Sensors and Motors Lab
Individual Lab Report #1

Luka Eerens

Team D: Deeply Emotional

Teammates:
Luxing Jiang, Keerthana P G, Ritwik Das

October 13th 2017

1. Individual Progress:

My primary role was to interface the ultrasonic range finder with the Arduino microcontroller and get it to output a stable distance readout and then drive the servo. My second role was to integrate the hardware, put together the motor driver, and assemble and connect all of the components together.

2. Components

2. 1 Ultrasonic Range Finder:

My task was to interface the Ultrasonic range finder with the Arduino. This sensor was the LV-MaxSonnar EZ MB1010. I looked at the specification sheet for this sensor and found that just like pretty much every other ultrasound sensor I have used, that it was low power enough to be directly powered by the Arduino without the need for auxiliary power cables.

There are 2 main ways of connecting the ultrasonic range finders, the first option is to read the analog pin the other is to read the PW (pulse width) pin. I tested both approaches and found that sampling from the PW pin yielded a more stable reading of the distance whereas analog reading seemed to have more noise.

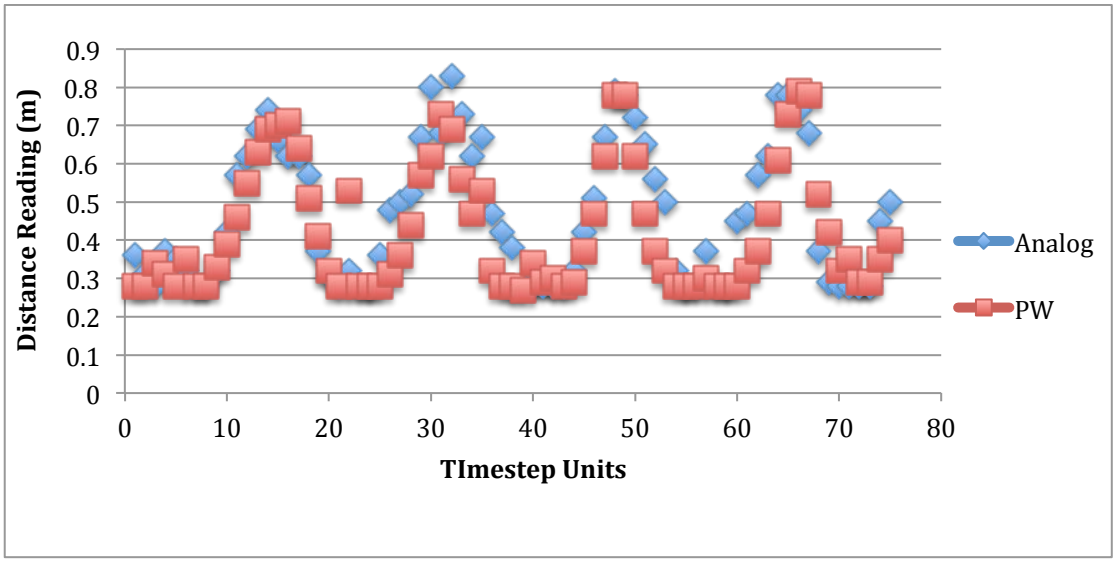

Figure 1: PWM VS Analog Distance Readouts from Ultrasound

So I connected the Arduino to the Ultrasonic range finder by connecting the PW pin of the ultrasound with PWM pin 9 of the Arduino. The code for operating this ultrasonic sensor on the Arduino was available online and I modified it to reflect our needs. It employed a scaled down, selective median filter that works just like a median filter but only adds values to the list if they are very close in value to the previous correct determined distance from the median value in the previous main code loop. This very close value is determined by an integer threshold value called ultraSonicPhi. This value was defined in the code by experimenting with different values and observing how well they perform. This is described in the challenges section later.

