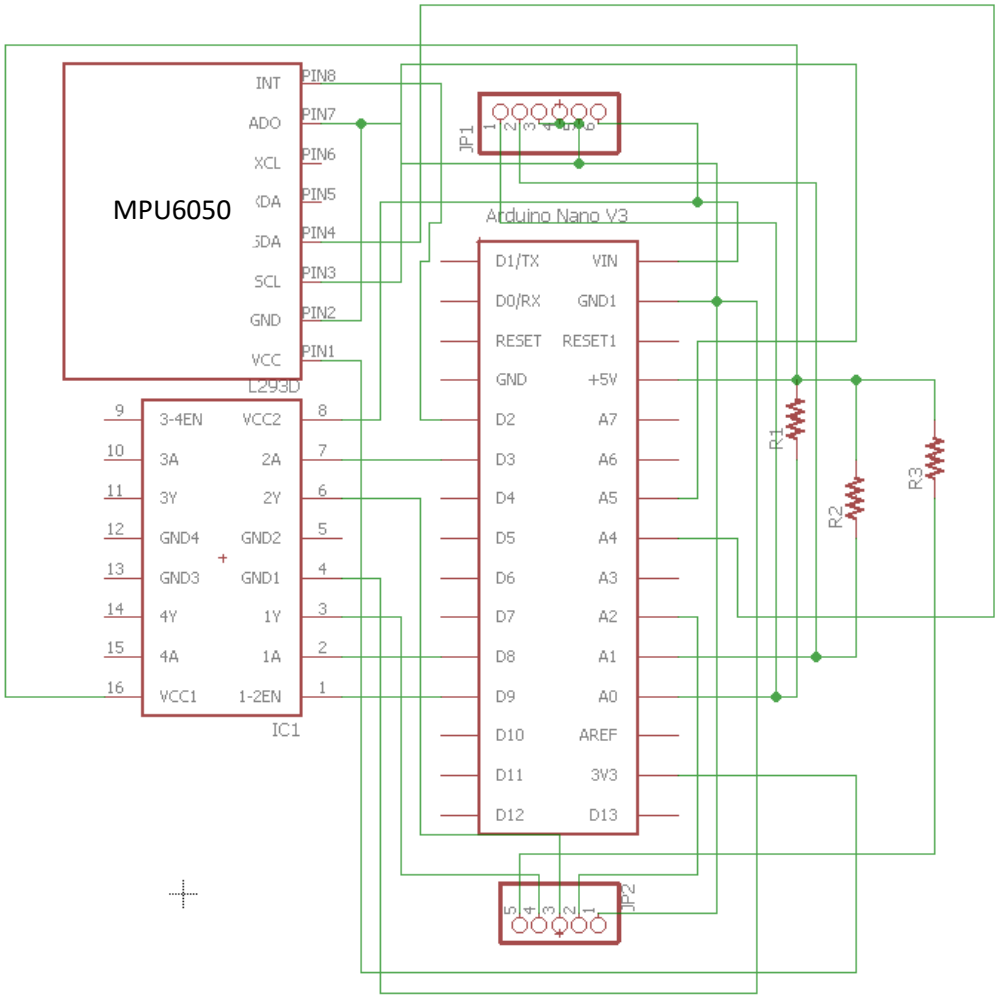
Phase 5: Integration of Gyro (MPU6050)

As the initial tests were favourable enough, we decided to make the band more reliable and eliminate false signalling. One of the main causes of false signalling was due to the reaction of the band to the pronation and supination of the arm. As the pronation and supination caused the muscle underneath the band to stiffen and relax. To eliminate this a Gyro-

Accelerometer sensor MPU6050 was adopted. The algorithm was tweaked such that as long as the hand of the wearer was moving, the motor would stop to take signals till the hand came to a rest. Thus eliminating the false signalling.

The Circuit

The circuit is as described below with the schematic.



