POE Lab 2
DIY 3D Scanner

David Abrahams and Brenna Manning

Septembe 29, 2015

1 Introduction

We created this DIY 3D scanner by building a pan-tilt mechanism, programming an Arduino Uno to collect data from our sensor, and writing python scripts to read, save, and then visualize the data. To accomplish this, we were given two servos and an infra-red distance sensor. Using our scanner and these scripts, we can scan an object, convert the scanner data into 3D points, and then show the object in 3D space.

2 Mechanical Assembly

Two servos were provided to use for making our pan-tilt mechanism. In our design, we mounted a plate to a servo, allowing the plate to rotate horizontally. A second servo is mounted perpendicularly to this plate. We mounted two plates in an L shape to the second servo, and attached our IR sensor to the plate. When all the parts are assembled, the first servo causes the sensor to pan left and right, and the second servo causes the sensor to pan up and down. The SolidWorks assembly is shown in Figure 3 below. All designed parts were laser cut from a sheet of MDF.

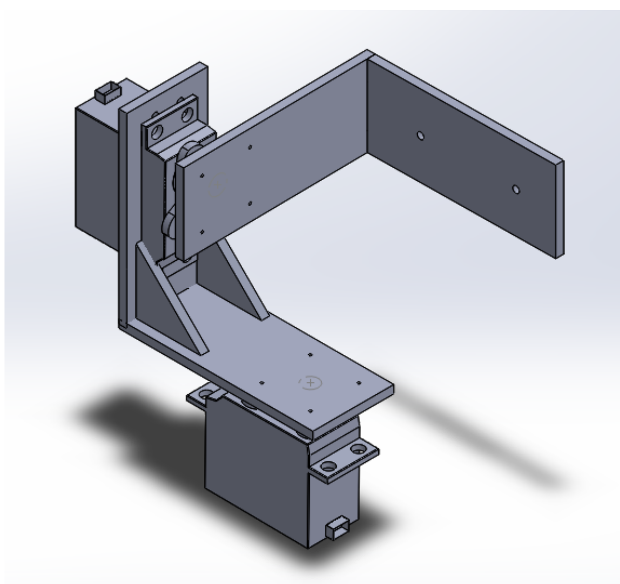

Figure 1: SolidWorks Assembly of Pan-Tilt

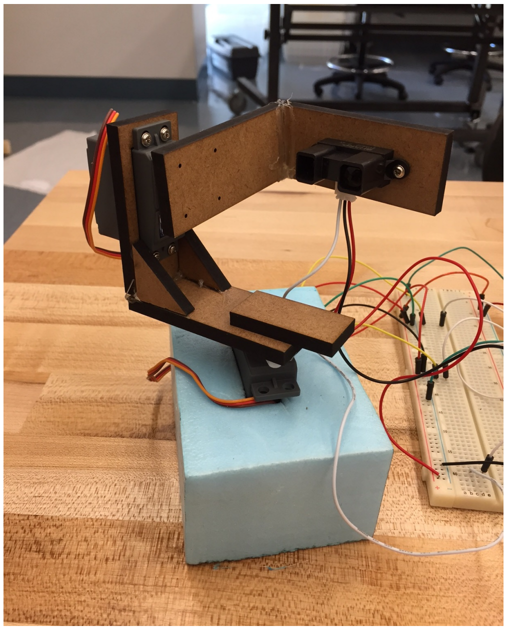

Figure 2: What We Built

3 Electrical

The two servos were each connected to power and ground. We connected the signal wires of each servo to digital output pins on the Arduino board. The IR distance sensor was also connected to power and ground, with the output wire connected to the A0 analog input pin on the Arduino.

