Individual Lab Report #01

Sensors and Motors Lab

February 13th, 2020

Alex Withers

Pit Navigator

Team G: The Pit Crew: Awadhut Thube, Justin Morris, Alex Withers

Individual Progress **3**
- Motor and Sensors Lab 3
  - Stepper Motor 3
  - Infrared Rangefinder Sensor 3
  - Circuit and Code 4
- Pit Navigator 5
  - Camera Resolution + Imaging Techniques 5
  - Gathering Resources 5
  - Testing Site 6
  - Simulation 6

Challenges **6**
- Motor and Sensors Lab 6
  - Stepper Motor 6
  - Circuit 7
- Pit Navigator 7
  - Gathering Resources 7
  - Simulation 7

Teamwork **7**
- Motor and Sensors Lab 7
- Pit Navigator 8

Plans **8**
- Pit Navigator 8
  - Testing Site 8
  - Simulation 8

Sensors and Motors Quiz **9**

Appendix A: Code that I wrote **12**

Individual Progress

Motor and Sensors Lab

For this lab, my responsibility was to create a subsystem that consisted of a Sharp Infrared Rangefinder sensor (referred to as IR sensor) and a 12V NEMA 17 Stepper Motor (referred to as stepper). I used the IR sensor to convert the analog output into a real-world value (distance in cm). I sent the distance to the GUI and used the value to drive the stepper position.

Stepper Motor

The stepper motor given to us was a 12V, 1.8 degree step, NEMA 17, Mercury Motor. I have worked with stepper motors of this kind in the past and the work to implement it in this lab was in line with my experience. We were also given a SchmalzHaus.com EasyDriver, to drive the stepper motor through activation of a STEP pin. To control the direction the motor steps in, I drove the DIR pin on the stepper driver to HIGH or LOW. The last interface on the stepper driver was grounding the M1 and M2 pins so that the stepper will take full 1.8 degree steps rather than microstepping. It was then an easy attachment from the motor wires to the motor pins according to the schematic and attaching 12V and ground.

Infrared Rangefinder Sensor

The IR Sensor that we used was a Sharp 2Y0A02 Infrared Rangefinder. It ran off of 5V and had 3 pins, Vin, GND, and OUT. The sensor’s range was 20cm to 150cm with values from 500 to 0 along its path. The transfer function is as follows:

