Task 7 (Sensors and Motor Control Lab) Quiz

1. Reading a datasheet. Refer to the ADXL335 accelerometer datasheet (https://www.sparkfun.com/datasheets/Components/SMD/adxl335.pdf) to answer the below questions.
   - What is the sensor's range?
   - The minimum value is 6g. The typ. value is 7.2g
   - What is the sensor's dynamic range?
   - The minimum is : -3g to +3g. The typ. Is -3.6g to +3.6g
   - What is the purpose of the capacitor C_{DC} on the LHS of the functional block diagram on p. 1? How does it achieve this?
   - The stated capacitor function is to counter the effect of noise in the power supply. The justification for this is that capacitors don't respond to sudden change in voltage due to the dv/dt term in their equation and noisy signals are a reason for sudden changes in voltage.
   - Write an equation for the sensor's transfer function.
   - $$a = \frac{V_{out} - 1.5}{0.3}$$
   - What is the largest expected nonlinearity error in g?
   - The largest expected non-linearity is 0.3% of 7.2g = 0.216g
   - How much noise do you expect in the X- and Y-axis sensor signals when the sensor is excited at 25 Hz?
   - 750 ug
   - How about at 0 Hz? If you can't get this from the datasheet, how would you determine it experimentally?
   - 0 ug. The output value can be determined by measuring the value when the sensor is still.

2. Signal conditioning
   - Filtering
     - What problem(s) might you have in applying a moving average?
     - The problem with a moving filter is that it responds to current changes later since it also averages the data from the previous timesteps. So everything is lagged by a particular window depending on the size of the moving filter. Smaller windows are more responsive and vice versa, but too small a window and a moving average filter would lose its purpose.
     - What problem(s) might you have in applying a median filter?
     - A median filter is computationally expensive due to the need for sorting the data points. It has a O(n*logk) complexity where k is the size of the window and n is the size of the data stream. Median filters also lose peak/extreme values in the signal.

- Opamps
  - In the following questions, you want to calibrate a linear sensor using the circuit in Fig. 1 so that its output range is 0 to 5V. Identify which of V1 and V2 will be the input voltage and which the reference voltage, the value of the reference voltage, and the value of Rf/Ri in each case. If the calibration can't be done with this circuit, explain why.

    - Your uncalibrated sensor has a range of -1.5 to 1.0V.

    $$\frac{(V_1 - V_2)}{R_i} = \frac{(V_2 - V_{out})}{R_f}$$

    Then V_{out} can be represented by V_2 and V_1 : V_{out} = (1 + $\frac{R_f}{R_i}V_2$ – $\left(\frac{R_f}{R_i}\right)V_1$ The input is either V_1 or V_2 ; the output is V_{out} , take input V_1 as an example. Considering the minimum and maximum value of input and output. We have equation:

    $$\begin{cases} 0 = \left(1 + \frac{R_f}{R_i}\right)V_2 - \left(\frac{R_f}{R_i}\right)(-1.5) \\ 5 = \left(1 + \frac{R_f}{R_i}\right)V_2 - \left(\frac{R_f}{R_i}\right)1 \end{cases}$$

    If V1 is the input, Rf/Ri = -2 which is impossible.
    If V2 is the input, reference voltage is -3V, Rf/Ri = 1 [Solution]

    - Your uncalibrated sensor has a range of -2.5 to 2.5V.

The calibration can't be done with this circuit.The solved equation leads to impossible values.
If V1 is the input, Rf/Ri = -1
If V2 is the input, Rf/Ri = 0

3. Control
   - If you want to control a DC motor to go to a desired position, describe how to form a digital input for each of the PID (Proportional, Integral, Derivative) terms.