2. 2 Hardware Integration:

The first step involved Soldering all of the components for the motor driver. This was done, with the final results displayed below:

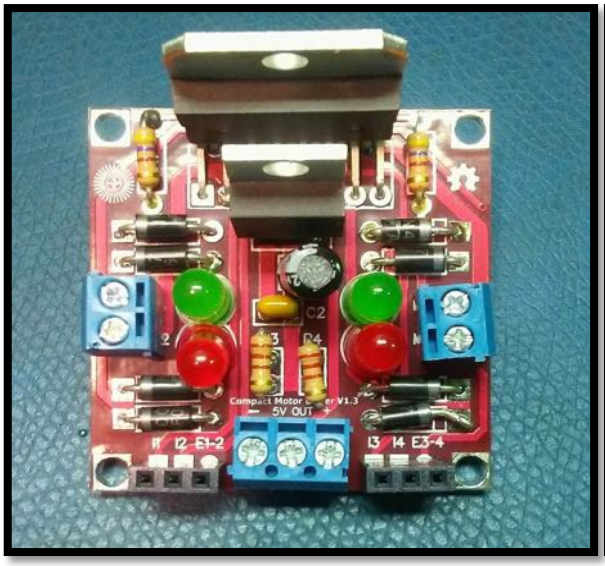

Figure 2: Front View of Finished Motor Driver

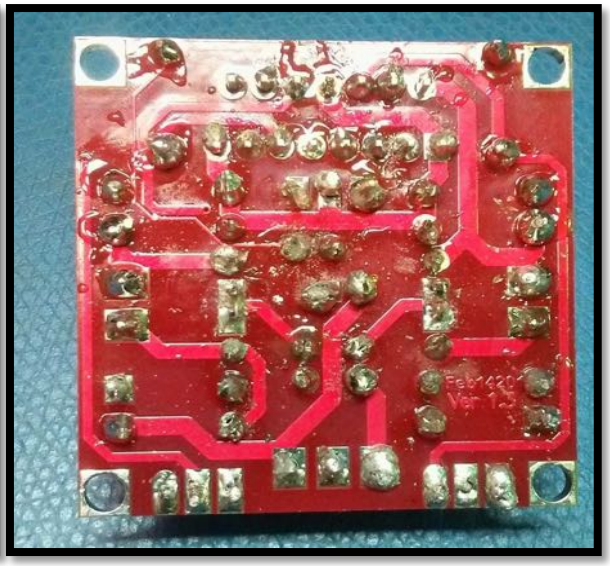

Figure 3: Rear View of Finished Motor Driver

The next step involved planning the layout of the motors and servos on our wooden plank, drilling the holes in the plank in order to wrap plastic fasteners around them. I did this part together with Luxing.

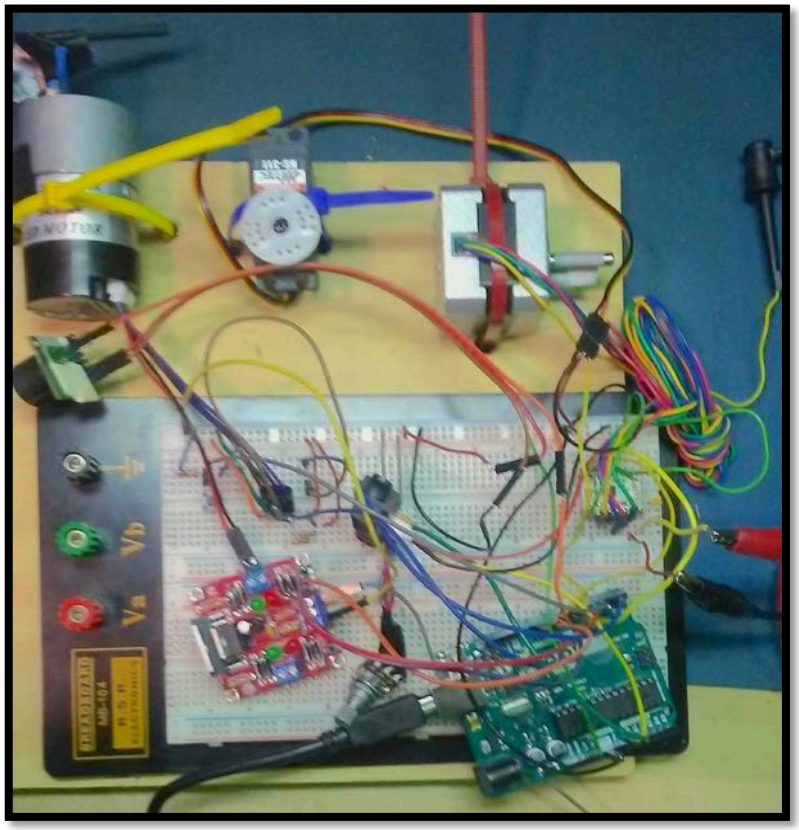

Figure 4:Breadboard with sensors and motors

3. Challenges:

3. 1 Ultrasonic Range Finder:

Drawbacks of noise filtering for this are that if you are coming in very fast, then, you will get quite a big variation in echo values.