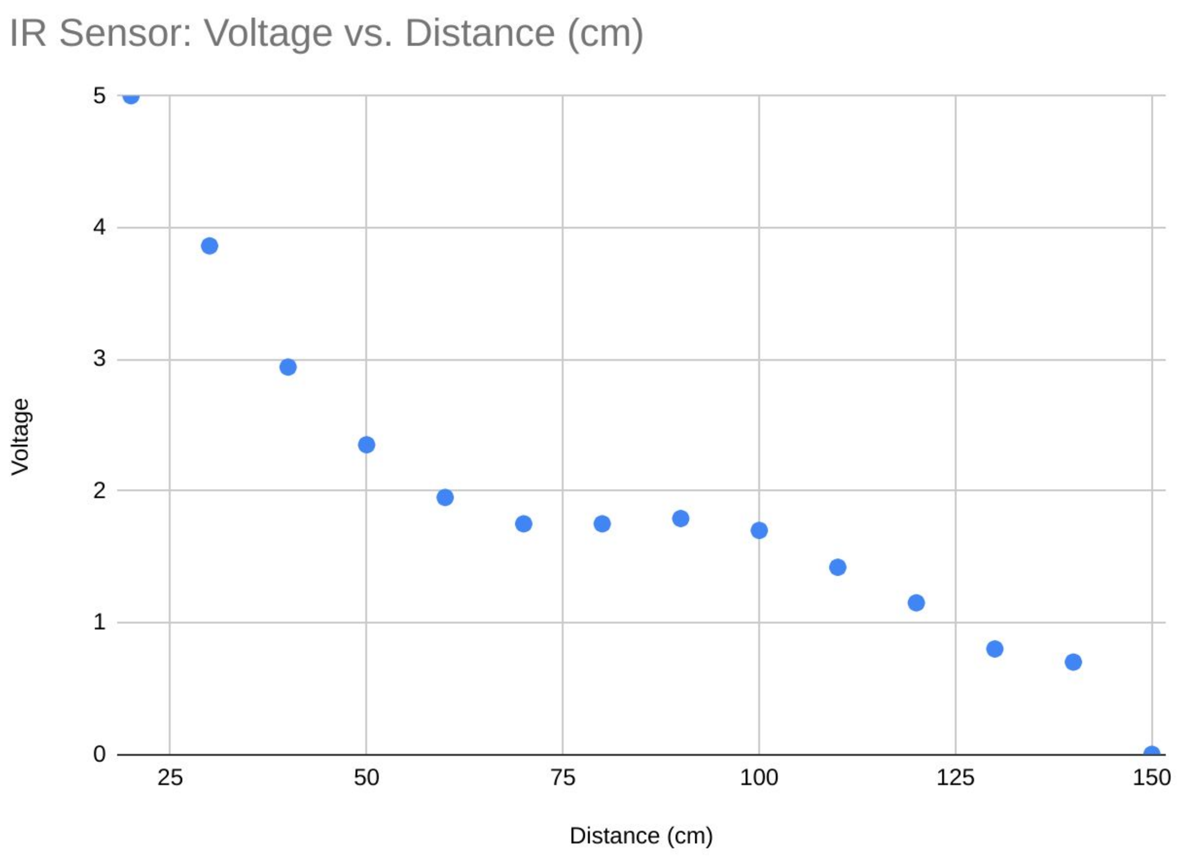

Figure 1: IR Sensor Transfer Function

As one can see this function is nonlinear and has a plateau in the middle of its range. It seems that this sensor was faulty as 60 to 100 cm ranges are indistinguishable from each other. To fix this for the demo, I averaged 5 values and if the resulting value was farther than 60 cm, it was at max distance. This limited the effective range of the sensor to 20 to 60 cm, however given the limited space in the lab, this was acceptable. The range from 20 to 60 cm is only slightly nonlinear, but not within any reasonable human perception, so I treated it as linear. This counted for the filtering sensor values deliverable.

Circuit and Code

I was in charge of connecting the master circuit and my subsystem's circuit. I have previously described some of my subsystem's circuit, but will describe in detail here. The arduino connects to the EasyDriver through the STEP, DIR, and GND pins on the 7,8 and GND respectively. The M1, and M2 pins of the EasyDriver connect to the master GND rail. And the A,B,C, and D motor coils connect to their respective places on the Easydriver. Finally the EasyDriver is powered by the 12V master rail and the GND master rail. The IR Sensor is powered on the 5V master rail and the GND master rail. The OUT wire is connected to A1 analog pin on the Arduino.

For the master circuit I was integrating the different subsystems onto a single breadboard. I matched 5V with 5V and GND with GND trying to use the master rails as much as possible to leave room on the arduino and not draw too much power from it either. There were a few overlapping pins that needed to be moved on the arduino. This was easily changed thanks to the arduino API.

The basic rundown of the subsystem code goes like this; I set up the different pins and inputs and outputs, then begin the loop. In the main loop I first measure the sensor 5 times and take the average of those. Then I move the stepper motor to the final location that the sensor said to move to. Loop again.

I also integrated the code with Awadhut and the final code goes something like this; Run setup for inputs and outputs. Begin loop, and read one serial input command. IR sensor and move stepper motor. Read the pressure sensor then move Servo based on input. Calculate the velocity of the dc motor and move the pos or change the velocity of the dc motor. Loop again.

Pit Navigator

Camera Resolution + Imaging Techniques

For the Pit Navigator project, I have come up with a metric that we can use to define what set of camera and lenses that the PitRanger can use on the moon to take images of the pit. I defined this metric as pixels per square meter of opposing wall. This metric defines the level of detail on the wall, how much data needs to be stored to provide what image of the wall, and the resolution and focal length of our camera. Additionally using similar triangles, I can use our test data of modeling the pit in Utah to determine what that resolution of the opposing wall was and the quality of the results of the modeling that come from it.