- $$u(t) = K_\mathrm{p} e(t) + K_\mathrm{i} \int_0^t e(\tau)\,d\tau + K_\mathrm{d} \frac{de(t)}{dt}$$

  The above equation is used for modelling the system.
  Kp – Proportional Term. Ki- Integral Term Kd- derivative term.
  e(t) – Error Rate.
  - If the system you want to control is sluggish, which PID term(s) will you use and why?
- Increase the Proportional Term and the integral term because these terms are connected to the error terms(e(t)) and therefore the system would respond faster.
  - After applying the control in the previous question, if the system still has significant steady-state error, which PID term(s) will you use and why?
- Increase the Integral Term, because this term controls the accumulation of the error in the past. It moves towards a particular desired value faster and eliminates steady state error.
  - After applying the control in the previous question, if the system still has overshoot, which PID term(s) will you apply and why?
- Increase the derivative term. This is because it acts as a feedback system.

Sensors and Motors Lab
Individual Lab Report

By Ritwik Das

Team D

Ritwik Das

Luka Eerens

Luxing Jiang

Keerthana Gopalakrishnan

Individual Progress:

For my contribution to the Sensor and Motors laboratory, I was responsible for the GUI and the software integration. I also helped Luxing a bit to get the IR sensor up and running.

A. GUI:

I chose Qt for this section.My GUI accomplishes the following tasks:

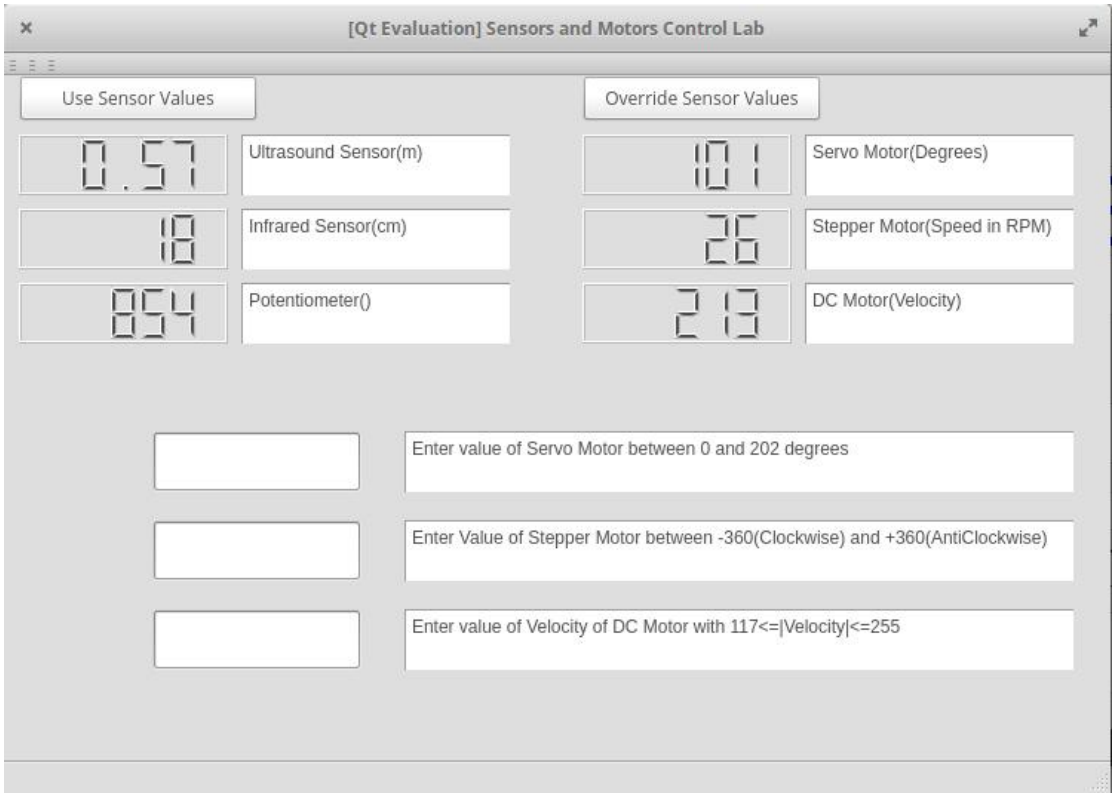
1. Report the value of each sensor and motor
2. Means for button override from the GUI motor control mode to the sensor control mode
3. Means for controlling the motors via GUI

