Lab 2: 3D Scanner

Liz Leadley and Prava Dhulipalla

September 2017

1 Introduction

In Lab 2, we learned how to build a basic 3D scanner from servos, sensors, and an Arduino. In conjunction with this, we learned how to design a pan-tilt mechanism and write code to analyze raw data input from the Arduino. Our mechanism ended up being slightly more complicated due to an initial misunderstanding of the assignment. Here are the basic steps/mini-tasks we took to finish the 3D Scanner project.

1. Test out sensor and servos, calibrate the sensors and generate an error plot
2. Create pan-tilt mechanism
3. Do mathematical analysis on the system for the 3D scan
4. Write control code for the Arduino and code to plot the plots on a 3D map
5. 2D pan scan of the letter and 3D pan and tilt scan of the letter

2 Materials

For this lab, we required the use of the following materials (please note that some of these are approximations of values).

- Arduino Uno R3
- solderless breadboard
- GP2Y0A02YK0F IR Distance Sensor
- 2 VIGOR VS-2MB Servos
- 2 Servo motors
- 9 Male-Male jumper wires
- $144 ft^2 MDF$
- 2 4-40 Hexnuts
- 2 4-40 3/4" Screws
- tape

3 General Procedure

Before diving into the mechanical, mathematical, electrical, and software aspects of this lab, we wanted to detail the general procedure for fulfilling this lab. Some of these procedural tasks will be explained more in later sections.

To start building the 3D scanner, we started off with testing out the materials - namely, the sensors and servos. For this, we used the recommended Arduino scripts. (For more detail, see the Electrical section.)

The IR sensor does detect distances - but its distances don't correspond to literal distance measurements. In order to do that, we had to calibrate the sensor with real life readings, generate an equation, and check it against values. (For more detail, see the Electrical section.)

We then worked on the mechanical system. The lab report document specifies that we create a pan-tilt mechanism. "Pan" means that the mechanism must be able to move horizontally with reference to the global frame (along the "x-axis"), and the "tilt" means that the mechanism must be able to move vertically with reference to the global frame (along the "z-axis"). This involved designing the mechanism, fabricating, testing, iterating further, and continuing this process until we had a working system. (For more detail, see the Mechanical section.)

After the mechanism was done, we then worked on wiring the servos and sensors, and mounting them onto the system. (For more detail, see the Mechanical and Electrical section.)

We also had to make a letter to actually scan. Because of the way we designed our mechanism, we actually needed a '3D-ish' letter. So we constructed a 3D 'H' to scan, out of cardboard. (For more detail, see the Mechanical section.)

Based on the mechanical system, we had to do some mathematical analysis in order to convert the readings we receive from the sensor and map them into actual global coordinates to display our scan. (For more detail, see the Mathematical section.)

Now that we had the math behind the system and the mechanism, it was time to write code. The code was essentially two parts. One part performed the analysis that took readings from the sensor and graphed them to produce a 3D scan. This was implemented in Python. Another part was the actual control code for the servos that also took in sensor readings. This was implemented in Arduino code. These two scripts 'talked' to each other. (For more detail, see the Software aspect.)

Then, it was finally time to test out our 3D scanner. After many, many, many, (we could keep on going on here but just trust us that we counted the amount of times we tested and it reached the triple digits) attempts, we managed to get one scan that seemed to resemble our letter. (For more detail, see the Results section.)

4 Mechanical

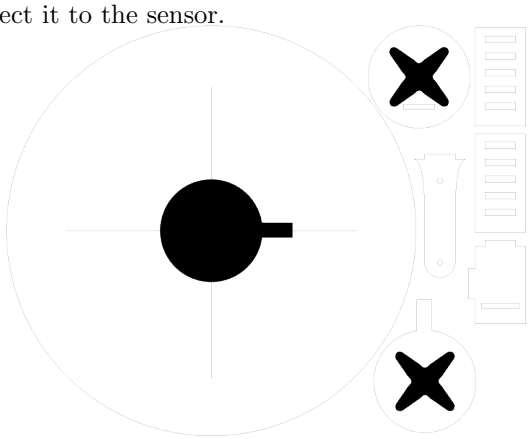
The instructions for the lab specified the creation of a pan-tilt mechanism to aid with the task of actually 3D scanning a letter. This essentially meant two axes of motion - the "x-axis" and the "z-axis" in respect to the global frame. Unfortunately, we ended up overcomplicating the system slightly which introduced a third axis of motion.