IC1 is the Motor control IC L293D, R1 is of the value 270 Ohms, R2 and R3 of 1000 Ohms. JP1 and JP2 are the L –Connector pins with 5 and 6 pins respectively. The MPU6050 unit is the motion processing unit comprising the Gyro and Accelerometer sensors.

JP1 is connected to Firgelli motor with potential meter feedback. The pin is to be connected such that the black flat portion is in the upper side.

JP1 Pin Description:

1 – One of the terminal of Stretch Sensor A.
2 – One of the terminal of Stretch Sensor B.
3 – Other terminal of Stretch Sensor A.

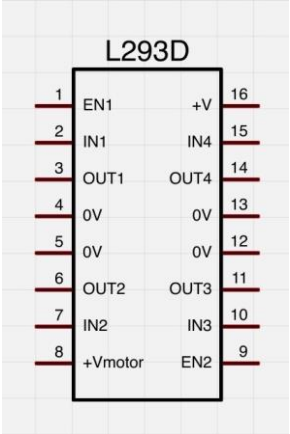
4 – Other terminal of Stretch Sensor B.
5 – Black (-ve) of the supply (6V or 12V according to the motor).
6 – Red (+ve) of the supply.

Six Pin JP2 is connected to Stretch Sensor and Power supply as described below.

JP2 Pin Description:

1 – To be connected to –ve potentiometer rail.
2 – To be connected to potentiometer brush.
3 – To Output 2 (-ve of the motor)
4 – To Output 1 (+ve of the motor)
5 – To +ve potentiometer rail.

IC1 i.e L293D Connection Instructions.



In the picture above EN1 is the enable pin which is used to control the speed of the motor. IN1 is the input pin 1 Arduino pin D8 which activates the OUT1 connected to one of the terminal of the motor. IN2 is the input pin 2 connected to D3 and activates OUT2 connected to one of the motor terminals. +Vmotor is the supply voltage (12V or 6V) required to run the motor. +V is the power required to turn on the IC i.e 5V.

The Programming

To enable easier interfacing between computer and other electronic components Arduino was used and the program/ algorithm was developed on Arduino IDE. The graphs were initially plotted on MakerPlot to understand the nature of the curve as discussed in the above sections.

```
#include <Chrono.h>
#include <Wire.h>
#include <I2Cdev.h>
#include <MPU6050.h>
```

Header files required:

Chrono for time keeping,

Wire , I2Cdev and MPU6050 for interfacing with gyro

```
// Sensor pin - GND
// Sensor pin - Analog In 0, with 10K resistor to +5V
//////variables for gyro reading//////////////////////////////////////////////////////////////////////
MPU6050 mpu;
double x_filtered=0 , y_filtered=0 , z_filtered =0 ,alpha_gyro = 0.99 ;
double sumX=0 ,sumY=0,sumZ=0 , avgX=0,avgY=0 ,avgZ=0;
int16_t ax, ay, az;
int16_t gx, gy, gz;
float x_acc, gyro;
int gyro_limits=800; // tweak for change in sensitivity of the gyro
/////////variables for motor and sensor//////////////////////////////////////////////////////////////////
Chrono myChrono, myChrono2, myChrono3 , chronoCalib;
int enablePin = 9;
int in1Pin = 8;
int in2Pin = 3;
int ledPin = 13;
```