B. Software Integration:

The integrated software accomplishes the following tasks in addition to the individual tasks:

1. State Machine implementation to switch from GUI motor control mode to the sensor control
2. Resolve conflicts and placing function calls as per given requirements in both the modes
3. Communication with Qt via Arduino serial monitor

A. GUI:



1. Report the value of each sensor and motor:

   My C++ code from the arduino serial port. The arduino outputs the state of each of the sensors and motors with commas. I used comma as the delimiter and just wrote the same to the GUI.

2. Means for button override from the GUI motor control mode to the sensor control mode:

   There are two buttons on the GUI as shown "Use Sensor Values" and "Override Sensor Values". They are used to switch control from the GUI controlling the motors to the sensors controlling the motors. This is done by writing a character to the arduino serial which is processed by the arduino program to switch the mode. This functionality has also been implemented via a button press on the breadboard.

3. Means for controlling the motors via the GUI:

   There are three text-boxes which take in the values by which the user wishes to operate the particular motor. Since each of the values are in different units and have their own limits and constraints, the instructions for the same are written in the adjacent box so that anyone can operate the GUI with ease.

B. Software Integration:

1. State Machine implementation to switch from GUI motor control mode to the sensor control

In this section I had to write code for de-bouncing for switching between the GUI and the sensors and also to do the requisite hardware connections. This functionality of switching has been enabled both via the GUI(explained earlier) as well as via a button press on the breadboard.

2. Resolve conflicts and placing function calls as per given requirements in both the modes

This section was very critical contrary to what was expected. The PID control code written by Keerthana didn't work with the collaborated code. This was because taking in input and giving output to the Serial monitor took a lot of time. I worked along with Keerthana to try to resolve it by setting a timeout limit. I didn't work with the PID from the start so it was hard for me to make changes to the position control code itself. We tried a lot of methods for I/O processing but none of them worked and in the end we only had the velocity control for the DC motor.

3. Communication with Qt via Arduino serial monitor.

In the software part, we are sending the states of the sensors and motors using strings which are manipulated within the Qt code.The Qt also writes motor control strings into the arduino Serial which is read by the ino program to change the state of the motors.

Challenges Faced:

Surprisingly for me, I didn't face any challenges in writing Qt code itself.However,I faced a lot of challenges in trying to get everything working together.

I was using Linux with Arduino and Qt installed.Some of the arduinos in the lab gave this error:"avrdude-stk500-recv-programmer-is-not-responding". This was when I tried to upload the code.I tried to debug it using some methods mentioned online but the didn't work.As a result of this, our team changed arduinos many times. Since we changed arduino many times,another very unfortunate thing that happened just hours prior to the demo was that the Qt couldn't read the arduino. I tried a lot of ways to solve the problem. But I couldn't see an error in my code since I had used the same code so many different times and it worked. We had to do the demo using the Arduino Serial monitor itself which was very demotivating for myself and the team since all the GUI code had been already written and tested perfectly and yet couldn't be shown during the demo.Later on in an attempt to solve the problem I tried the same program with a couple of more arduinos and it worked on one of them . At first I though this was an arduino problem but then I realized that the vendor ID

and product ID of some of the Arduino UNOs are different from the another set of UNOs. This didn't match with the Qt code and thus there was an error. I changed this and then it again started working. Earlier before the presentation I never thought that the vendor ID could be an issue. I should have solved this simple issue earlier but I am glad I know it now(and more about Arduinos). I had very very limited experience before this program with Arduino and none on using a GUI. After this lab I hope to do better the next time.