For the panning motion, we had a turntable that the object to be scanned could rest on. This would be rotated by one of the servos. This ended being a more complicated mechanism than what was necessary - by spinning the object and not actually just building a single mechanism that had both the pan and tilt motion, separate from the object, it meant we couldn't simply just use spherical coordinates. Our initial thought for doing this was actually because we had misinterpreted the assignment directions - we had thought a 3D scanner needed to be able to map a 3D object in space. In reality, it was only one "face" of a 3D object that it needed to scan. Our turnable would be able to spin the object around so that the sensor could read all of the objects 'faces' (if we had servos that could turn 360 degrees), which fulfilled what we thought what the project requirement. However, though it is a pan mechanism, it complicated the actual execution of the 3D scanner.

For the tilt motion, we went with a more basic approach. The sensor was attached to a physical component that was attached to the servo in order to move it up and down (the tilt). This was a more rudimentary and straightforward approach - instead of moving the object, we are moving the sensor, which allows us to draw more from using spherical coordinate equations.

For actual fabrication, we decided to make use of lasercut MDF pieces as well as cardboard (with necessary screws, hexnuts, etc. required to attach everything together) to build our system. The drawing of the lasercut pieces was actually done through Illustrator rather than Solidworks for the sake of simplicity. We ended up having to fabricate twice because the tolerances of the first job were loose. Though the fit was better the second time around, we still required the use of tape in order to secure the servos and sensors to the laser cut pieces.

Figure 1: The largest circle. The smaller circles attach to the servos, with the bottom one being responsible for pan and the top for tilt. The part with two circles hols the motors, and the other three parts help box in the tilt motor and connect it to the sensor.



When trying to set everything up, we noticed that there were two problems. One was that the sensor wasn't lifted high enough to actually sense the letter we had created. We ended up creating a cardboard base for the sensor-tilt aspect of our mechanism to raise it up. Our second issue was that our turntable was very heavy for the servo - when the servo was actuated, that pan aspect of the mechanism would move. This would affect the quality of our 3D scan, so we ended up taping the servo responsible for the pan motion to the table. Also to be secure, we ended up taping the letter to the turntable as well so it wouldn't move in respect to the turntable - which would also mess up the readings.

Our final deliverable for mechanical was the actual 3D letter we had to create in order to test out our system. We ended up constructing this out of cardboard. The letter we ended up choosing was not actually one of our initials, but an 'H' because it a) had an interesting shape and b) was symmetrical so it could stand on our turntable.

The letter and the cardboard are visible in the photo in our results section, as are the final laser-cut parts.

In retrospect, there are two major things we probably should have done. One was continuously refabricate until our mechanical system was up to par. Although relying on tape and cardboard is expected, we did it to such a large degree that it might have made more sense to 'go back to thedrawing board' and try to improve our mechanical component, instead of adding these additions. However, this did teach us something about fixing a design until it works - 'testing' out the fixes we would have fabricated before actually fabricating them (for instance, the stand). Secondly, although we did create a pan-tilt mechanism, it wasn't one integrated system. Although the assignment directions didn't specify an individual mechanical component, it seemed to have been implicitly understood by most, if not all, of the other groups. While our multiple mechanical components allows for more of a 3D scanning 'system,' an individual mechanism would be more of a 3D scanner. This distinction didn't seem important at the time of designing, but testing our 3D scanning 'system' brought about noticeable differences, mostly in the way we had to test.

5 Electrical

The first step for electrical was testing out the various components that we were allowed to use for the lab. This included sensors and servos. We tested out the servos by using an Arduino and connecting to a

solderless breadboard. This task was very simply - we had to connect the 5V output from the Arduino to power the servo, and connect the Arduino ground to ground the servo. Then we connected the signal wire to a digital motor in the Arduino.

After this electrical-set up, we used some of the example code the Arduino IDE provides to sweep the servo. For this, we used the "Sweep" script. From this, we learned that we essentially pass in an angle value and the servo motors move to the angle given.

Then we tested out the sensor. We powered and grounded the sensor the same way we power and grounded the servo for the servo test, but the signal wire was instead connected to an Analog port (so it can read values other than 0 or 1). We used the example script "AnalogInOutSerial" in order to test.

From this, we were able to get sensor readings. However, for a 3D scan, what we really want to get is distance, not raw sensor readings. We thus needed a calibration plot/equation to relate the sensor reading and distance. To get measurements for this calibration plot, we placed the sensor a known distance (which we recorded) away from the wall, and brought it farther away. Our bounds for the graph were initially highly arbitrary - we started recording at 0.15 m and ended at 0.5 m - but actually had more interesting implications later (essentially not realizing that our equation would be very incorrect for lower values).