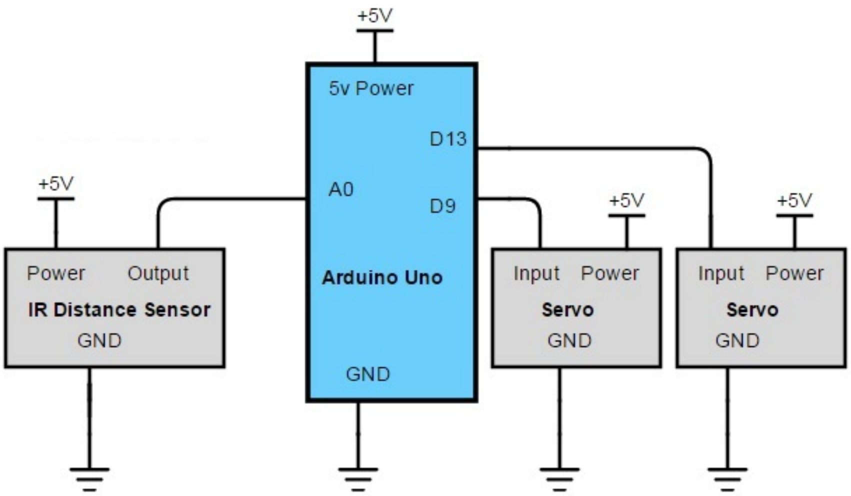

Figure 3: Schematic of Arduino, Servos, and Sensor

4 Sensor Calibration

The output detected from the infrared sensor is a unit-less value, where 1023 corresponds to a voltage of 5V. We create a list of distances in centimeters which correspond to certain voltage values, at 250mV intervals. The value detected from the sensor is converted to mV by multiplying by $\frac{5000\text{mV}}{1023}$. Next, we divide the output voltage by the interval size to choose which list element to access. Using the remainder of $\frac{output\_voltage}{interval\_size}$ we can determine the difference between the sensed value and the list value, and thus account for it using the adjacent list value. We then convert the distance being sensed in centimeters to inches. You can view our implementation in the `calibration` function in Appendix B.

Using this process, we are able to convert the unit-less output values from the sensor into distance values in inches. We tested this calibration by using the sensor to measure the distances to objects that were known distances away. This test confrimed that our calibration worked

5 Code Structure

Our code for this lab is split up among three files:

5.1 `Sweep.ino`

The first is the Arduino code, `Sweep.ino`. This script sweeps the two servos using the functions `horz_sweep` and `vert_sweep`. Both functions are broken up into two nearly identical loops, one for rotating one direction, the other for rotating back. We utilized the `millis()` function to make our delays not depend on how long it takes to probe the sensor. We also wait for half of our delay before actually reading the sensor, in order to give the servo time to rotate. Finally, we actually read the IR sensor with `analogRead` multiple times (set with the `data_read_num` variable) at each step, and average the results before sending the data to the `Serial`. We also send our loop control variables (`i` and `j`) to the `Serial`, so when we plot we can orient each distance in 3D space.

5.2 `read_arduino.py`

The second is a python script, `read_arduino.py`. This script interfaces with the Arduino by reading the `Serial`. Using our calibration, explained in Section 4, this script converts the sensor data it gets directly from the arduino into inches. This script saves this data to a file called "data.txt", which contains lists of distances the sensor detected, along with the position of each servo at the point that distance was measured (the `i` and `j` variables from `Sweep.ino` in Appendix A). If an invalid input is ever received, the `distances` variable is cleared, in case all previously received data was corrupted (this can happen when the Servo lurches, for example).

We used multithreading in this script. The process that asks the user to start and stop reading input and the process that reads data off the Serial are two separate python threads. This allows the user to stop the program at any time.

5.3 `plot.py`

The third file is `plot.py`. We define the angle ranges the sensor sweeps across at the top. The function `read_data` takes the file name of the file generated by `read_arduino.py` as an input, reads the scanner data, creates an array of data points, and returns it. The `get_angles` function takes in an array of indices (which will be servo positions `i` and `j` from `read_arduino.py`), the minimum and maximum angle the servo sweeps between, and number of steps as inputs. It returns the angle the sensor is facing. The function `get_cartesian` takes in the distances and servo positions and calculates the angles using `get_angles`. The output of this function is the data points in Cartesian coordinates using spherical to Cartesian math. The `matplotlib` function takes the Cartesian X, Y, and Z coordinates as inputs and then creates a 3-dimensional plot using Matplotlib. This plot is a visualization of what we scanned created from the data points. The function `plot_heat.py` essentially "flattens" the 3D point into 2D, and represents y, which is the depth of the object, as the color. This is done using 3D interpolation in the `matplotlib` library.

6 Results

We constructed a three-dimensional letter D, shown in Figure 4. We then scanned this object. Using `plot.py`, we plotted our results in 3D space and obtained the graph in Figure 5. If we flatten this graph onto the X-Z plane and interpolate between points, we obtain Figure 6. This heat map shows the shape of the object we scanned, with the blue area representing points detected that were nearest to the scanner.

Figure 4: The object we scanned.

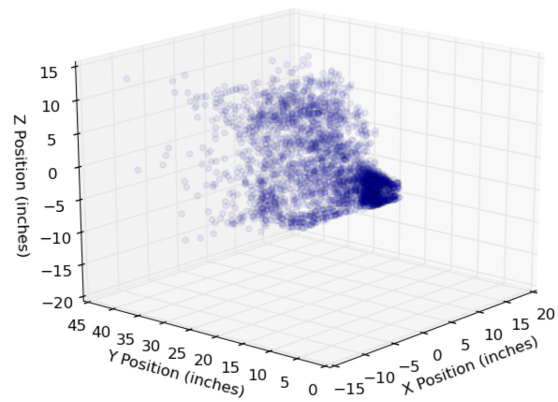

Figure 5: Our 3d scatter plot.