Teamwork:

1. Ritwik Das- I worked on the GUI and the software integration
2. Luka Eerens - He worked on the ultrasound sensor and the servo motor
3. Keerthana - She worked on the DC Motor and the potentiometer.
4. Luxing - He worked on the IR sensor and the Stepper motor.

Progress Review of the Project:

We have surveyed the landscape for datasets and have gone through some of the formalities associated with acquiring them. We are speaking ofcourse of multimodal datasets, and the ones that we have decided to focus on are the SEMAINE dataset, IEMOCAP dataset and Microsoft MISC dataset.
In our literature review, we have found a great number of research papers dealing only with bimodal emotion recognition (voice and visual modalities) and we will look into building the code for atleast one of these papers and training it for the immediate future (Next 2-3 weeks). I have partially finished writing code for one of the papers which won the Emotional Challenge. The vision part uses a pretrained ResNet-50. I have done the feature extraction and data preprocessing for output and feeding it into the network respectively. I also made progress on the voice modality which uses a temporal filter along with CNNs. The remaining parts are the LSTMs and by the progress review we hope to have implemented this paper and drawn various conclusions from the network.
So we are in the near term only focusing on bimodal emotion recognition and will slowly incorporate the third modalities later in the semester.

Appendix A:

Source Code

File: mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "QSerialPort"
#include "QSerialPortInfo"
#include "QDebug"


using namespace std;


MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    arduino = new QSerialPort;
    arduino_is_available =false;
    arduino_port_name = "";
    serialBuffer = "";
    usingsensorvals = true;
    qDebug() << "Number of available import:" <<
QSerialPortInfo::availablePorts().length();
    foreach(const QSerialPortInfo &serialPortInfo,
QSerialPortInfo::availablePorts()){
        qDebug()<<"1";
        if(serialPortInfo.hasVendorIdentifier() &&
serialPortInfo.hasProductIdentifier()){

if(serialPortInfo.vendorIdentifier()==arduino_uno_vendor_ID){

if(serialPortInfo.productIdentifier()==arduino_uno_product_ID){
                //qDebug() <<  "Vendor ID:  "<<
serialPortInfo.vendorIdentifier();
```

```
                //qDebug() <<  "Product ID:  "<<
serialPortInfo.productIdentifier();
                arduino_port_name = serialPortInfo.portName();
                arduino_is_available = true ;
            }
        }
    }
    if(arduino_is_available){
        //open and configure the serial port
        arduino->setPortName(arduino_port_name);
        arduino->open(QSerialPort::ReadWrite);
        if(!arduino->isOpen()){
            qDebug() << "Is Close";
        }
        else{
            qDebug() << "Is Open";
        }
        arduino->setBaudRate(QSerialPort::Baud9600);
        arduino->setDataBits(QSerialPort::Data8);
        arduino->setParity(QSerialPort::NoParity);
        arduino->setStopBits(QSerialPort::OneStop);
        arduino->setFlowControl(QSerialPort::NoFlowControl);

QObject::connect(arduino,SIGNAL(readyRead()),this,SLOT(readSerial()))
;
    }
    else{
        //if arduino available is error
        QMessageBox::warning(this,"Serial Port Error","couldn't find
arduino");
    }

}
void MainWindow::readSerial()
{
    QByteArray serialData;
    serialData = arduino->readAll();
    serialBuffer += QString::fromStdString(serialData.toStdString());
    if(serialBuffer[serialBuffer.length()-1] == 'E'){
//       qDebug() << serialBuffer;
        QStringList bufferSplit = serialBuffer.split(",");
        qDebug() << bufferSplit;
        MainWindow::updateReadings(bufferSplit);
```