As mentioned earlier

The biggest challenge with the ultrasonic range finder is that it was intrinsically noisy and was quite stubborn in its performance. In the case of controlling a servo, the random extreme fluctuations in ultrasound readings became magnified by the sudden and occasional noisy servo rotation in the opposite direction of where it was meant to be.

I decided to address this firstly using sample median filters, which improved the accuracy of the readings by ignoring outliers for each measurement sample. Several different sample sizes were considered and it was found that having smaller sample sizes lead to a more fluid motion of the connected servo (because of the reduced latency) but these smaller sample sizes were often not able to have a median that was isolated away from the outliers after sorting the sample size.

So I tried to bring the best of both worlds by having a small sample size paired with a filter that only let numbers be appended to this sample list if they were not that much smaller or greater than the previous correct median computation of the previous main loop.

This filter was just an integer threshold called ultraSonicPhi and various difference thresholds were experimented on. It was found that the value of 500 was seemed to yield curves, which blended smoothness with frequency of datapoints the best. Figure 5 shows the results of this test.

The problem with this algorithm is that it only adds values to the list if they differ by only so much to a previous accepted median value for sample size, but that median value may have been on the low end of acceptable and so every other measurement thereafter will be ignored because this threshold difference will not be enough..

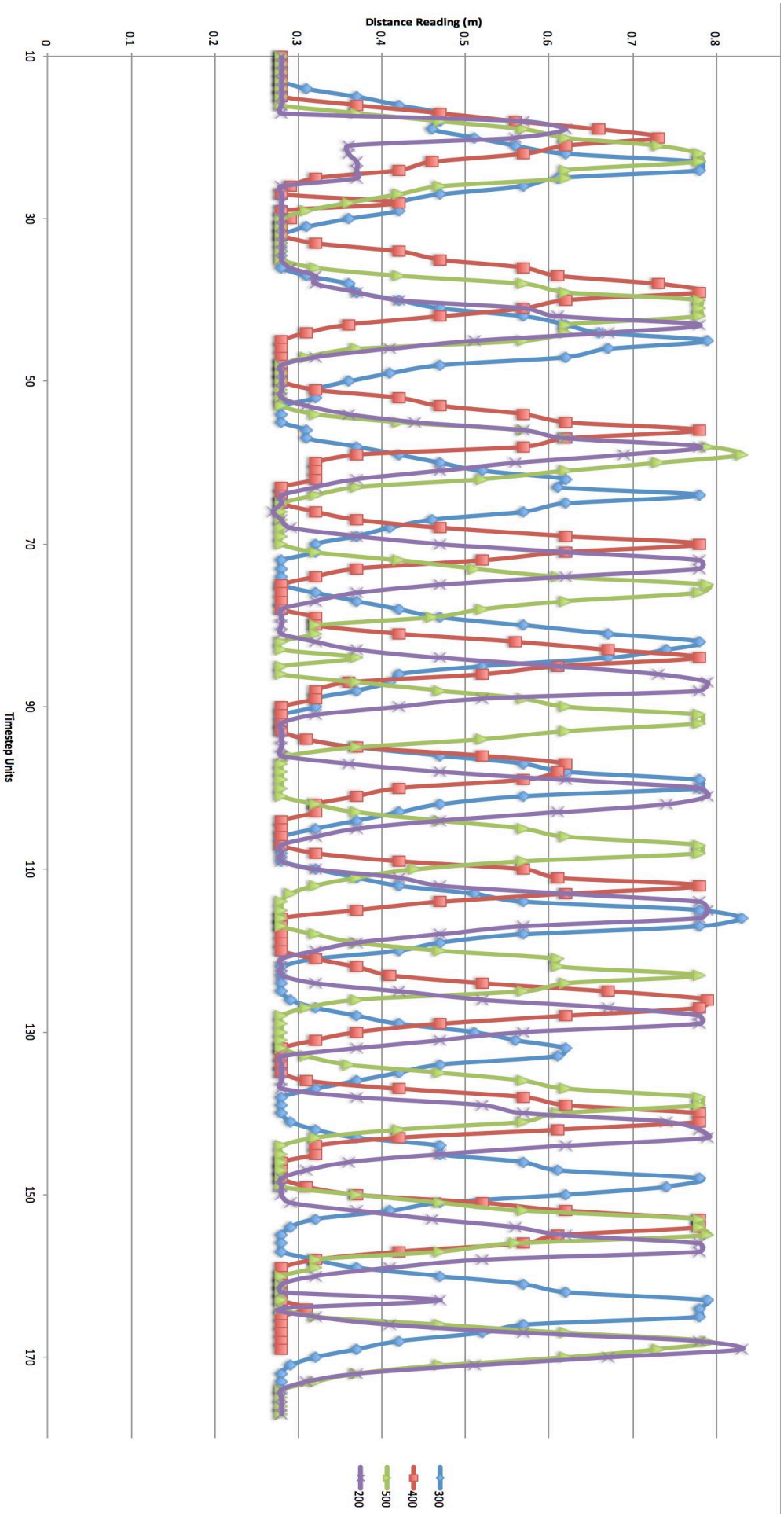

Figure 5: Distance Measurements for recursive hand waving filtered by different UltraSonicPhi Values

3. 2 Hardware Integration:

I made some careless mistakes that were time consuming to make right in regards to soldering together the motor driver. This included soldering the capacitor on the wrong way and the laborious process that was needed to slowly pull the capacitor out when residual solder held the pins in place. This procedure took about 20 minutes to do but would have taken longer had it not been for Luxing arriving to the lab to provide a helping hand.

Another little challenge with the hardware integration was figuring out how to Frankenstein together a circuit from all of the individual ones that our team used to develop their own sensor-actuator subsystem. Every student instinctively laid out their circuit differently and so just using the same wires, and connection points was not enough as the servos and DC motors were already fixed in place, and sometimes the wires currently used were not long enough to reach their set-up location.

Nonetheless this challenge in itself wasn't that complicated as each team mate set up their own hardware to test their subsystem and by looking at their subsystems, I was able to integrate it into the fabric of the circuit. Even though debugging of the sensors and motors was still taking place close to the deadline, the hardware itself was integrated on the circuit board, which we all cluttered around.