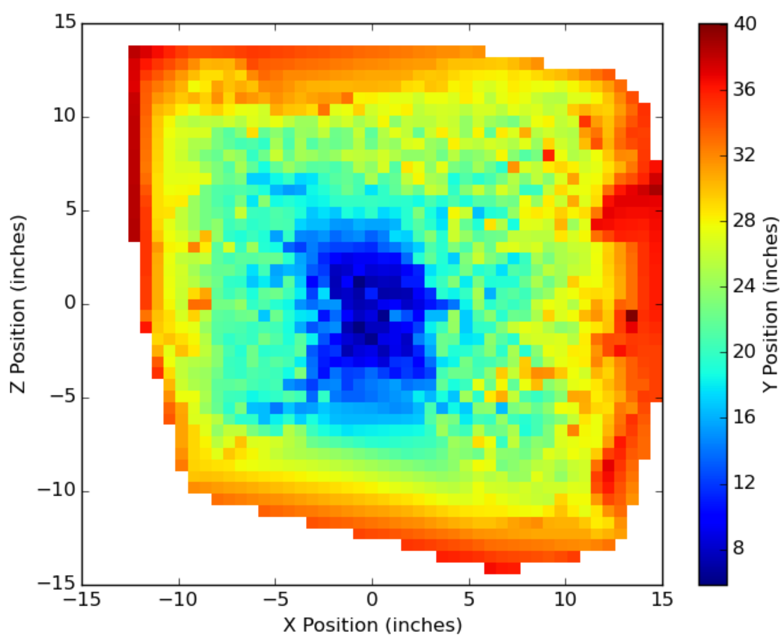

Figure 6: The distances projected onto the X-Z plane. Y value is represented by color.

A Sweep.ino

```
#include <Servo.h>

int sensorPin = A0;  // select the input pin for the IR sensor
int sensorValue = 0;  // variable to store the value coming from the sensor

// create servo objects to control the servos
Servo horz_servo;
Servo vert_servo;

// variables to store the servo positions
int h_pos = 0;
int v_pos = 0;

// How wide the servos should rotate
byte h_degrees = 45;
byte v_degrees = 45;

// The number of steps the servos take per sweep.
byte h_points = 40;
byte v_points = 40;

// How many degrees each step should be
float h_step_width = (float) h_degrees / h_points;
float v_step_width = (float) v_degrees / v_points;

// How long each step should take.
byte h_delay = 40;
byte v_delay = 40;

// How many times we should read the IR sensor at each step. The higher this
// number, the more accurate the result. This could potentially slow the scan
// however.
byte data_read_num = 5;

// variable to store the current time
unsigned long time;

void setup()
{
  horz_servo.attach(9);  // attaches the servo on pin 9 to the servo object
  vert_servo.attach(13);  // attaches the servo on pin 13 to the servo object
  Serial.begin(9600);
}

// going_down is 1 on way down, 0 on way up. This function sweeps the
// horizantal servo
void horz_sweep(byte j, byte going_down)
{

  if (j % 2 == going_down)
  {
    for (int i = 0; i < h_points; i += 1)
    {
      time = millis();  // register current time
      h_pos = i * h_step_width;  // set servo to new position
      horz_servo.write(h_pos);
      delay(h_delay / 2);  // sleep for half of the wait time to give the servo
      // time to move before we read in the data

      // probe the sensor multiple times and average the results
      int temp_Value = 0;
      for (byte k = 0; k < data_read_num; k += 1)
        temp_Value += analogRead(sensorPin);
      sensorValue = temp_Value / data_read_num;
      // print to the serial and sleep the remaining time
```

```
      Serial.println(String(sensorValue) + ", " + String(i) + ", " +
                     String(j));
      delay(h_delay - (millis() - time));
    }
  }

  else
  {
    // If i is a byte here, the servo does strange things. We do not know why
    // yet
    for (int i = h_points - 1; i >= 0; i -= 1)
    {
      time = millis();
      h_pos = i * h_step_width;
      horz_servo.write(h_pos);
      delay(h_delay / 2);

      // probe the sensor multiple times and average the results
      int temp_Value = 0;
      for (byte k = 0; k < data_read_num; k += 1)
        temp_Value += analogRead(sensorPin);
      sensorValue = temp_Value / data_read_num;
      // print to the serial and sleep the remaining time
      Serial.println(String(sensorValue) + ", " + String(i) + ", " +
                     String(j));
      delay(h_delay - (millis() - time));
    }
  }
}

// this function sweeps the vertical servo down then up, while sweeping the
// horizantal servo at each step
void vert_sweep()
{
  for (int j = 0; j < v_points; j += 1)
  {
    v_pos = j * v_step_width;  // set servo to new position
    vert_servo.write(v_pos);
    horz_sweep(j, 1);  // sweep the horizantal servo
    delay(v_delay);
  }

  for (int j = v_points - 1; j >= 0; j -= 1)
  {
    v_pos = j * v_step_width;  // set servo to new position
    vert_servo.write(v_pos);
    horz_sweep(j, 0);  // sweep the horizantal servo
    delay(v_delay);
  }
}

void loop()
{
  vert_sweep();
}
```