3D modeling of the pit is not within our scope, so we wanted to produce our own deliverable which could be a rollout image of the pit. I discovered this pottery imaging technique and found that if we were to apply this to our own project then it would be the lowest data size to be sent back to earth of all the pictures that we take. While rollout imagery is normally used for taking images of the outside of cylindrical objects, with our moving rover we could move the camera to all the correct places and image the pit that way. This informs us that while we can see all of the pit with 2 opposing images, to negate distortion we will need to visit the the pit edge 18 times around, or every 20 degrees. This provides us with mathematical bounds that the planner will take into effect based on how close we land to the pit and how long things are taking.

Gathering Resources

There has been a significant amount of work done in this area right here at CMU, and compiling the different sources of code and robots and resources has been one of my main jobs, along with how we can interact with the different groups and what is useful and what isn't. I've been getting the local skid steer planning algorithm from Moonranger, the brinkmanship code from Neil Khera, and the simulator from the previous MRSD team.

Testing Site

The MoonRanger team has acquired another test site, Gascola. This site is available whenever we need it, under minimal supervision so we can nearly test whenever with minimal notice. This is ideal for our project because we may not know that we need to go out until the day before or need to test at night. I visited the site with them and determined 2 good cliffs that can serve as pit edges. I am also beginning the process of gaining a pass to the main facility so that our team can use the bathroom if we need it. We can also remove shrubbery and trees smaller than 3 inches around to make the test site more moon-like.

Simulation

I have been working on getting the previous team's Simulation going on another computer and am recording the steps that I take to build it. This simulation will be beneficial to testing our project before going out to the field and working on a real robot. We will be able to test our detection algorithms and planning algorithms faster than real time in sim. I have not gotten it to build quite yet and am continuing to do that.

Challenges

Motor and Sensors Lab

Stepper Motor

My first implementation was to control the speed of the motor based on the IR sensor reading, getting slower as the IR sensor reads a nearer value. I did this by increasing the time between steps proportionally to the nearing values. I realized when implementing with the GUI that the deliverable was to control the position of the stepper rather than the velocity, and changed it accordingly.
While implementing position control there was an issue with returning to 0, where after the stepper had moved away from zero for some time, the 0, that the stepper returned to was not in the same physical location. This was due to floating point rounding errors. Because the stepper motor ticks every 1.8 degrees and the input is in singular degrees that there is some information lost each time the stepper reached its goal. This was solved for the demo by converting all counting degrees to floating point values and worked reasonably well with significantly less errors. I realize now that rather than tracking degrees, I should have tracked steps as there are 200 easily identifiable steps and only converted to degrees when reporting position.

Circuit

During testing our force sensor broke and a replacement was needed. Due to the timing of the break we could not acquire one for days after. In the meantime I rewired our main circuit to operate off of the potentiometer that we had on hand to continue the software testing that was needed.

Pit Navigator

Gathering Resources

The sheer number of resources available are overwhelming, and none work right out of the box. It takes a significant effort to get these resources to build and to run on another machine when there are no instructions to do so. While these resources might be sound, unless our team can use them, they are worthless. So figuring out what to use, what to fix and what to discard has been a significant challenge.

Simulation

We have the most information on the previous team's simulation and it seems to have the most out of box potential, so I am doing my best to get that sim to build and run on our machine, however it has been quite the challenge to find and install all the hidden dependencies and file locations that need to be moved.