```
        serialBuffer = "";
    }
}
void MainWindow::updateReadings(const QStringList readings){
    if(readings.length() == 8){
        ui->Sensor1->display(readings[1].toStdString().c_str());
        ui->Sensor2->display(readings[2].toStdString().c_str());
        ui->Sensor3->display(readings[3].toStdString().c_str());
        ui->Motor1->display(readings[4].toStdString().c_str());
        ui->Motor2->display(readings[5].toStdString().c_str());
        ui->Motor3->display(readings[6].toStdString().c_str());
    }
}
void MainWindow::controlMotor(const QStringList motor_reading,int
n,const QString type){
    if(arduino->isWritable()){
        arduino->write(type.toStdString().c_str());
        for(int i=0;i<n;i++){
            arduino->write(motor_reading[i].toStdString().c_str());
            arduino->write("\n");
        }
    }
    else{
        qDebug() << "Not able to control Motors";
    }
}
void MainWindow::on_overridesensorvals_clicked(){
    if(arduino->isWritable()){
        usingsensorvals = false;
        arduino->write("O");
    }
    else{
        qDebug()<<"Not able to Override";
    }
}
void MainWindow::on_usesensorvals_clicked()
{
    if(arduino->isWritable()){
        usingsensorvals = true;
        arduino->write("U");
    }
    else{
        qDebug() << "Not able to use Sensor values";
    }
```

```
}

void MainWindow::on_motor1_returnPressed()
{
    QString T = ui->motor1->text();
    QString t = "A"+T;
    qDebug()<<t;
    if(arduino->isWritable()){
        arduino->write(t.toStdString().c_str());
        qDebug()<<"Getting Written from QT";
    }
    ui->Motor1->display(T.toStdString().c_str());
}
void MainWindow::on_motor2_returnPressed()
{
    QString T = ui->motor2->text();
    QString t = "B"+T;
    qDebug()<<t;
    if(arduino->isWritable()){
        arduino->write(t.toStdString().c_str());
        qDebug()<<"Getting Written from QT";
    }
    ui->Motor2->display(T.toStdString().c_str());
}

void MainWindow::on_motor3_returnPressed()
{
    QString T = ui->motor2->text();
    QString t = "CS"+T;
    qDebug()<<t;
    if(arduino->isWritable()){
        arduino->write(t.toStdString().c_str());
        qDebug()<<"Getting Written from QT";
    }
    ui->Motor3->display(T.toStdString().c_str());
}

MainWindow::~MainWindow()
{
    if(arduino->isOpen()){
        arduino->close();
    }
    delete ui;
}
```

Appendix B:
Source Code
File:main.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "QSerialPort"
#include "QSerialPortInfo"
#include "QDebug"


int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.setWindowTitle("Sensors and Motors Control Lab");
    w.show();

    return a.exec();
}
```

Source Code
File:mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QDialog>
#include <QtSerialPort/QSerialPort>
#include <QSerialPortInfo>
#include <QDebug>
#include <QWidget>
#include <QMessageBox>

namespace Ui {
class MainWindow;
```

```
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
private slots:
    void readSerial();
    void updateReadings(const QStringList readings);
    void controlMotor(const QStringList motor_reading, int n,const
QString);
    void on_overridesensorvals_clicked();
    void on_usesensorvals_clicked();

    void on_motor1_returnPressed();
    void on_motor2_returnPressed();
    void on_motor3_returnPressed();

private:
    Ui::MainWindow *ui;
    bool usingsensorvals;
    QSerialPort *arduino;
    static const quint16 arduino_uno_vendor_ID = 9025;
    static const quint16 arduino_uno_product_ID= 67;
    QString arduino_port_name ;
    bool arduino_is_available ;
    QByteArray serialData;
    QString serialBuffer;

};

#endif // MAINWINDOW_H
```

Appendix C:
Source Code
Integrated Arduino Code

```
#include <Servo.h>
#include<Filters.h>
//#include<DateTime.h>

Servo servo;
const int ultrasoundPin = 9;
const int servoPin = 5;
float distVal;
int distTriggerVal;                          // variable that is read by ultrasound pin
int distTriggerList[] = {0, 0, 0};           // Contains 5 sensor reading values to feed into noise cancelling function
int orderedList[] = {0, 0, 0};
const int distListSize = 3;
int pos = 0;
int previousSonarValues[] = {0};
int ultraSonicPhi = 500;
bool control_GUI = false;
int STEP = 8;
```