4. TeamWork

We are a 4-person team and therefore are only using 3 sensors in this lab to drive our motors and servos. The devices that we have decided to use are:

- Ultrasonic range finder
- Ambient Light Sensor
- Rotary Encoder

The way that we have split up the tasks amongst ourselves is:

- Luka Eerens: Ultrasonic Range Finder, Hardware Integration, Servo
- Luxing Jiang: Ambient Light Sensor, Stepper Motor, System Integration
- Keerthana P G: Rotary Encoder, PID, DC Motor
- Ritwik: Graphical User Inteface (GUI)

Due to our small team, large workload and large list of deliverables outside of this lab, we decided to partition the tasks early on and factored in our availability during the due week.

In order to quickly provide our codes for system integration we decided that I would set up our hardware quickly and develop my Ultrasonic Range Finder code early as the entire upcoming week was fully booked with meetings, midterm exams, assignments, and start-up deliverables.

We did not work in close proximity to each other because of our unavailability to each other during the week so ensured that good code readability and manageability was done. I was able to quickly build the ultrasonic sensor and servo control and passed it over to Ritwik early on to test with the GUI. It worked like a charm and had no issues. However due to poor coordination this left Keerthana and Luxing stuck with the DC motor and Stepper motor close to the deadline which meant there wasn't much time left to debug the software integration.

5. Future Plans

Our MRSD capstone project is software heavy (multimodal emotion recognition) and features almost no moving parts. The only mechanically actuated component of our project is the instrument cluster, which needs to be directed towards the human. The development of this subsystem is something we have relegated to the spring semester as we consider it unrelated to the task of multimodal emotion recognition. We want to be sure that we have algorithms that detect emotion from the joint use of visual, acoustic and lexical modalities and so will focus our efforts on that instead.

This subsystem however will emulate the product of our sponsor by being something you place on a table-top, which essentially has 2 degrees of freedom of rotation. The camera in our instrument cluster needs to be able to track a human if they were to pace around the table and therefore would need a servo aligned with the Y-axis (perpendicular to table surface). The robot would also need to orient its camera up to the humans face if they are close by. This means having a second servo aligned with an axis perpendicular to the first to allow for upwards and downwards tilt.

As far as moving pieces go, we will have something that looks very similar to a 2 degree of freedom gimbal. Everything else will be the instrument cluster and software.