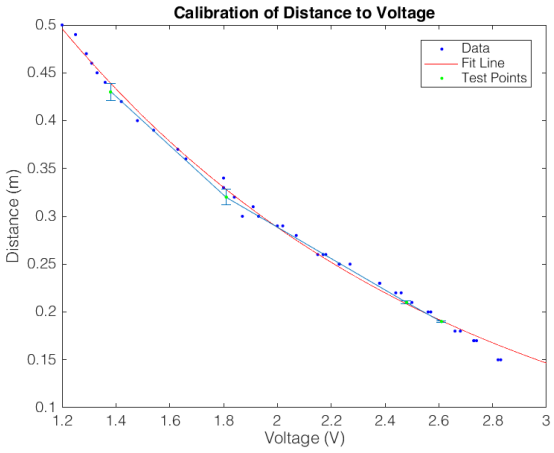
To calibrate, we converted the sensor data into voltage. The data was converted into voltage at the recommendation of a professor who informed us that it would help us easily compare the calibration to the calibration chart located in the spec sheet.

We then plotted the results of our calibration and used Matlab in order to get a fit for our equation. Our points ended up looking more like an exponential decay, and our equation reflects that:

$$Distance = 1.119 * e^{-0.6775(Voltage)}$$

To validate our equation, we took sensor readings (converted to voltage) at distances we didn't use to calibrate. We compared the distances that we tested with the distances predicted by the equation generated from the calibration plot, and generated an error plot. As can be seen from the four points we tested, the error is fairly minimal - two of the points barely have any error. The other two points have a small enough error within what we deemed acceptable. However, based on our calibration, we figured our model was more accurate around the 0.18 m to 0.25 m range (although in retrospect, we should have taken more points to more concretely determined bounds of least error).

Figure 2: Calibration and Error Plot. Notice that we seem to have a fairly good line of best fit within the range we had calibrated, especially around the lower regions. Again, we were informed that more error points would have been preferable, but the amount of points we had was fine for our purposes.

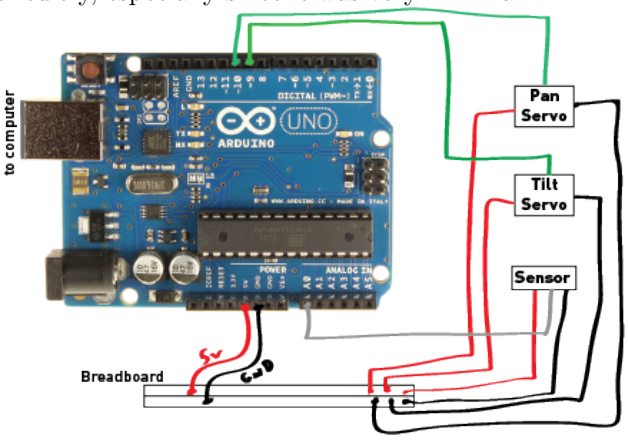


One thing to note is that we started our sensor readings at 0.1 m, going up to 0.5 m. According to what we learned from the Ninjas and the calibration sheet, if we had started the calibration plot from 0.0 m, we would have seen different behavior - not continuing the trend of an exponential decay, but actually decreasing in the voltage output shown. This is important because it informs us what distances to avoid for

the sensor - anything below about 0.15 m starts to become more inaccurate in terms of our model (even our calibration plot seems to agree with this; the lowest calibration distances stray farther away from the best fit curve).

Our overall electrical diagram essentially consisted of powering and grounding both servos, the sensor, and connecting their respective signal wires to the Arduino (we made a use of a solderless breadboard for power and grounding. Then we connected the Arduino to the laptop. Though this might seem like an easy task - we got it wrong plenty of times, especially hooking up the servos. Figuring out which one was ground, which one was power, and which one was signal was an arduous and frustrating process. However, we eventually figured it out! (And kept note of it for future wiring.)

