PoE Lab 2: 3D Scanner

Ariana Olson, Izzy Harrison

September 2016

1 Procedure

In this lab, the goal was to take a 3d scan of a recognizable object and meaningfully represent the recieved data. We decided to Scan the letter A. To achieve this, we used an Arduino, two servos, and an infrared sensor. We designed and laser cut parts to create a pan-tilt mechanism from these components. We wrote code in Arduino to make the servos sweep across the letter, and give us the data it received.

1.1 Calibration

In order to make the data we received from the sensor useful, we first needed to calibrate it. To calibrate the sensor, we aimed the IR sensor at a wall and recorded the voltage output at distances from 20cm to 160 cm in increments 0f 5cm. This range represents the limits of accuracy from the sensor according to the datasheet. These voltage readings were taken using analogRead() on the Arduino, and printed to the serial port at the push of a button using the following code:

```
      if (buttonState == HIGH) {
        // the output is printed in to serial in the form "distance: sensorValue"
        Serial.print(distance);
        Serial.print(": ");
        Serial.println(sensorValue);

        // increment distance by 5
        distance += 5;
      }
```

To see the circuit diagram for the calibration, see figure 8 After all of the readings were taken, the voltages were manually entered into the Matlab calibration script, which fit a polynomial to the data. Because we expected to only use distances less than 100cm, we simplified the best fit to a first order polynomial, as the readings in that range are approximately linear (See Figure 1), and we were not concerned with pinpoint accuracy, more just approximations to improve the visualization.

```
% line of best fit
p = polyfit(X2, V2, 1);   %approximating the curve to be linear at the distances we are using
Fit = p(1) .* X2 + p(2);
```

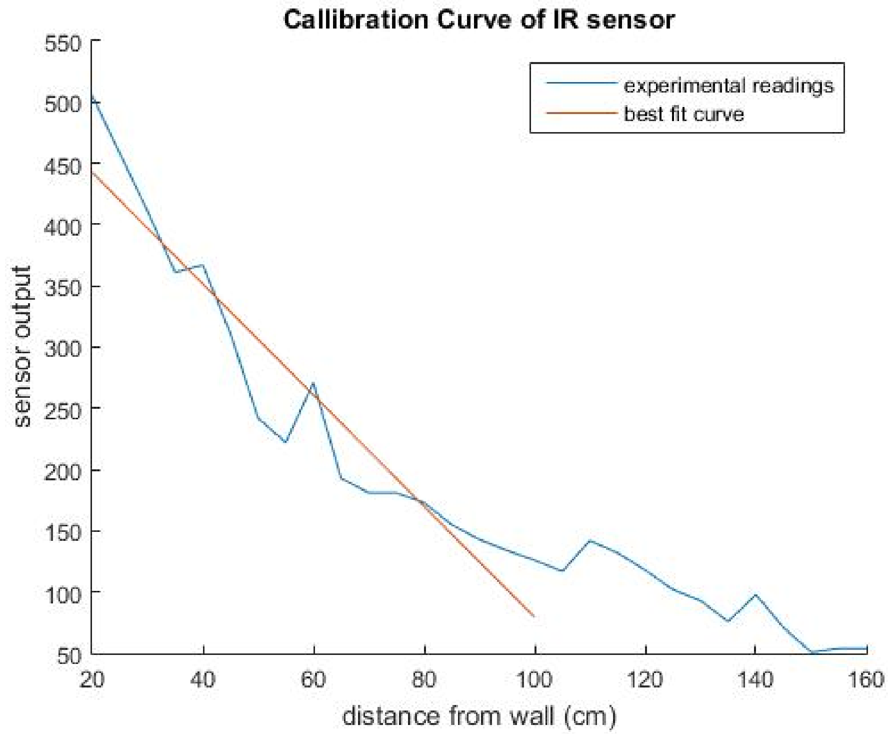

Figure 1: Because the section of the curve that was most relevant to our purposes can be approximated by a line, we fit a first-order polynomial to the data that falls under 100cm.