Teamwork

Motor and Sensors Lab

Individual	Motor	Sensor	Description
Alex W.	Stepper Motor	IR Sensor	Responsible for Stepper code, Wiring, Sensor filtering, Transfer function, Code integration
Awadhut T.	DC Motor, Servo	Force Sensor, Button	Responsible for DC code, Servo code, Button debouncing, Force threshold, Code integration
Justin M.	GUI		Responsible for learning Processing, creating gui, establishing the communication protocol

Pit Navigator

Individual	Main	Sub	Description
Alex W.	Simulation	Getting Resources, Looking for cameras	I have been responsible for gathering all the resources at our disposal and getting the simulation up and running. Looking for cameras is something I picked up as a third while the others are working with cameras
Awadhut T.	Detection	Camera control and camera mount	Determining the best way to detect pit from afar, and controlling the camera. Like different exposures
Justin M.	Get Data	Robot control	Main job is getting the different pit models and camera images, while getting access to physical robot to figure out the hooks

Plans

Pit Navigator

Testing Site

I am obtaining the pass for the testing site and will come up with the testing plans to run once we have the resources running to do so.

Simulation

I will build the simulation and get it running, then attempt to add into the sim, the different pit models that we have access to. Then add other code like Neil's brinkmanship to determine if we can stop at the pit edge.

Sensors and Motors 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.

o What is the sensor's range?

· **6 g**

o What is the sensor's dynamic range?

· **7.2 g**

o 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 capacitor is between the power and ground, which means that it is used to smooth out the power input. The Capacitor will charge when the power first flows and when the power drops, the capacitor will slowly discharge to keep the voltage high.**

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

· **If Vs =3, V = .3*g +1.5**

o What is the largest expected nonlinearity error in g?

· **.009 g, from a max of 3 g at .3% nonlinearity**

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

· **30 micro g**

o How about at 0 Hz? If you can't get this from the datasheet, how would you determine it experimentally?

· **Based on the equation given, the noise would be in micro gs over the sqrt of HZ, and if the Hz is 0 then the equation is divided by 0. The way to experimentally test this is to turn the z axis of the sensor perpendicular to gravity and take a single measurement. And the error between your measurement and 1 g is your noise.**

2. Signal conditioning

o Filtering

§ Name at least two problems you might have in using a moving average filter.

· **If you use not enough values to capture your average, it can be biased by outliers**

· **If you use too many values it can have a strong delay in showing the actual changes.**

§ Name at least two problems you might have in using a median filter.

§ **You need to read the sensor multiple times to get one value**

§ **Using a window that is too small for the frequency of noise will not get rid of the noise**

o 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 in each case: 1) which of V1 and V2 will be the input voltage and which the reference voltage; 2) the values of Rf/Ri and the reference voltage. If the calibration can't be done with this circuit, explain why.

· Your uncalibrated sensor has a range of -1.5 to 1.0V (-1.5V should give a 0V output and 1.0V should give a 5V output).

- **V1 = ref of -3V**
- **V2 = input**
- **Rf/Ri = 1**

· Your uncalibrated sensor has a range of -2.5 to 2.5V (-2.5V should give a 0V output and 2.5V should give a 5V output).

- **It seems that this calibration cannot be done on this circuit. As the scale is already 5V and the final scale is 5V then the gain must be 1. For the and there is no RF/RI and Vref combination to keep the gain 1 and offset anything but 0.**