mpu object of the class MPU6050. Variables used in Gyro control

Chrono calib defines the time for calibration required. Enable pin is for the motor to switch on and also for the speed control

```
int SensorPin1 = A0 , SensorPin2 = A1 ; // pins for analog input from the band to take in the resistance changes

int positionPin = A2; //motor position pin from the potentiometer readings

int i = 0, buttonState = 0, motorPin1 = 8, motorPin2 = 7; //output for the motor to run

double sum1 = 0.0, avg1 = 0, sum2 = 0, avg2 = 0; // used for sensor averaging and initial setting zero

double a = 0, b = 0 , c = 0, d = 0; // used for motor not reaching issue

int calibTime = 5000; // calibration time to be allowed in milliseconds


int minPosn = 350, maxPosn = 800; //set variables for motor positions min and max

double filtered1 = 0, filtered2 = 0, alpha = 0.1, filteredMapped, sensorMaximum, sensorMinimum , startTime = 0, Th = 8 ; // initialising filtered values from the sensor setting the alpha value or the level of filtering required

double mapPotandSen = 0;

bool track = true, track2 = true , flag = true;

////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////

void setup()

{

  // initialize serial communications

  Serial.begin(9600);

  //mpu.initialize();


  pinMode(in1Pin, OUTPUT);

  pinMode(in2Pin, OUTPUT);

  pinMode(enablePin, OUTPUT);

  pinMode(SensorPin1 , INPUT);

  pinMode(SensorPin2 , INPUT);

  pinMode(positionPin , INPUT);

  pinMode(ledPin , OUTPUT);
```

Declaring the necessary pins input or output

```
  digitalWrite (enablePin, HIGH);
  centerMotor (400); // start the motor at 400/1000 position
}

void loop()
{
   // read the voltage from the voltage divider (sensor plus resistor)
  //digitalWrite(enablePin, HIGH);
  mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz); // function from the MPU6050 library
  int sensor1 = analogRead(SensorPin1);
  int sensor2 = analogRead(SensorPin2);

  //int sensor = sensor1+sensor2;

  if (i < 1000)
  {
    sum1 = sum1 + sensor1;
    sum2 = sum2 + sensor2;
    sumX = sumX + gx ;
    sumY = sumY + gy ;
    sumZ = sumZ + gz ;
    i++;
  }
```

Initial reading of the 1000 stretch sensor values and summing the gyro sensor values.

```
else if (i == 1000)
 {
  avg1 = sum1 / 1000.0;
  avg2 = sum2 / 1000.0;
  avgX = sumX/i;
  avgY = sumY/i;
  avgZ = sumZ/i;
  delay(2000);
  digitalWrite(ledPin , HIGH); // to indicate the offsetting is done
  delay (1000);
  digitalWrite(ledPin , LOW); // to indicate the offsetting is done

  i = 1005;
  chronoCalib.restart();
 }
```

Averaging the read 1000 values and offsetting the zero in both the gyro and the stretch sensor.

```
 else if ( i == 1005)
 {
////////////filtered values of   Gyro//////////////////////////////
  x_filtered = alpha_gyro*x_filtered + (1-alpha_gyro)*(gx-avgX) ;
  y_filtered = alpha_gyro*y_filtered + (1-alpha_gyro)*(gy-avgY) ;
  z_filtered = alpha_gyro*z_filtered + (1-alpha_gyro)*(gz-avgZ) ;
```

Stream of gyo values after EMA filter

```
//////////////////////filtered values of the sensors//////////////
  filtered1 = (1 - alpha) * filtered1 + alpha * (sensor1 - avg1);
  filtered2 = (1 - alpha) * filtered2 + alpha * (sensor2 - avg2);
```

Stream of stretch sensor values after EMA filter

```
////////////// calibration to set maximum and minimum/////////////////
  // led every 200ms to indicate the band is calibrating
```

```
If (chronoCalib.elapsed() < calibTime)
{
 int ledInterval = chronoCalib.elapsed();
 if ( (ledInterval % 200) > 0 && (ledInterval % 200)  < 50)
 {
  digitalWrite(ledPin , HIGH);
 }
 else
 {
  digitalWrite(ledPin , LOW);
 }
```

To blink the led every 200ms to indicate that the calibration is being done.

```
    sensorMaximum = findMax (filtered1+filtered2);
    sensorMinimum = findMin (filtered1+filtered2);
  }
else
   {
    gyroSensor(); // gyro sensor activation
mapPotandSen =
map (filtered1+filtered2 , sensorMinimum, sensorMaximum, minPosn, maxPosn);//
mapping the values to the max and min position of the motor.
```

```
 if (mapPotandSen > maxPosn)
    {
      mapPotandSen = maxPosn;
    }
   if (mapPotandSen <minPosn)
    {
      mapPotandSen= minPosn;}
```