From the best fit line, we determined our calibration equation to be

$$d = \frac{V - 533.0294}{-4.5368} \tag{1}$$

where d is the calculated distance from the object and V is the voltage reading from the Arduino

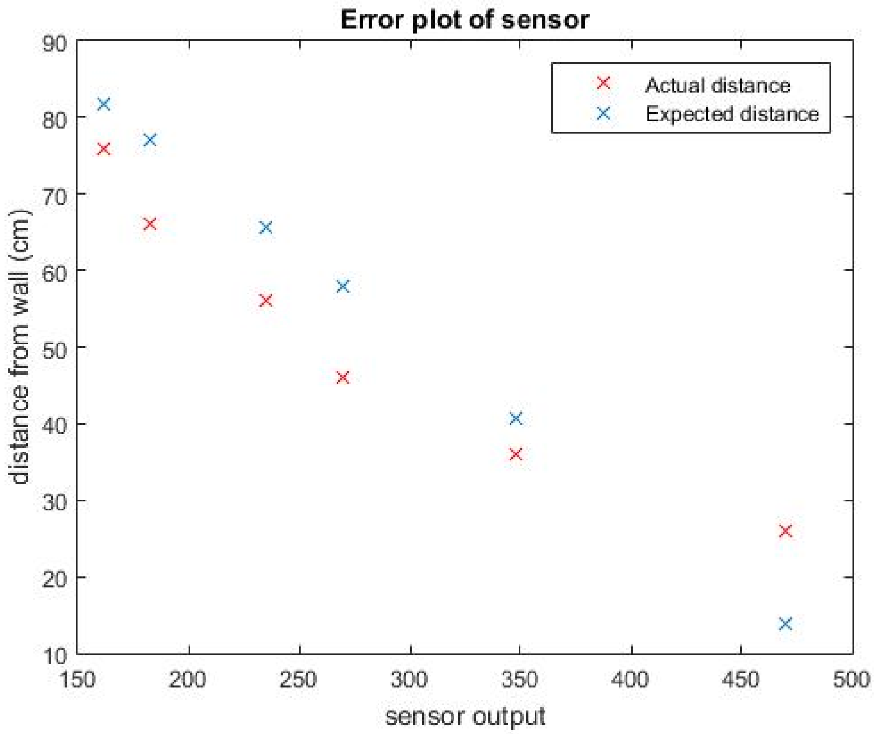

Figure 2: Calibration Test

To test the accuracy of our calibration, we took recordings of sensor values at distances we did not use to calibrate. we used the distances 26cm - 76cm with a step of 10cm. Figure 2 shows our measured distances in red, and expected distances from our line of best fit.

1.2 Pan-tilt Mechanism

1.2.1 Mechanical

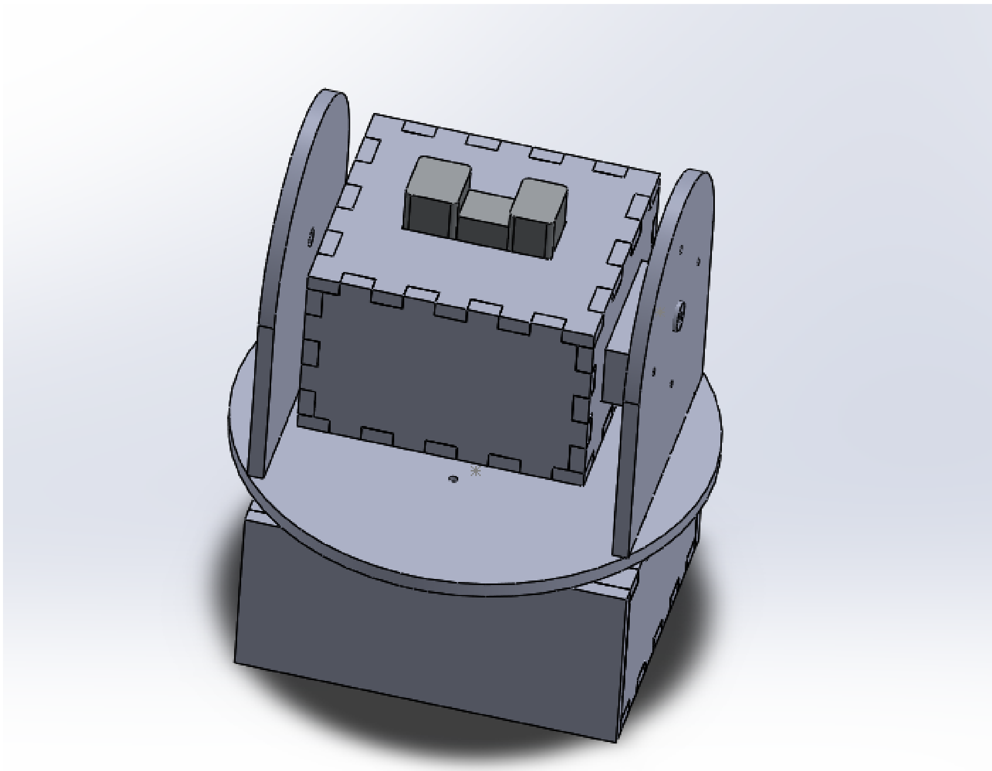

Figure 3: CAD assembly of our scanner

Our scanner is made up of a basic pan-tilt mechanism. the tilt mechanism is made up of a box containing one of our servos and our sensor, which is held up by and pivots about two parallel plats which are mounted to a circular base plate. the base plate connected to a servo on the bottom, and spins, acting as the pan part of the mechanism. The bottom servo is contained in a larger box which is the base of Theo, keeping him upright. This design is inspired by an old school security camera. The parts were laser cut, using 1/8" inch plywood. For solidworks assembly of our scanner, see Figure 3. for pictures, see Figure 4.