```
int DIR = 7;
int sensorValue = 0;
float filterFrequency = 50;
FilterOnePole lowpassFilter(LOWPASS, filterFrequency);

// if the total gain we get is not in the PWM range we scale it down so that it's not bigger than |255|
int potpin = A1;      // select the input pin for the potentiometer
int inpin0 = 3;
int reading1 = 0;
int state_val = 0; // the current reading from the input pin
int previous1 = LOW;       // the previous reading from the input pin
long time = 0;                // the last time output pin was toggled
long debounce = 200;
int pwmpin = 6;
int val = 0;
int bri;
volatile double count = 0;//set the counts of the encoder
volatile double angle = 0;//set the angles
volatile double prev_angle = 0;
volatile double curr_spd = 0;
boolean A, B;
double PWM = 6;
const int InA1 = 4;    //
const int InB1 = 10; //
int encodPinA1 = 2;                              // encoder A pin
int encodPinB1 = 11;                             // encoder B pin

double setpoint = 100;//1 setting it to move through 100 degrees
double tar_spd = 0.3;
double Kp = 5;// you can set these constants however you like depending on trial & error
double Ki = 0;
double Kd = -5;
char mode = 'P';
int vel = 116;

float last_error = 0;
float error = 0;
float changeError = 0;
float totalError = 0;
float pidTerm = 0;
float pidTerm_scaled = 0;
volatile int affirm_trigger;
int mytimestep = 2;
void Stepper_GUI(bool clockwise, float degree) ;
```

```
void potentiometer()
{
  val = analogRead(potpin);
  bri = map(val , 0 , 1023 , 0, 255 );
  digitalWrite(InA1, LOW);
  digitalWrite(InB1, HIGH);
  analogWrite(pwmpin, bri);
  // Serial.println(bri);
}
void StepperMotor(float speed)
{
  digitalWrite(DIR, LOW);
  for (int i = 0; i < 200 * 4; i++)
  {
    digitalWrite(STEP, HIGH);
    delayMicroseconds(speed);
    digitalWrite(STEP, LOW);
    delayMicroseconds(speed);
  }
}
void Stepper_GUI(bool clockwise, float degree) {
  if (clockwise) {
    digitalWrite(DIR, HIGH);
  } else {
    digitalWrite(DIR, LOW);
  }

  float num_of_circle = degree / 360.f;
  for (int i = 0; i < 200 * 4 * num_of_circle; i++)
  {
    digitalWrite(STEP, HIGH);
    delayMicroseconds(1000);
    digitalWrite(STEP, LOW);
    delayMicroseconds(1000);
  }
}


void setup() {
  servo.attach(servoPin);              // attaches the servo on pin 9 to the servo object
  Serial.begin(9600);                  // Make sure baud rates are identical for all subsystems
  pinMode(ultrasoundPin, INPUT);
  pinMode(A0, INPUT);
```

```
  pinMode(STEP, OUTPUT);
  pinMode(DIR, OUTPUT);
  pinMode(encodPinA1, INPUT);//encoder pins
  pinMode(encodPinB1, INPUT);
  attachInterrupt(digitalPinToInterrupt(inpin0), affirm, CHANGE);
  attachInterrupt(0, Achange, CHANGE);
  pinMode(PWM, OUTPUT);
  pinMode(InA1, OUTPUT);
  pinMode(InB1, OUTPUT);
  analogReference(DEFAULT);
  pinMode(pwmpin, OUTPUT);
  Serial.setTimeout(mytimestep);
}


void affirm() {        //Used for Button0
  int button0State = digitalRead(inpin0);
  if (button0State == HIGH) {
    control_GUI = !control_GUI;
    affirm_trigger = 1;
  }
}

void readSonar() {

  for (int i = 0; i < distListSize; i++) {
    distTriggerVal = pulseIn(ultrasoundPin, HIGH);        // These values are given in microseconds
    //Serial.println(distTriggerVal);
    // If the distValue is not an outlier
    if (distTriggerVal > 800 && distTriggerVal < 2500) {