Figure 3: Apologies for the non-Fritzing circuit diagram - we thought this might be a cooler and crisper way to represent our electrical circuitry, especially since it was very minimal

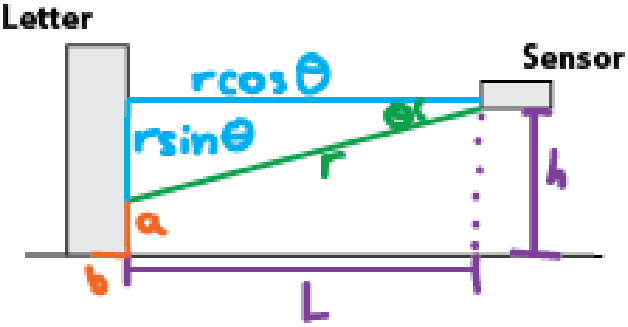


6 Mathematical Analysis

From calibrating the sensor, we now have a way to turn the output of the sensor (after it has been converted into voltage) into distance. However, this distance doesn't actually represent the 3D object in the global frame (e.g. something that actually resembles what we see). In order to map the sensor readings into the Cartesian plane, we had to use concepts from trigonometry and spherical coordinates.

If we had gone with the idea of a single pan-tilt mechanism (a servo panning the other servo while tilting the sensor) without a turntable, we would have been able to use the general spherical coordinates equations to convert distances into actual points. But the addition of the turntable and the set-up of our servos introduced something that was slightly more complex.

Figure 4: Diagram of our system set-up. Thanks to one of the Ninjas for carefully going through all the math with us so that our system could actually work! Note that the letter size isn't actually accurate - the point at which the line meets it is actual half the height.



However, mapping out the diagram of the system allowed us to generate equations to map our distance readings into Cartesian coordinates. We came up with the following equations (with the help of a Ninja):

$$x = bcos\phi$$

$$y = bsin\phi$$

$$z = h - rsin\theta$$

$$b = l - rcos\theta$$

(We're just used basic trigonometric concepts to generate the equations above, so maybe our system is not as complicated as we thought it would be.)

Phi is merely the angle of the turntable - we didn't depict it in the diagram.

However with these equations comes a caveat: in order for this to work, we had to be especially meticulous when measuring the length and height of our system. In addition, we had to make sure that our sensor, when at our "zero degree" point, was lined up with the midpoint of the letter so it was lined up exactly at the center of whatever we were scanning.

Within our equations, h and L were things we had to measure ourselves. Theta, phi, and r were the pan angle, the tilt angle, the distance reading, respectively. Therefore those values could be either sensed (as with the distance reading) or taken from how we set up the 3D scanner in code (as with the pan and tilt mechanism - e.g. saving the angle values we send to the servo).

Funnily enough, the Software section is the section that will go over the values we used/measured.

7 Software

Software was split up into two code chucks (technically there was the Matlab code we did in order to get the calibration plot, but we felt this was unneccesary to include). One is the Arduino code that actually was responsible for getting values from the servos and sensors, controlling the servos, and sending this data into Python. Python was responsible for live-plotting this data after doing the appropriate math on the raw servo and sensor data values.

7.1 Arduino Code

See the full code in the Appendix. This section will focus on the parts of the code that seem like they were the most important to the outcome of the 3D scan/to explain.

```
void setup() {

  // initializing pins
  pinMode(sensorPin, INPUT);
  myservoPan.attach(servoPanPin);
  myservoTilt.attach(servoTiltPin);

  // initializing Serial
  Serial.begin(9600);

  // starts both servos at 0
  myservoPan.write(90-20);
  myservoTilt.write(90-30);
}
```

This is the void setup() part of most Arduino scripts. First, I set up the sensors and servos appropriately. Then, I initialize Serial with a 9600 baud rate. This is important, as to communicate with Python, it needs to send data to the Serial. I also initialize with a 9600 baud rate because that is what is common, and that is also a parameter I use in the Python script. In addition, I also start the servo at certain

```
  if(Serial.available() > 0) {
    cmd_id = Serial.read();
  } else {
```

```
    cmd_id = 0;
  }
```

In Python, I have it set up where it communicates with Arduino by sending a 'start flag,' and as a result the Arduino starts scanning. This part checks whether this flag is even available or not - if it is not available, it is automatically initialized to zero to prevent any scans from happening. Again, great thanks to my friend who gave me this idea and helped me set it up (I did not think of this at all).

```
if(cmd_id == 1) {
      // pans the sensor/servo
      // delays included so it doesn't print too fast and also the servo doesn't move too
          fast
      for (panPos = 90-20; panPos <= 90+20; panPos+=1) {
        myservoPan.write(panPos);
        delay(17);

        // tilts the sensor/servo (up and down) and takes sensor readings
        for (tiltPos = 90-30; tiltPos <= 90+30; tiltPos+=1) {
          myservoTilt.write(tiltPos);
          delay(17);

          // need about 40 ms in order to accurately get the data
          data1 = analogRead(sensorPin);
          delay(40);
          data2 = analogRead(sensorPin);
          delay(40);
          data3 = analogRead(sensorPin);
          delay(40);
          data4 = analogRead(sensorPin);
          delay(40);
          data5 = analogRead(sensorPin);
          delay(10);

          sensorReading = (data1 + data2 + data3 + data4 + data5)/5*5/1023.0; // averages
              out all the data points, converts to voltage
          data = "";
          Serial.println(String(data + tiltPos + " " + panPos + " " + sensorReading));
        }
      }
  }
```