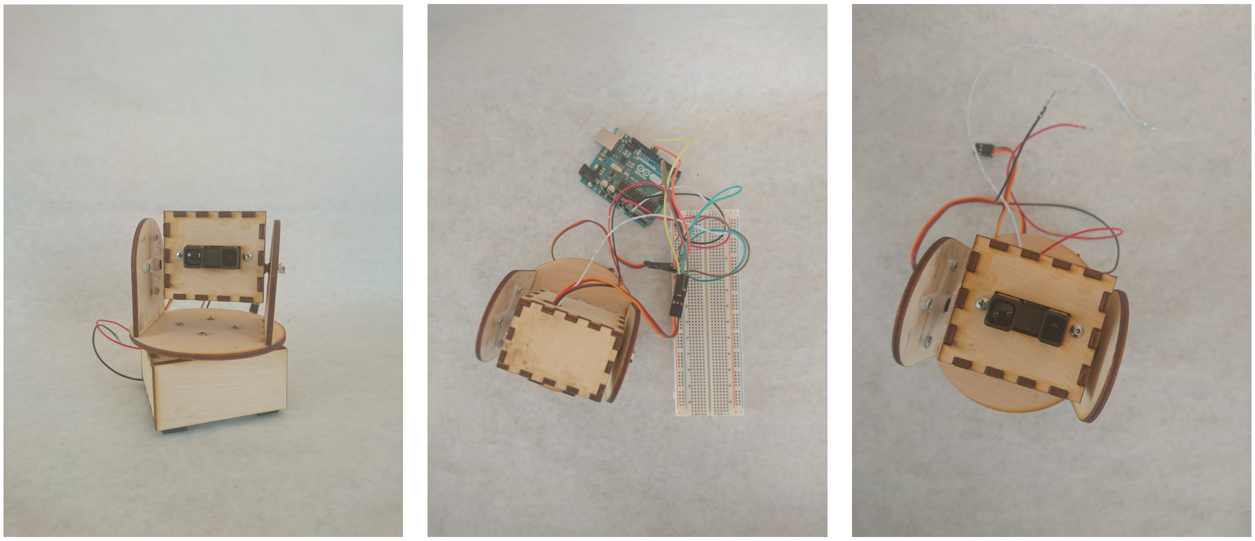

Figure 4: Theo the 3D scanner

1.2.2 Algorithm

The circuit diagram of the scanner can be found in Figure 9 under section 2. We used the built in Servo.h library in Arduino to control the servos to simplify controling the motors. The basic pattern of movement for the pan-tilt mechanism was the panning servo making one horizontal sweep, taking measurements at specified steps along the path. Once this sweep is completed, the tilting servo moves down one step, and the panning servo is reset back to its starting angle. The cycle then repeats until the scanner has swept through an entire vertical sweep. We used two for loops in our algorithm to execute this motion, and a while loop to ensure that this scanning pattern would only execute once, as can be seen below.

```
  while(tiltPos <= tiltArc){
      // move the servos in a scanning pattern.

      // tilt servo is incremented vertically downward once
      for(tiltPos = tiltStart; tiltPos <= tiltArc; tiltPos += tiltStep) {
        // pan servo is incremented horizontally once for each tilt increment
        for(panPos = panStart; panPos <= panArc; panPos += panStep) {
          // advances pan servo by one step
          panServo.write(panPos);
          delay(150);
        }

        // swings pan servo back to the starting position
        panServo.write(panStart);

        // advances tilt servo by one step
        tiltServo.write(tiltPos);
        delay(200);
      }
```

1.3 Data Collection

At each step that the scanner moves, both the panning servo angle and the tilting servo angle are printed to serial. additionally, the sensor takes three readings at each step, and prints the average of these. This is in order to reduce error in the readings and create a better final representation.

```
Serial.print(panPos); // prints pan servo angle to serial port
Serial.print(" ");  // separates data by a space on the same line
Serial.print(tiltPos);  // prints tilt servo angle to serial port
```

```
Serial.print(" ");  // separates data by a space on the same line

// take three readings at the same position
read1 = analogRead(sensorPin);
delay(10);
read2 = analogRead(sensorPin);
delay(10);
read3 = analogRead(sensorPin);
delay(10);

// prints the averaged distance reading to serial port
Serial.println((read1 + read2 + read3)/3);
```