Figure 6: 2 Degree Of Freedom Gimbal

Also given the light weight of our instrument cluster (a cheap and simple camera such as a PiCamera or Microsoft Kinect paired with a lightweight microphone), we probably do not see the need for any motor driver or relay type device. The servos that we are looking into require a low power signal from the Arduino and are fed with the power to actually execute the turns from either a battery or the mains.

Progress Review Goals

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). We hope to have implemented it by next meeting, and if not than atleast a good portion should be complete.

So we are in the near term only focusing on bimodal emotion recognition and will slowly incorporate the third modalities later in the semester.

.

References:

Figure 4: Xeryon Gimbal found on this link
https://static.wixstatic.com/media/d9ce98_62281bcc95f64049aa9a41015a8a44e4~mv2_d_1625_1955_s_2.png/v1/fill/w_242,h_263,al_c,usm_0.66_1.00_0.01/d9ce98_62281bcc95f64049aa9a41015a8a44e4~mv2_d_1625_1955_s_2.png

Appendix 1. 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?
     Min: 6g (From: -3g <= sensor range <= +3g)
     Typ: 7.2g (From: -3.6g <= sensor range <= +3.6g)
   - What is the sensor's dynamic range?
     Min: ±3g
     Typ: ±3.6g
   - What is the purpose of the capacitor C_{DC} on the LHS of the functional block diagram on p. 1?
     Decouple accelerometer from power supply noise.

How does it achieve this?
Capacitor resists sudden voltage changes and acts as local energy storage device.

- Write an equation for the sensor's transfer function.

$$V_{out} = V_0 + sensitivity * acceleration$$

$$\alpha = \frac{V_{out} - 1.5}{\frac{300}{1000}} = \frac{3 - 1.5}{0.3} = 5m/s^2$$

- What is the largest expected nonlinearity error in g?

$$Error_{nonlinearity} = 0.3\% * 7.2g = 0.216g$$

- How much noise do you expect in the X- and Y-axis sensor signals when the sensor is excited at 25 Hz?

$$= 150 * \sqrt{25} = 750g$$

- How about at 0 Hz? If you can't get this from the datasheet, how would you determine it experimentally?
  Measure output of sensor when sensor is still

2. Signal conditioning
   - Filtering
     - What problem(s) might you have in applying a moving average?
       There are significant challenges with finding max values by applying a moving average. To add, all sorts of response samples such as ones with drastic changes or oscillations with constant amplitude disappear when applying moving average filters. One should thus think carefully about the nature of the data before settling on this filter.
     - What problem(s) might you have in applying a median filter?
       Applying a median filter requires you to sample multiple measurements before passing only one single value through. This means that there is a buildup in the latency that squares with sample size as you need 2 for loops to iterate through the values in order to sort them. Incrementing both by 1 will taka a lot more time than the previous for loop sizes... This latency also means that you run the risk of losing data, and if you were to drive a motor or a servo based on this, you would observe sudden stochastic changes in torque or position even if the sensor was exposed to something that changes very gradually. Another challenge includes losing data especially peaks in the signal as when the samples around them are sorted the peak will never be the median.