There is a lot in this code chunk, but I didn't want to split it up further in fear of losing how everything interacts with each other. First, notice that this only executes if the 'start flag' from Python is 1. Then, the turntable turns - I swept it a total of 40 degrees, which seemed to work well. I have a delay of 17 ms between the turns - this was suggested by the professors and Ninjas so that the servo doesn't turn too fast or slow, for more accurate results. Tilt undergoes the same control as pan, except I swept it 60 degrees instead. Within the Tilt control chuck, I take five data points, 40 ms between, and then average them. This was a suggestion from the professors/Ninjas - this will cause more accurate results because inaccurate errors will be 'padded' by the averaging out. If I really wanted to be fancy, I could have had more data points or I could have tried to remove the data point with the most deviation, but this averaging out sufficed (in the end, it sufficed). Finally, I display the result to the Serial Monitor seperated by a space - this is important because the space (" ") is the delimited that the Python script uses to parse the data.

The angles I chose to pan/tilt it allow minimal exposure to the wall behind and to the edges of the '3D' letter, but the trade-off is that it loses some of the scan of the actual letter. I think if I were to redo this lab, I might have let the tradeoff be in the other direction - with just filtering out the wall better and keeping the edges.

```
if(cmd_id == 1) {
      // pans the sensor/servo
      // delays included so it doesn't print too fast and also the servo doesn't move too
          fast
      for (panPos = 90; panPos <= 90; panPos+=1) {
        myservoPan.write(panPos);
        delay(17);
```

```
7
8           // need about 40 ms in order to accurately get the data
9           data1 = analogRead(sensorPin);
10          delay(40);
11          data2 = analogRead(sensorPin);
12          delay(40);
13          data3 = analogRead(sensorPin);
14          delay(40);
15          data4 = analogRead(sensorPin);
16          delay(40);
17          data5 = analogRead(sensorPin);
18          delay(10);
19
20          sensorReading = (data1 + data2 + data3 + data4 + data5)/5; // averages out all the
                data points, converts to voltage
21          data = "";
22          Serial.println(String(data + tiltPos + " " + panPos + " " + sensorReading));
23        }
24    }
```

This section is the same as the previous section! I just wanted to show the changes I made to the script - which is essentially just remove the tilt part and just have the pan part. I'm pretty sure the 2D scan was supposed to be an incremental step up to the 3D scan, however, we were initially confused on what a '2D scan' meant (we thought that was essentially what the final 3D scan is supposed to be) so we ended up writing the code for both the pan and the tilt, and then dialling down to just the pan part of it.

7.2 Python Code

See full code in the Appendix. This section will focus on the parts of the code that seemed like they were the most important to the outcome of the 3D scan/to explain. Also apparent LaTex formatting dislikes using lstlisting with Python - comments are the ones with the in front of them - so sorry about this!

```
1 cxn = serial.Serial('/dev/ttyACM1', 9600)
```

This is the line in the Python code that actually gets data from the Arduino. It needs to know the port the Arduino is connected to as well as the baud rate of the Serial Monitor. Values need to be outputted to the Serial Moniter in order for the Python script to 'grab' them.

```
1 # height and length in order to do mathematical calculations to convert the distance
      readings into Cartesian
2 h = 0.08
3 l = 0.25
4
5 # thresholds to remove noise
6 # play around with these *** mixed results with same thresholds
7 maxdis = 0.35
8 mindis = 0.15
```

This section is only important for the values they contain (all in m). According to our Mathematical Analysis section, we needed to record the height and length of our system. Here are the measurements for them. Also, in order to clean the background values, we set a min and max distance. This worked pretty well at some points, not so well at other points - the best way to clean the background was limiting how much the servo sweeps. However, some trials, these values seemed to do a very good job.

```
1 #Only runs Arduino when the keyboard shortcut "1" is pressed
2   cmd_id = int(input("Press 1 to start scan : "))
3   cxn.write([int(cmd_id)])
4   while cxn.inWaiting() < 1:
5       pass
```