We used Python and the Pyserial library to collect sensor data from the Arduino. As can be seen in lines 60-66 in the Arduino code and line 28-43 in the Python script, the getPositions function in the python script "handshakes" with the Arduino to signal to each other that both are ready to start. Once each program has printed a "Ready" signal to serial, The python script reads the data printed to serial from the Arduino and formats the incoming information into a list of values.

```
60    //indicate to python script that the scanner is ready to write to serial
61    Serial.println("Ready");
62  }
63
64  void loop() {
65    // check for the "Ready" signal from the Python script. This is sent when
66    // the script is run.
67    if (Serial.available()) {
```

```
28   # initialize the list
29      positions = []
30
31      while True:
32          # signals Arduino to start printing to the serial port
33          arduino.write("Ready")
34
35          # reads one line at a time from the serial port
36          s = arduino.readline()
37
38          if s == 'end\r\n':
39              # checks for ending signal from Arduino
40              return positions[1::]   #excludes initial "Ready" reading
41          else:
42              # add entries to positions list
43              positions.append(s)
```

The Arduino signals to Python that it is finished scanning by printing an "end" signal, as can be seen below.

```
// signal to Python script to stop reading from serial port
Serial.println("end");
```

```
38  if s == 'end\r\n':
39      # checks for ending signal from Arduino
40      return positions[1::]   #excludes initial "Ready" reading
```

Finally, the script formats the data into an array with the first column containing the angles of the panning servo, the second containing the angles of the tilting servo, and the third containing the voltage readings from the sensor. The array is exported to the file "data.csv" which can be loaded into Matlab for plotting.

```
for i in range(len(positions)):
    # separate each item into 3 entries of a sublist within positions
    positions[i] = positions[i].split()

# convert values to integers and separate into 3 lists
pan =[int(sublist[0]) for sublist in positions] # contains pan angles
tilt = [int(sublist[1]) for sublist in positions]   # contains tilt angles
```

```
irSensor = [int(sublist[2]) for sublist in positions]  # contains IR sensor readings

# create a numpy array of the lists for exporting
data = np.array([pan, tilt , irSensor])

# export to a .csv file
np.savetxt('data.csv', data, delimiter=",")
```

1.4 Data Interpretation

We used Matlab to manipulate the raw data collected from the Arduino and the create our representations. We attempted both converting our data, taken at angles, to a cartesian coordinate system, and also graphing the coordinates as they were with no conversion.

1.4.1 2 Dimensional representation

To test our data, we first made a quick plot of a panning only scan. As can be seen in the plot, the two legs of the "A" appear closer than the background and appear as negative spikes on the plot.

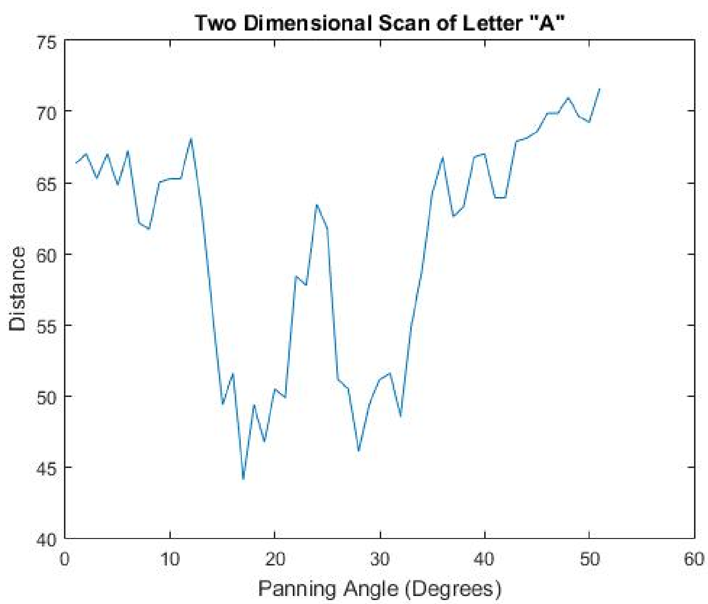

Figure 5: a 2d scan of our letter

1.4.2 Polar Data Interpretation

The next representation that we created was a 3 dimensional representation of the raw data, without converting from spherical coordinates to Cartesian. The representation was made with the Matlab function surf. The full plotting script can be found in section 4.3.3 under the Appendix. Below can be seen our 3 dimensional representation of the scanned "A".