To make the motors commanded position to be within the safe limits of operation.

```
    categoriseSlopeRunMotor(slopeSensor(filtered1+filtered2), -5 , 5 , 25, 25); // run the motor to the position of commanded , this is the main function for the entire code.

}

}


////////////if installed an external button uncomment the below code.//////////////////////////////////////////

  //  if (buttonState == HIGH)

  //  {

  //    i = 0;

  //    sum1 = 0;

  //    sum2 = 0;

  //    avg1 = 0;

  //    avg2 = 0;

  //    flag= true;

  //    centerMotor(410);

  //

  //

  //  }

// }


}
```

//////////////////////////functionsused//

```
double findMax( double x)  // to find the maximum of sensor value so that the max limits can be fixed
{
  static double maxi = x;


  if (x > maxi)
  {
    maxi = x;
   }


   return maxi;
 }
```

Function to find the maximum of stream of values

```
double findMin( double x)     // to find the maximum of sensor value so that the min limits can be fixed
{
  static double mini = x;
  if (x< mini)
  {
    x= mini;
  }
  return mini;
}
```

Function to find the minimum of stream of values

```
void updateSensorValue( void )    // in case you need a fresh sensor value intermittently
{
  //rawNew = (analogRead(SensorPin1) - avg);
  //filtered = (1 - alpha) * filtered + alpha * rawNew;
}
```

```
double slopeSensor( double value)   // used to find the gradient wrt time of the sensor , and the argument to be passed is the filtered value of the sensor
{
  //static double value1 = 0;
  //static double slope = 0;
  static double slope = 0, sum1 = 0, sum2 = 0, sum3 = 0, initialValue = 0;
  static int count1 = 0, count2 = 0, count3 = 0;
  if (flag == true)
  {
    //initialValue = value;
    myChrono.restart();
    flag = false;
  }

  if (myChrono.elapsed() <= 67)
  {
    sum1 = value + sum1;
    count1++;
  }

  if (myChrono.elapsed() > 67 && myChrono.elapsed() <= 134)
  {
    sum2 = value + sum2;
    count2++;
```

Function to find the slope of the stream of values. The slope is found every 67 ms and averaged for 3 findings to make the value more reliable

In continuation of the above function

```
  }


  if (myChrono.elapsed() > 134 && myChrono.elapsed() <= 200)
  {
    sum3 = value + sum3;
    count3++;
  }



  if ( myChrono.elapsed() > 200 &&  myChrono.elapsed() <= 225)
  {
    slope = ((((sum2 / count2) - (sum1 / count1)) / 0.134) + (((sum3 / count3) - (sum2 / count2)) / 0.134)) / 2.0 ;


    myChrono.stop();
    sum1 = 0, sum2 = 0, sum3 = 0, count1 = 0, count2 = 0, count3 = 0; // initialise all the variables
    flag = true; // set flag so that new slope value can be calculated.



  }
 if (myChrono.elapsed() > 225)   // if while execution the time exceeds 225 seconds the flag is set to true so that a new gradient is calculated
 {
  flag = true;
 }
  return slope;
}
```

```
//////////////////function for categorisation of the values of gradient
////////////////////////////////////////////////

// x is the slope value , negThresh is the negative threshold i.e the value of slope below which the motor is to run reverse , Motor relief is the relief for min and max position

void categoriseSlopeRunMotor( double x , double negThresh , double posThresh , int negMotRelief , int posMotRelief )

{

 int pot2 = analogRead(positionPin);

  if (pot2 > (maxPosn + posMotRelief) || pot2 < (minPosn - negMotRelief))

  {

    return;

  }


  if (x <= posThresh && x >= negThresh) // motor is inactive when the slope value is in between the threshold limits

  {

      digitalWrite(in1Pin ,LOW);

      digitalWrite(in2Pin ,LOW);

    //Serial.println(" in the halt the motor");

  }

  if ((x > posThresh) ) // to run the motor forward

  {

      pot2 = analogRead(positionPin);

      myChrono2.restart();

      while( pot2 < (mapPotandSen) && (pot2 <(maxPosn+posMotRelief) && pot2   >(minPosn-negMotRelief)))

      {

      digitalWrite(in1Pin ,HIGH);

      digitalWrite(in2Pin, LOW);

      pot2 = analogRead(positionPin);
```