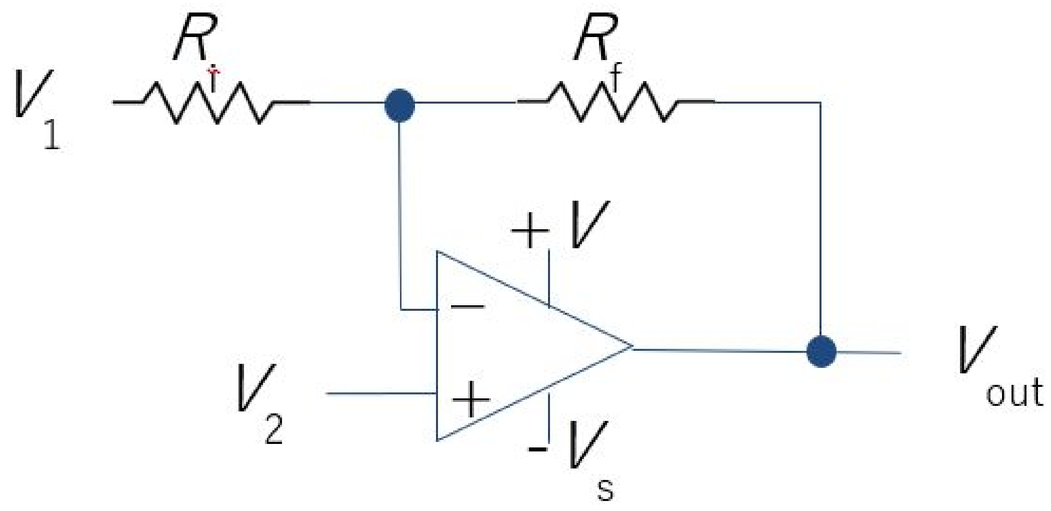

Fig. 1 Opamp gain and offset circuit

3. Control

o 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.

- **Error = e**
- **Proportion control = e*Kp**
- **Integral Control = e/s*Ki**
- **Derivative Control = e*s*Kd**

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

- **I would use the P term because it increases the response time, and how quickly and powerfully the motor responds.**

o 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?

- **I would use the I term because the integral will build up steady state error and apply a stronger and stronger force.**

o After applying the control in the previous question, if the system still has overshoot, which PID term(s) will you apply and why?

- **I would use the D term which will dampen the system, and has a significant effect on percent overshoot.**

Appendix A: Code that I wrote

```
/*
Initializing variables for stepper control.
*/
float degree;
float curDegree = 0;
const int stepperDirPin = 8;
const int stepPin = 7;
const int Irpin = A1;
const int LedPin = 13;
int mode =1;
String incoming_data = " ";

void readSer() {
  incoming_data = "";
  if (Serial.available() > 0){
    incoming_data = Serial.readStringUntil(';');
  }
}

////////////////////// . Stepper functions  ///////////////////////////

void stepAngle(){
  ////Average IR sensor/////
  int value =0;
  for(int i = 0; i<5;i++){
    value += analogRead(Irpin);
  }
  value = value/5;



  /////Print IR reading in cm/////////
  int dist = map(value, 0, 500, 150, 20);
  Serial.print("i");
  if(dist<100){    Serial.print('0');  }
  if(dist<10) {    Serial.print('0');  }
  Serial.print(dist);
  Serial.println(';');

  /////Only use Linear portion of IR sensor///////
  if      (value >500)            {  degree = 360;  }
  else if (value>180 && value<500){  degree = map(value, 180, 500, 0, 360); }
  else                            {  degree = 0; }


  ///////Change Stepper position based on IR reading//////
  float cd=curDegree-degree;
  int dir = 1;
  if (cd < 0){
    digitalWrite(stepperDirPin,HIGH);
    dir = -1;
  } else {
    digitalWrite(stepperDirPin,LOW);
    dir = 1;
  }
  for(float i = 0; i< abs(cd/1.8);i++){
    digitalWrite(stepPin,HIGH);
    delay(10);
    digitalWrite(stepPin,LOW);
    delay(10);
    ///////Print change in Degree//////
    curDegree = curDegree -1.8*dir;
    Serial.print("s");
    if(curDegree<100){  Serial.print('0');  }
    if(curDegree<10) {  Serial.print('0');  }
    Serial.print(int(curDegree));
    Serial.println(';');
  }
  curDegree = degree;
}


void setup() {
  Serial.begin(9600);
  pinMode(stepPin, OUTPUT);              // Stepper motor initializations
  pinMode(stepperDirPin, OUTPUT);
  pinMode(Irpin, INPUT);
  digitalWrite(stepperDirPin,LOW);
}

void loop() {
  /////Read in from GUI////
  readSer();
  if(incoming_data.substring(0,1).equals("t")){
    inc = incoming_data.substring(1).toInt();
//    Serial.println(inc);
  }

  /////Stepper////
  stepAngle();

}
```