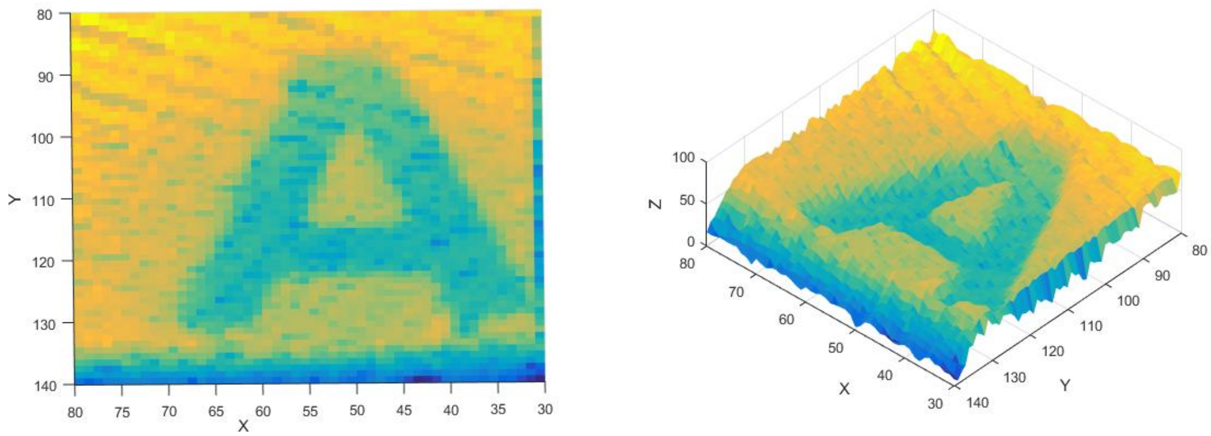

Figure 6: Surface plot of raw polar data

This representation ended up being the clearest visualization of the "A" out of the techniques we used. This is because at the angles and distances we used, the coordinates approximated Cartesian well.

1.4.3 Cartesian Data Interpretation

We also attempted to create a representation of the "A" using the same data set as above, but by converting the spherical coordinates to Cartesian using the Matlab function sph2cart as can be seen below

```
% convert to radians
panRad = pi * dataCut(1, :) / 180;
tiltRad = pi * dataCut(2, :) / 180;

% convert voltages to distance
distance = (dataCut(3, :) - p(2))./p(1);

% convert to cartesian coordinates
[x, y, z] = sph2cart(panRad, tiltRad, distance);
```

Using surf, as before, we created a plot of the data. Although the "A" is visible, this representation is much less clear and appears to be less accurate. The full plotting script can be found in section 4.3.2 under the Appendix.

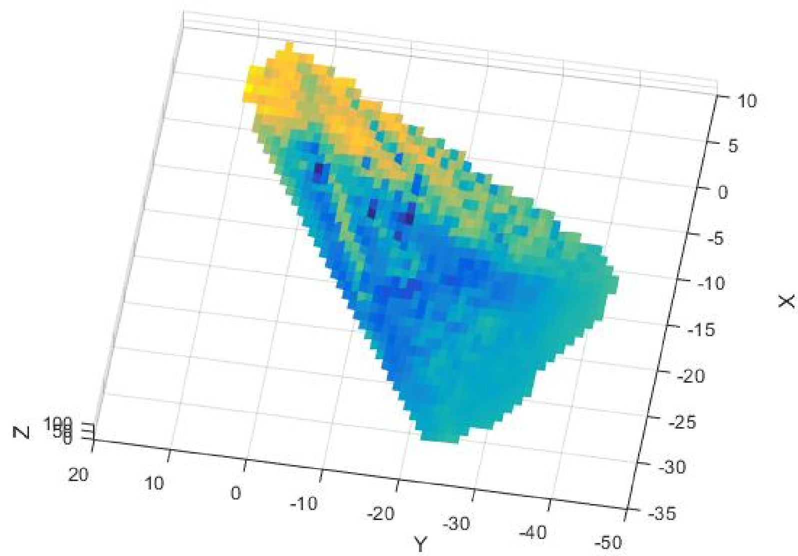

Figure 7: For this plot we converted the coordinates to Cartesian, and meshgridded them. you can see an A in it, but the shape of the graph doesn't make much sense.