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

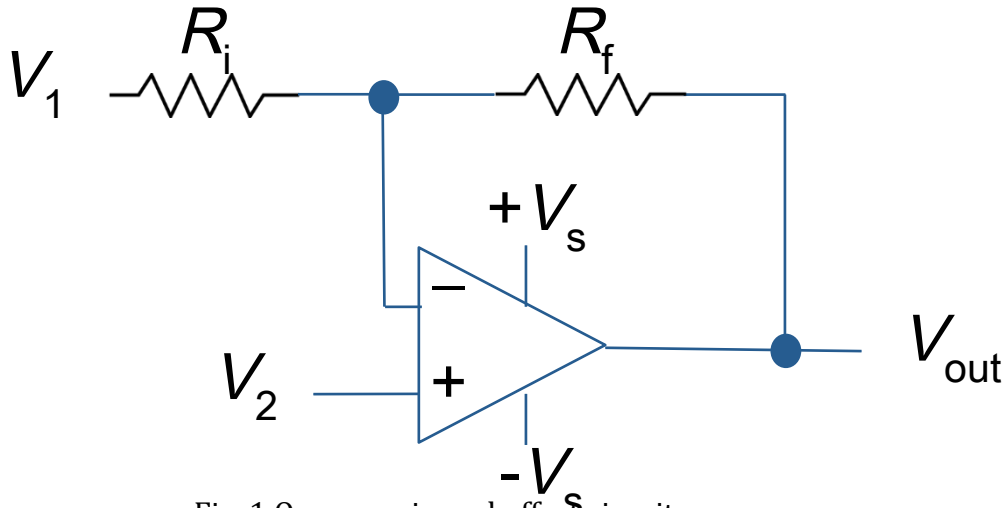

Fig. 1 Opamp gain and offset circuit

For uncalibrated sensor which has a range from -1.5V to 1.0V

$$\Delta V_{in} = 1 + 1.5 = 2.5$$
$$\Delta V_{out} = 2.5 + 2.5 = 5.0$$

$$\therefore \frac{\Delta V_{out}}{\Delta V_{in}} = \frac{5.0}{2.5} = 2.0$$

$$\frac{V_2 - V_1}{R_i} = \frac{V_{out} - V_2}{R_f}$$
$$V_{out} = V_2\left(1 + \frac{R_f}{R_i}\right) - \frac{R_f}{R_i}V_1$$

$$\therefore 1 + \frac{R_f}{R_i} = 2$$
$$\therefore \frac{R_f}{R_i} = 1$$

$$V_{out} = 2V_2 - V_1$$

$$\text{If } V_{out} = 0 = 2(-1.5) - V_1$$
$$\therefore V_1 = -3$$
$$\text{If } V_{out} = 5 = 2(1.0) - V_1$$
$$\therefore V_1 = -3$$

$\therefore V_1$ is response voltage such that $\frac{R_f}{R_i} = 1, V_1 = -3V, and\ V_2$ is input voltage

For uncalibrated sensor which has a range from -2.5V to 2.5V

$$V_{out} = V_2\left(1 + \frac{R_f}{R_i}\right) - \frac{R_f}{R_i}V_1$$

Make sure V_{out} changes with V_{in} so substitute V_{in} into the 1 in the expression above.

$$\therefore \frac{\Delta V_{out}}{\Delta V_{in}} = \frac{5.0}{2.5 + 2.5} = 1.0$$

$$\therefore 1 + \frac{R_f}{R_i} = 1$$

$$\therefore \frac{R_f}{R_i} = 0$$

$$\therefore \Delta V_{out} = V_2 - V_1$$

$$\text{If } V_{out} = 0 = -2.5 - V_1$$
$$\therefore V_1 = -2.5V$$
$$\text{If } V_{out} = 5 = 2.5 - V_1$$
$$\therefore V_1 = -2.5V$$

However in this scenario, $V(R_f) \neq 0$ and $R_f = 0$ and so in $V = IR,\ I \to \infty$ so no solution…

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.
     At each timestep calculate the error for the past, present and future. Assign a singular weight to each error (past error, present error, future error) and use these together in order to get desired output. Here is an equation for PID:

     $$Output(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \tau \frac{de(t)}{dt}$$

     Where:
     $K_p e(t)$ = current error
     $K_i \int_0^t e(\tau)d\tau$ = All past errors accumulated
     $K_d \tau \frac{de(t)}{dt}$ =Change in error (Can be used to predict future error)

   - If the system you want to control is sluggish, which PID term(s) will you use and why?

K_p and K_i should be increased because they determine size of adjustment step.

- 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 K_i because this term represents all past errors accumulated and a larger K_i means a larger coefficient to all these errors, which could help reduce or even eliminate this 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 K_d and thereby help predict future error to make the output converge with the correct true value faster.

Appendix 2: Arduino Code:

```
#include <Servo.h>

Servo servo;  // create servo object to control a servo: We are going to measure distance through pulse width modulation
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;          // This is the magnitude of tolerable difference in time it takes to receive
// an echo at each timestep. This is used to ignore sudden bursts or dips in echo receive time that lie within acceptable range



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);
}
```

```
void readSonar(){

   for (int i = 0; i < distListSize; i++){
      distTriggerVal = pulseIn(ultrasoundPin, HIGH);     // These values are given
in microseconds

      // 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;
}

// 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() {

  readSonar();
  order();
  median();
  Serial.println(distVal);
  distVal = mapfloat(distVal, 0.28, 0.85, 0.0, 202.5);
  distVal = int(distVal);
  //Serial.println(distVal);
  //Serial.println();
  servo.write(distVal);


  delay(15);
 }
```