I thank one of my friends for helping me with this. This is the way I can start the Arduino script by just running the Python script. This was very helpful because it prevented me from having to run the Arduino script and then quickly run the Python script.

```
# convert to floats (for precision) and radians
        theta = (dataArray[0]*math.pi)/180
        phi = (dataArray[1]*math.pi)/180
        # converted to actual distances with calibration
        r = 1.119*math.exp(-0.6775*dataArray[2])*100 #calibration equation
```

The angles I receive for the servos (theta and phi) are in degrees, and I need to convert them into radians for Python to do calculations with them. And then I find the distance to the point using the calibration equation that we found back in the 'Electrical' section. I also multiply by 100 in order to convert the result of the equation into cm, since the original equation was in cm.

```
#converting everything into Cartesian
        b = l - r * math.cos(theta)
        xpt = b * math.cos(phi)
        ypt = b * math.sin(phi)
        zpt = h - r*math.sin(theta)
```

Now that we have the angles and the distances from the previous code block, and now we have to convert it into actual x-y-z points to plot the entire scan. These equations are familar from the Mathematical Analysis section - essentially just converted into Python instead of equations.

```
if r > mindis and r < maxdis: # only within arbitrary thresholds, won't capture background
    as              much as a result
    #adds to matrices
    x.append(xpt)
    y.append(ypt)
    z.append(zpt)
```

Here you will see that I only will 'allow' the scan result to exist if the distance is within the thresholds I set. Note that this is a threshold on the distance, not any certain coordinate. If I had to redo it, I would have probably set a threshold on the x-pt so it was only between 15 and 25 cm, or something similar, which probably would have been the system less finicky.

8 Results

Once we had constructed the mechanical aspect, wired up the necessary electrical components, and wrote the software part, we were ready to test.

In the lab, they specified to start with a 2D pan scan first. Our code for this was very similar to our implementation of a 3D scanner, except we commented out the tilt portion of the code so that it would only pan (only turn the turntable). We didn't modify our Python code in any way - so it would read that the tilt angle was 0 degrees and perform calculations based on that, graphing it on the 3D plane. Our result looked like this:

Because of the way the letter 'H' is, this didn't show anything too interesting in terms of the actual letter shape - it looks like a line. But having to test out the pan mechanism first helped us a lot - it allowed us to determine at what angles we should pan in order to encompass the letter. But this makes sense - if turning the 'H' from side to side, and the sensor is set up at the middle, all it should grab is a line - nothing so interesting.

We should also note at this point that our step size was 2 degrees because of what we noticed in our initial testing of our system - it took a very long time to get the 2D and 3D plots working. Scans thereafter, even if we undid our attempt to lower the step size to one, didn't work. Although we realize that this isn't a great sign to the feasibility of our system (needing to have exact values and be in an exact position each time is very hard to do when taking multiple scans), we did understand that trying to decrease the step size just to get more points/resolution in the 3D scan was not the best use of our time.

To perform the actual 3D scan, we actuated both servos. This meant repeating the previous experiment except adding in the tilt motion as well. This was a lot harder to get working than the 2D scan, but after many trials we managed to get a 3D scan that resembled our letter H - to a very good degree (it was so surprisingly good to the point where we had actually thought we were dreaming).

Figure 5: Note that the axes are in cm - not sure why I didn't label originally.

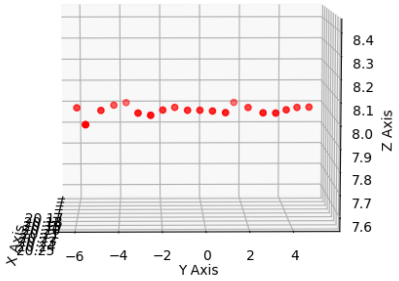
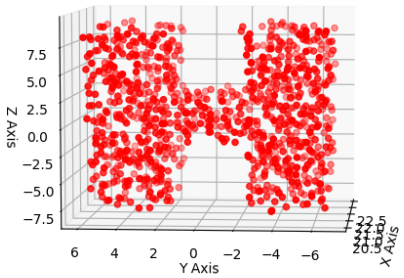


Figure 6: Note that the axes are in cm - not sure why I didn't label originally.



This was, of course, the BEST trial, and I did chose the best angle so it looked like an 'H' (other angles didn't fare so well). I also tried many, many, times, changing values as I saw fit, adjusting the system, etc. There are, of course, still points that are very random - my sensor might have been at the end of it's capacity by the end of the night, or else the averaging didn't do as well, or the math was kind of weird at points. I'm going to be honest, it's quite confusing. I expected random nothing, and I expected maybe a slight curve to the data when it finally worked - but maybe it was just pure luck. Also I guess I did explicitly choose values/a picture angle that wouldn't capture the wall at all and would minimize errors, so I shouldn't be surprised.