2 Circuit Diagrams

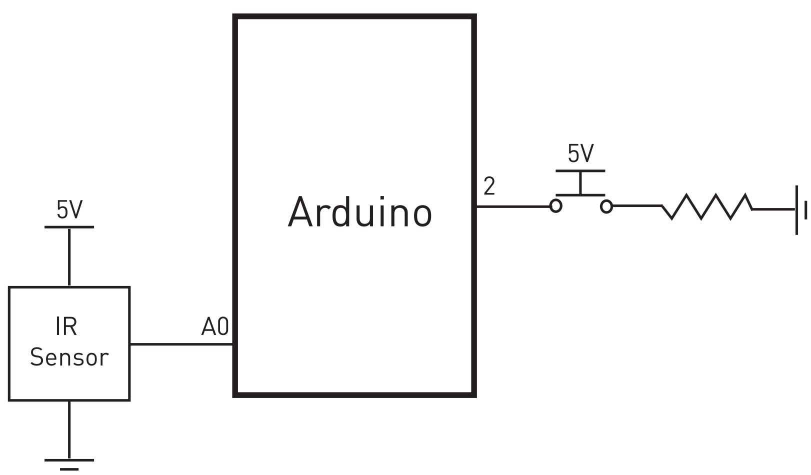

Figure 8: To calibrate our Sensor, we hooked it up to a push button to make the data collection more simple.

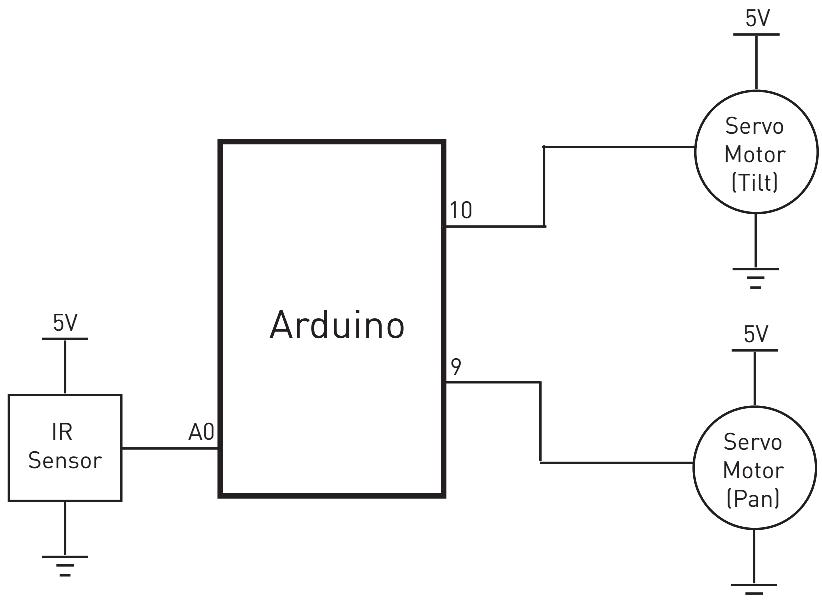

Figure 9: To operate our Scanner, we had the servos attached to digital pins 9 and 10, and the IR sensor hooked up to the A0 analog pin.

3 Reflection

In the beginning of this lab, things went very smoothly. we got ahead pretty fast, with our hardware up and running, and a rough plot of a scan. once we got that bare minimum, we struggled getting to the next step. One big issue we had was trying to plot our data in Python, and then giving up and trying to get it to Matlab. We started out by just copying and pasting it from Python to Matlab. The other big struggle we had was trying to get the data to plot after converting it from spherical to Cartesian. we spend so much time trying to achieve this, and still didn't get it quite to where we wanted to. We also had some issues with unexplained servo seizuring, and we spent most of class tuesday trying to solve this issue, until a ninja eventually told us it would be negligable. Overall, things worked out alright and we learned some cool stuff!