      // If this is the first timestep
      if (previousSonarValues[0] == 0) {

        // quicker to append these values directly than transform them each before appending
        distTriggerList[i] = distTriggerVal;
        orderedList[i] = distTriggerVal;                          // This comes in handy in the order function
      }
      // Any timestep after that
      else {
        // If it is a gradual change and not a sudden one (smaller than ultraSonicPhi)
        if (abs(distTriggerList[i] - previousSonarValues[0]) <   ultraSonicPhi) {
          distTriggerList[i] = distTriggerVal;
          orderedList[i] = distTriggerVal;
```

```
            }
            // Pretend like the measurement never exhisted
            else {
               i--;
            }
         }
      }

      if (distTriggerVal < 800 || distTriggerVal > 2500) {
         i--; // Reset i iterator by reversing it by one
      }
   }
}


void order() {
   // Sort the list of signal values in order to isolate the outliers at the edges of the list

   float smallest = 3000;              // This is clearly not a value it can reach
   int smallestIndex = 0;
   for (int j = 0; j < distListSize; j++) {
      for (int i = 0; i < distListSize; i++) {
         if (orderedList[i] < smallest) {
            smallest = orderedList[i];
            smallestIndex = i;
         }
      }
      distTriggerList[j] = smallest;
      orderedList[smallestIndex] = 4000;           //you now want the next smallest value, so make the current smallest
value huge
   }
}


void median() {
   // Find the average of the 3 centermost values

   //distVal = float(distTriggerList[2] + distTriggerList[3] + distTriggerList[4])/float(3);
   distVal = distTriggerList[1];
   previousSonarValues[0] = distVal;

   // Now convert this microsecond value into a distance value
   distVal = float(distVal) / 1000000;              // first so convert to sec
   distVal = distVal * 343;                           // next convert to dist (speed of sound = 343m/s)
```

```
}


// This function was obtained from:
https://electronics.stackexchange.com/questions/83458/best-way-to-map-ints-to-float-in-arduino
float mapfloat(float x, float in_min, float in_max, float out_min, float out_max) {
   return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
struct motorA {
   bool active;
   int distance;
};
struct motorB {
   bool active;
   bool rot_direction;
   float mot_speed;
};
struct motorC {
   bool active;
   char mode;
   int set_point_val;
};

void Interrupt() {
   if (!digitalRead(inpin0)) {
      return;
   }
   control_GUI = !control_GUI;
}
int ccc = 0;
// 2 interesting things the ultrasonic range finder detects stuff by reflecting echo on table surface
// it also occasionally detects crazy values ESPECIALLY WHEN going from away to towards the ultrasound
void loop() {

   if (affirm_trigger == 1) {
      detachInterrupt(digitalPinToInterrupt(inpin0));
      delay (65);
      attachInterrupt(digitalPinToInterrupt(inpin0), affirm, CHANGE);
   }
   readSonar();
   order();
   median();
   struct motorA a;
   struct motorB b;
```

```
struct motorC c;
a.active = false;
b.active = false;
c.active = false;
String input    = Serial.readString();

float number    = 0 ;
if (input[0] == 'O' || (control_GUI && input[0] != 'U')) {
   int i = 1;
   if (control_GUI) {
      i = 0;
      if (input[i] == 'A') {
         a.active = true;
         i = 1;
      }
      else if (input[i] == 'B') {
         b.active = true;
         if (input[i + 1] == '-') {
            b.rot_direction = true;
            i = 2;
         }
         else {
            b.rot_direction = false;
            i = 1;
         }
      }
      else if (input[i] == 'C') {
         c.active = true;
         c.mode = input[i + 1];
         if (input[i + 2] == '-') {
            c.set_point_val = -1;
            i = 3;
         }
         else {
            c.set_point_val = 1;
            i = 2;
         }
      }
   }
   control_GUI = true;
   while (input[i] != '\0' && input[i] != '\n' && input[i] != EOF) {
      if (int(input[i]) < 48 ||    int(input[i]) > 57) {
         break;
      }
```