9 Problems and Reflection

This lab was a steeper learning curve from the previous lab, so there was expected to be more stress surrounding the completion of this lab. However, some of our choices throughout the completion of this lab further created more unnecessary stressed.

For one, us misunderstanding the assignment directions and thinking that a 3D scan meant that it would need to be a literal 3D representation of an object was heavily overscoped for what was actually required of this lab. Even when we learned the actual meaning of the assignment directions, we continued implementing our 'turntable' system, instead of listening to the Ninjas and pivoting away from that. We also probably

should have clarified with the professors, especially since we didn't think we understood how to get a full 3D scan representation of the object.

Stemming off from that, I (as in Prava, not using the group-encompassing 'we' here because this is a problem I know I definitely have) know I could have asked for help more. I did ask for help numerous times during the process, but for when I really needed help - with the code, I assumed I was very stupid and just needed to fix it myself. As I later learned from my peers, a large portion of them did have to rely on Ninja and professor help to complete their code. A task that almost killed my will to be an engineer could have been much less stressful if I had two ounces of common sense and just ask for help.

We did ask for an extension due an unfortunate illness that affected one-half of our team. However, even with this extension, we didn't manage to finish in time. It was at this point that I (again, just Prava) kind of lost confidence in my own abilities. Maybe it was because life was hitting me really hard at that point, but I felt like a failure for not completing the lab on time.

The primary problem was unfortunate timing and the consequences stemming from that. Since we both focused on our own discrete parts and then integrated them together when it was time to finish the project, there was a lot of expectations of individual work. Unfortunately, both of our external stresses seemed to spike right around the time we were supposed to come together and integrate our components. One way this could have been remedied is integrating each small component every step of the way. Although this might have been more time along the way, it would have culminated in less stress in the end.

In the future, we learned that we should heed the Ninja's advice. And I (Prava) learned that it is always better to ask for help and communicate when life gets hard. I had a legitimate excuse for why my stress levels were so high that night that I literally could not think of how to fix my system then - but instead of being forthcoming with this information, I merely internalized the stress and hoped that everything would turn out okay. Turns out - in some cases, it doesn't turn out okay.

Aside from our problems, we both learned a lot about this lab. I (Prava), didn't even really know what servos were and their usefulness. After learning what they do, I felt I had a much better grasp on thinking about mechanical systems then I did initially (I was always confused about how to design a bunch of moving parts). Within our subteams, we also learned a lot. Liz was a part of mechanical, and they learned a lot about designing and fabricating with an end goal in mind (3D scanner). I (Prava) was a part of software and learned a lot about Python-Arduino interfacing, actuating and controlling motors, effective testing strategies (reducing the step size so one scan doesn't take a long time), and making 3D graphical representations in Python. Before this lab, I had actually not really done any of these things. We both also had a better understanding about figuring out the math for a complex system - after walking through it with a Ninja, we both really understood the steps we should have taken in order to generate equations to graph the points.

10 Conclusion

In the end, we created a 3D scanner, and ended up learning a lot in the process. While the technical aspects of this lab were a real challenge, our work strategy was the real inhibitor towards progress on this lab. So - while it was cool we learned a lot about mechanical, software, and mathematical aspects of our system, the real takeaway from this lab was that there are a lot of behaviors within ourselves that we should change for a better lab/engineering experience.

11 Appendix

11.1 Arduino Code

This code controlled the servos and sensors and sent sensor and servo values to the Python script.

```
#include <SPI.h>

// creates servo objects to control a servo

#include <Servo.h>
#include <math.h>