4 Appendix

4.1 Arduino Source Code

4.1.1 Scanner Control

```
/*
  3D Scanner
  By Izzy Harrison and Ariana Olson
  September 2016

  Controls a 2 servo pan/tilt mechanism and takes readings from a distance sensor
  a points along the way to generate 3 dimensional information about the space
  in front of the scanner

  Distance data is printed to Serial, where it can be read and collected by
  an outside script and represented as 3 dimensional data.

  This program is intended to be run alongside a Python script that uses
  pyserial to collect data from Serial. To run, upload this code, but DO NOT
  open the Serial Monitor. After code is uploaded, run the python script. The
  scanner will start to scan at the "ready" signal from the script.
 */
```

```
#include <Servo.h> // using Servo.h library for simplified control of the motors

// create servo objects to control servo motors
Servo panServo; // "pan servo" horizontal rotation
Servo tiltServo;  // "tilt servo" vertical rotation

// assign an analog pin to the IR distance sensor
const int sensorPin = A0;

// initialize variables to store servo positions
const int panStart = 30;
const int tiltStart = 80;
int panPos; // position of the panning servo
int tiltPos; // position of the tilting servo

// set limits on the angle a servo can sweep through on a full rotation (maximum 180 degrees)
int panArc = panStart + 50;
int tiltArc = tiltStart + 40;

// set step size for how many degrees a servo can rotate in between sensor measurements
int panStep = 1;
int tiltStep = 1;

// initialize sensor reading variables
// at each point, the scanner takes 3 readings which are averaged to reduce sensor error
int read1;
int read2;
int read3;

void setup() {
  panServo.attach(9); // attaches the pan servo on pin 9 to the servo object
  tiltServo.attach(10); // attaches the tilt servo on pin 10 to the servo object

  // bring system to initial position
  panServo.write(panStart);
  tiltServo.write(tiltStart);

  // set the Baud rate of the serial port
  Serial.begin(9600);

  //indicate to python script that the scanner is ready to write to serial
  Serial.println("Ready");
}

void loop() {
  // check for the "Ready" signal from the Python script. This is sent when
  // the script is run.
  if (Serial.available()) {
    // ensures that the scanner only completes one scan and does not restart the scan.
    while(tiltPos <= tiltArc){
      // move the servos in a scanning pattern.

      // tilt servo is incremented vertically downward once
      for(tiltPos = tiltStart; tiltPos <= tiltArc; tiltPos += tiltStep) {
        // pan servo is incremented horizontally once for each tilt increment
        for(panPos = panStart; panPos <= panArc; panPos += panStep) {
          Serial.print(panPos); // prints pan servo angle to serial port
          Serial.print(" ");  // separates data by a space on the same line
          Serial.print(tiltPos);  // prints tilt servo angle to serial port
          Serial.print(" ");  // separates data by a space on the same line

          // take three readings at the same position
          read1 = analogRead(sensorPin);
          delay(10);
          read2 = analogRead(sensorPin);
          delay(10);
```

```
          read3 = analogRead(sensorPin);
          delay(10);

          // prints the averaged distance reading to serial port
          Serial.println((read1 + read2 + read3)/3);

          // advances pan servo by one step
          panServo.write(panPos);
          delay(150);
        }

        // swings pan servo back to the starting position
        panServo.write(panStart);

        // advances tilt servo by one step
        tiltServo.write(tiltPos);
        delay(200);
      }
    }

    // signal to Python script to stop reading from serial port
    Serial.println("end");
  }
}
```

4.1.2 Callibration Data Collection

```
const int sensorPin = A0;
const int buttonPin = 2;
int sensorValue;
int distance = 20;

int buttonState;              // the current reading from the input pin
int lastButtonState = LOW;    // the previous reading from the input pin

long lastDebounceTime = 0;  // the last time the output pin was toggled
long debounceDelay = 50;    // the debounce time; increase if the output flickers
void setup() {
  pinMode(buttonPin, INPUT);
  Serial.begin(9600);
}

void loop() {
  sensorValue = analogRead(sensorPin);
  int reading = digitalRead(buttonPin);
  if (reading != lastButtonState) {
    // reset the debouncing timer
    lastDebounceTime = millis();
  }

  if ((millis() - lastDebounceTime) > debounceDelay) {
    // whatever the reading is at, it's been there for longer
    // than the debounce delay, so take it as the actual current state:

    // if the button state has changed:
    if (reading != buttonState) {
      buttonState = reading;

      // only toggle the LED if the new button state is HIGH
      if (buttonState == HIGH) {
        // the output is printed in to serial in the form "distance: sensorValue"
        Serial.print(distance);
        Serial.print(": ");
        Serial.println(sensorValue);

        // increment distance by 5
```

```
        distance += 5;
      }
    }
  }
  lastButtonState = reading;
}
```