```
        number = max(number, 0) * 10 + input[i] - 48;
        i++;
      }
    }
    else if (input[0] == 'U') {
      control_GUI = false;
      //      Serial.println(control_GUI);
    }
    if (!control_GUI) {
      float motorval = mapfloat(distVal, 0.28, 0.85, 0.0, 202.5);
      int raw, volt, cm;
      float disVal = analogRead(A0);
      disVal = lowpassFilter.input(disVal);
      float Speed = map(disVal, 700, 100, 400, 1000);
      StepperMotor(Speed);
      servo.write(int(motorval));
      potentiometer();
      Serial.print("B,"); Serial.print(distVal); Serial.print(","); Serial.print(round(disVal)); Serial.print(",");
Serial.print(round(val)); Serial.print(",");
      Serial.print(round(motorval)); Serial.print(","); Serial.print(round(Speed)); Serial.print(","); Serial.print(round(bri));
Serial.print(",E");
    }
    else {
      //      Serial.println(input);
      //      Serial.println(number);
      if (a.active) {
        a.distance = int(number);
        servo.write(a.distance);
      }
      else if (b.active) {
        b.mot_speed = number;

        Stepper_GUI(b.rot_direction, b.mot_speed );
      }
      else if (c.active) {
        //if (C.mode == 'P') {
        if (c.mode == 'P') {
          loop_PID_POS();
        }
        else if (c.mode == 'S') {
          vel = number * c.set_point_val;
          velcontrol(vel);
        }
      }
```

```
  }
}
void loop_PID_POS() {

  PIDcalculation_POS();// find PID value

  if (angle > setpoint) {
    digitalWrite(InA1, LOW);// Forward motion
    digitalWrite(InB1, HIGH);
    //      delay(50);
  }
  else {
    digitalWrite(InA1, HIGH);// Forward motion
    digitalWrite(InB1, LOW);
    //      delay(5);
  }

  analogWrite(PWM, pidTerm_scaled);

  analogWrite(PWM, pidTerm_scaled);

  delay(100);
}


void PIDcalculation_POS()
{
  angle = (1 * count);//count to angle conversion
  //   Serial.println(count);
  error = setpoint - angle;

  changeError = error - last_error; // Serial.println(error);derivative term
  totalError += error; //accumalate errors to find integral term
  pidTerm = (Kp * error) + (Ki * totalError) + (Kd * changeError);//total gain

  pidTerm = constrain(pidTerm, -255, 255);//constraining to appropriate value
  pidTerm_scaled = abs(pidTerm);//make sure it's a positive value
  last_error = error;
}



void Achange() //these functions are for finding the encoder counts
{
```

```
  A = digitalRead(encodPinA1);
  B = digitalRead(encodPinB1);

  if (A == B)
  {
    count++;
  }
  else
  {
    count--;

  }
}

void velcontrol(int vel) {
  if (vel >= -255 && vel <= 255)
  {
    if (vel > 0)
    {
      motorForward(vel);
    }
    else
    {
      motorBackward(-1 * vel);
    }
  }
}
void motorForward(int PWM_val)   {
  analogWrite(PWM, PWM_val);
  digitalWrite(InA1, LOW);
  digitalWrite(InB1, HIGH);
}

void motorBackward(int PWM_val)   {
  analogWrite(PWM, PWM_val);
  digitalWrite(InA1, HIGH);
  digitalWrite(InB1, LOW);
}
```