```

```
8
9  Servo myservoPan;  // pan mechanism
10 Servo myservoTilt;  //  tilt mechanism
11
12 // defines pins
13 const int sensorPin = A0;
14 const int servoPanPin = 9;
15 const int servoTiltPin = 10;
16
17 //for interfacing with python
18 int cmd_id = 0;
19 String data = "";
20
21 // define misc. values
22 int sensorReading = 0; // average of all the data points, need volts also not raw sensor
       readings
23
24 // takes five data points in order to average them later
25 float data1 = 0;
26 float data2 = 0;
27 float data3 = 0;
28 float data4 = 0;
29 float data5 = 0;
30
31 // stores servo positions
32 int panPos = 0;
33 int tiltPos = 0;
34
35 void setup() {
36
37   // initializing pins
38   pinMode(sensorPin, INPUT);
39   myservoPan.attach(servoPanPin);
40   myservoTilt.attach(servoTiltPin);
41
42   // initializing Serial
43   Serial.begin(9600);
44
45   // starts both servos at 0
46   myservoPan.write(90);
47   myservoTilt.write(90-30);
48 }
49
50 void loop() {
51
52   if(Serial.available() > 0) {
53     cmd_id = Serial.read();
54   } else {
55     cmd_id = 0;
56   }
57
58    if(cmd_id == 1) {
59       // pans the sensor/servo
60       // delays included so it doesn't print too fast and also the servo doesn't move too
           fast
61       for (panPos = 90-20; panPos <= 90+20; panPos+=1) {
62         myservoPan.write(panPos);
63         delay(17);
64
65         // tilts the sensor/servo (up and down) and takes sensor readings
66         for (tiltPos = 90-30; tiltPos <= 90+30+10; tiltPos+=1) {
67           myservoTilt.write(tiltPos);
68           delay(17);
69
70           // need about 40 ms in order to accurately get the data
71           data1 = analogRead(sensorPin);
72           delay(40);
73           data2 = analogRead(sensorPin);
```

```
            delay(40);
            data3 = analogRead(sensorPin);
            delay(40);
            data4 = analogRead(sensorPin);
            delay(40);
            data5 = analogRead(sensorPin);
            delay(10);

            sensorReading = (data1 + data2 + data3 + data4 + data5)/5*5/1023.0; // averages
                out all the data points, converts to voltage
            data = "";
            Serial.println(String(data + tiltPos + " " + panPos + " " + sensorReading));
          }
        }
    }
    // reset cmd_id to 0
    cmd_id = 0;
}
```

11.2 Python Code

Comments in the code are denoted by the before it - apologies. This code took values from the Arduino and made a real-time 3D plot of the data.

```

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
import math
import numpy as np
import serial

# connects to correct port that Arduino is connected to
# might need to change this port depending on which port the Arduino is connected to
cxn = serial.Serial('/dev/ttyACM1', 9600)

# height and length in order to do mathematical calculations to convert the distance
    readings into Cartesian
h = 0.08
l = 0.25

# thresholds to remove noise
# play around with these *** mixed results with same thresholds
maxdis = 0.35
mindis = 0.15

# empty variables for the x, y, and z coordinates
x = []
y = []
z = []
xpt = 0
ypt = 0
zpt = 0

# empty dataArray, used for getting values from Arduino
dataArray = ["0", "0", "0"]

# creating the plot and defining it as 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

#interactive plotting, show the plot
plt.ion()

#setting labels for the plots
ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
```

```
ax.set_zlabel('Z axis')

# start count
count = 0;

while 1:

    while(count < 100): # arbitrary count threshold, can change

  #Only runs Arduino when the keyboard shortcut "1" is pressed
  cmd_id = int(input("Press 1 to start scan : "))
  cxn.write([int(cmd_id)])
  while cxn.inWaiting() < 1:
      pass

        #parsing Arduino data
        data = str(cxn.readline(), 'utf-8').strip();
        print(data) # print for debugging
        dataArray = data.split(" ") # a space (' ') is the delimiter
        print(dataArray) # print for debugging

        #convert to floats
        dataArray[0] = float(dataArray[0])
        dataArray[1] = float(dataArray[1])
        dataArray[2] = float(dataArray[2])

        # convert to floats (for precision) and radians
        theta = (dataArray[0]*math.pi)/180
        phi = (dataArray[1]*math.pi)/180
        # converted to actual distances with calibration
        r = 1.119*math.exp(-0.6775*dataArray[2])*100 #calibration equation

        #converting everything into Cartesian
        b = l - r * math.cos(theta)
        xpt = b * math.cos(phi)
        ypt = b * math.sin(phi)
        zpt - h - r*math.sin(theta)

        if r > mindis and r < maxdis: # only within arbitrary thresholds, won't capture
            background as              much as a result
    #adds to matrices
    x.append(xpt)
    y.append(ypt)
    z.append(zpt)

        #liveplots data
        ax.scatter(xpt, ypt, zpt, c='r', marker ='o') #plot point
        plt.draw()
        plt.show()

    count = count + 1;
```

12 Resources

- Professors, Ninjas, and Peers
- Communicating between Arduino and Python Has a number of helpful resources
- Matplotlib Tutorials Helped with plotting in Python and 3D plotting