4.2 Python Source Code

4.2.1 Scanner Data Collection and Formatting

```
'''
Poe Lab 2: 3D scanner
By Izzy Harrison and Ariana Olson
September 2016

Data Collection for 3D Scanner

Reads scanner data sent by Arduino over Serial, formats it into a numpy array,
and exports the array as a .csv file.

To use: upload the panTilt code to the Arduino, and then run this script.
Once the script and the Arduino have performed a "handshake", the scanner will start collecting
distance readings and writing data to the serial port until a full scan has been completed.
Ths script then closes the serial port and formats and exports the data to a .csv file that can be
used in another script (in our case a Matlab script) to visualize the data.
'''

import numpy as np
import serial

def getPositions():
    ''' "Handshakes" with the Arduino and then reads the serial port to grab
    the sensor data and servo angles (each line being a string of three values:
    "panAngle tiltAngle sensorReading").
    Returns a list of these strings.
    '''

    # initialize the list
    positions = []

    while True:
        # signals Arduino to start printing to the serial port
        arduino.write("Ready")

        # reads one line at a time from the serial port
        s = arduino.readline()

        if s == 'end\r\n':
            # checks for ending signal from Arduino
            return positions[1::]   #excludes initial "Ready" reading
        else:
            # add entries to positions list
            positions.append(s)

if __name__ == "__main__":

    print "start"

    # create an instance of the Serial class.
    # port must match serial port Arduino is writing to
    arduino = serial.Serial(port = "COM11", baudrate = 9600)

    # generate list of serial data
```

```
    positions = getPositions()

    # close the serial port
    arduino.close()

    for i in range(len(positions)):
        # separate each item into 3 entries of a sublist within positions
        positions[i] = positions[i].split()

    # convert values to integers and separate into 3 lists
    pan =[int(sublist[0]) for sublist in positions] # contains pan angles
    tilt = [int(sublist[1]) for sublist in positions]  # contains tilt angles
    irSensor = [int(sublist[2]) for sublist in positions]  # contains IR sensor readings

    # create a numpy array of the lists for exporting
    data = np.array([pan, tilt, irSensor])

    # export to a .csv file
    np.savetxt('data.csv', data, delimiter=",")

    print "done"
```

4.3 Matlab Source Code

4.3.1 Calibration

```
close all;

% callibration data (manually recorded)
X = 20:5:160;   % distances from wall
V = [505 458 411 361 367 311 242 222 271 193 181 181 173 155 143 134 126 ...
    117 142 132 118 102 93 76 98 71 51 54 54];  % arduino sensor readings
X2 = X(1:17);   %range of distances we are taking data at
V2 = V(1 , 1:length(X2));   %corresponding readings

% line of best fit
p = polyfit(X2, V2, 1);   %approximating the curve to be linear at the distances we are using
Fit = p(1) .* X2 + p(2);

% plot the line of best fit with the measured data
hold on
plot(X, V)
plot(X2, Fit)

title('Callibration Curve of IR sensor')
xlabel('distance from wall (cm)')
ylabel('sensor output')
legend('experimental readings', 'best fit curve')
```

4.3.2 Generate Plots with Spherical Coordinates

```
close all
% data(1, :) contains the angles of the panning servo at each sensor
% reading
% data(2, :) contains the angles of the tilting servo at each sensor
% reading
% data(:, 3) contains the senor readings, measured in voltage, from the IR
% sensor
load data.csv

p = [-4.5368, 533.0294];    % best fit parameters from calibration

x = data(1, :); % angles of panning servo
```

```
y = data(2, :); % angles of tilting servo
z = (data(3, :) - p(2))./ p(1); % calibrated distance readings

% create the grid that the readings will be mapped to
[X, Y] = meshgrid(min(x):max(x), min(y):max(y));

% create a matrix of the data to map to the grid
Z = griddata(x, y, z, X, Y);

% smooth out the spikes in the plot
Z = imgaussfilt(Z);

% plot the surface
surf(X, Y, Z, 'LineStyle', 'none')
xlabel('X'); ylabel('Y'); zlabel('Z')
```

4.3.3 Generate Plots with Cartesian Coordinates

```
close all
% data(1, :) contains the angles of the panning servo at each sensor
% reading
% data(2, :) contains the angles of the tilting servo at each sensor
% reading
% data(:, 3) contains the senor readings, measured in voltage, from the IR
% sensor
load data.csv

dataCut = data(:, 1:end);   %gives uo the ability to 'crop' the data
p = [-4.5368, 533.0294];    % best fit parameters from calibration

% convert to radians
panRad = pi * dataCut(1, :) / 180;
tiltRad = pi * dataCut(2, :) / 180;

% convert voltages to distance
distance = (dataCut(3, :) - p(2))./p(1);

% convert to cartesian coordinates
[x, y, z] = sph2cart(panRad, tiltRad, distance);

% create the grid that the readings will be mapped to
[X, Y] = meshgrid(min(x):max(x), min(y):max(y));

% create a matrix of the data to map to the grid
Z = griddata(x, y, z, X, Y);

% smooth out the spikes in the plot
Z = imgaussfilt(Z);

% plot the surface
surf(X, Y, Z, 'LineStyle', 'none')
xlabel('X'); ylabel('Y'); zlabel('Z')
```