The slope values within which the motor stays inactive or the fingers are in constant position

The slope values within which the motor runs forward or when the fingers are in flexion

```
      if (myChrono2.elapsed()>=200 && myChrono2.elapsed()<=250) // for breaking the loop if the motor is struck and not able to reach the commanded position
      {
        b=a;
        a= pot2;
        myChrono2.restart();
        if ( a>=b-5 && a <= b+5)
        {
        break;
        }
      }

      }
      digitalWrite(in1Pin , LOW);// to ensure the motor halts after the loop
      digitalWrite (in2Pin, LOW);
      if (myChrono.elapsed() > 225) // if the time has elapsed more than 225 seconds while executing the above loop
      {
        flag = true;
      }
```

In continuation of the above code snippet

```
 }
//Serial.println("forward");
 if (x <= negThresh )
{
pot2 = analogRead(positionPin);
myChrono3.restart();
    while(pot2 > (mapPotandSen) && (pot2 <(maxPosn+posMotRelief) && pot2 >(minPosn-negMotRelief)))
    {
```

```
      digitalWrite(in2Pin ,HIGH);
      digitalWrite(in1Pin, LOW);
      pot2 = analogRead(positionPin);
     if (myChrono3.elapsed()>=200 && myChrono3.elapsed()<=250)
     {
       c=d;
       c= pot2;
       myChrono3.restart();
       if ( c>=d-5 && c <= d+5)
       {
       break;
       }
     }

    }
    digitalWrite(in1Pin , LOW);
    digitalWrite(in2Pin, LOW);
     if (myChrono.elapsed() > 225)
     {
       flag = true;
     }
}
//    Serial.print(pot2);
//    Serial.print(" , in reverse , ");
//    Serial.println(mapPotandSen);
```

```
//Serial.println(" reverse");
}
```

```
void centerMotor(int value) // to command initial position of the motor to where it should start from.
{
 int pot = analogRead(positionPin);
 while (pot < (value - 10))
 {
  digitalWrite(in1Pin , HIGH);
  digitalWrite(in2Pin , LOW);
  pot = analogRead(positionPin);
 }
 while (pot > (value + 10))
 {
  digitalWrite(in2Pin , HIGH);
  digitalWrite(in1Pin , LOW);
  pot = analogRead(positionPin);
 }

 if (pot >= (value - 10) && pot <= (value + 10))
 {
  digitalWrite(in1Pin , LOW);
  digitalWrite(in2Pin , LOW);
 }
}
```

Function To command the motor to the desired position in the initial start.

```
void gyroSensor(void) // to activate the gyro sensor
{
   while((x_filtered > gyro_limits||x_filtered<- gyro_limits) || (y_filtered > gyro_limits||
y_filtered< - gyro_limits) || (z_filtered > gyro_limits||z_filtered<-gyro_limits))
    {
     mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
     x_filtered = alpha_gyro*x_filtered + (1-alpha_gyro)*(gx-avgX) ;
     y_filtered = alpha_gyro*y_filtered + (1-alpha_gyro)*(gy-avgY) ;
     z_filtered = alpha_gyro*z_filtered + (1-alpha_gyro)*(gz-avgZ) ;

     }

}
```

To activate the gyro sensor and eliminate the false signalling. Runs an empty while loop as long as the hand is in the movement

Flowchart for the basic algorithm.