Sweep.ino

B read_arduino.py

```
import serial
import threading
import json
import math
import os
```

```
import sys
toplevel_dir = os.path.join(os.path.dirname(__file__),
    os.path.pardir,
    os.path.pardir)

filename = os.path.join(toplevel_dir, "data", "data.txt")

running = True

referenceMv = 5000
interval = 250  # mV
distance_list = [150, 140, 130, 100, 60, 50, 40, 35, 30, 25, 20, 15]  # distance in cm for each 250 mV

class StopEvent:

    def __init__(self):
        self.is_set = False

    def set(self):
        self.is_set = True

def to_millivolts(val):
    return int(val * referenceMv / 1023)

def calibration(mV):
    """
    >>> calibration(0)
    150.0
    >>> calibration(240)
    140.4
    >>> calibration(3500)
    15
    >>> calibration(260)
    139.6
    """
    index = mV / interval
    if index >= len(distance_list) - 1:
        centimeters = distance_list[-1]
    else:
        frac = (mV % interval) / float(interval)
        centimeters = distance_list[index] - ((distance_list[index] - distance_list[index + 1]) * frac)

    return centimeters

def to_inches(cent):

    return cent / 2.54

def get_angles(indices, anglemin, anglemax, steps):
    angle = anglemin + (float(indices) / steps) * (anglemax - anglemin)
    return angle

def read_arduino(distances, ser, stop_event):
    # read the arduino until the user tells it to stop
    while not stop_event.is_set:
        try:

            line = ser.readline()
            # split it into a list of ints
            vals = [int(s.strip()) for s in line.split(',')]

            # convert the distance to inches
            vals[0] = to_inches(calibration(to_millivolts(vals[0])))
            print "Adding " + str(vals) + "..."

            # this checks for funky input.
```

```
            if len(vals) == 3:
                distances.append(vals)
            else:
                raise ValueError()

        except ValueError:
            del distances[:]
            print "Invalid input received. Clearing data."
        except SerialException, OSError:
            pass
            # Just keep chuggin.

def save_data(distances, fn):

    # If the filename's parent folder doesn't exist, create it
    if not os.path.isdir(os.path.dirname(fn)):
        os.makedirs(os.path.dirname(fn))

    # save the data to the file!
    with open(fn, 'w') as outfile:

        json.dump(distances, outfile)

def main():

    # initialize the data and stop event
    distances =[]
    stop_event = StopEvent()

    # ask the user to start the program
    s = ''
    while s.lower() != 's':
        s = raw_input('Press s to start! --> ')

    # start reading in data off the arduino
    ser = serial.Serial('/dev/ttyACM0', 9600)
    t = threading.Thread(target=read_arduino,
                         args=(distances, ser, stop_event))
    t.start()

    # allow the user to stop at any time0
    q = ''
    while q.lower() != 'q':
        q = raw_input('Press q to quit! --> ')
    stop_event.set()

    # after we've stopped, save the data to a file.
    save_data(distances, filename)


if __name__ == '__main__':
    main()
```

read_arduino.py

C plot.py

```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import json
import os
from numpy import sin, cos, radians
from matplotlib.mlab import griddata
from matplotlib.colors import LogNorm
```

```python
# import seaborn as sns

toplevel_dir = os.path.join(os.path.dirname(__file__),
    os.path.pardir,
    os.path.pardir)

filename = os.path.join(toplevel_dir, "data", "data.txt")
v_points = 40
h_points = 40

h_deg_center = 90.0
v_deg_center = 90.0

h_deg_range = 45.0
v_deg_range = 45.0

camera_distance_from_center = 1.25  # inches

thresh_distance = 50  # inches
cut_off = 17

# Due to how our servos and sensor are oriented, at a high index the sensor
# looks down and left
h_degrees_min = h_deg_center - h_deg_range / 2
h_degrees_max = h_deg_center + h_deg_range / 2
v_degrees_min = v_deg_center - v_deg_range / 2
v_degrees_max = v_deg_center + v_deg_range / 2


def read_data(fn):
    #takes in file name (ex: data.txt)
    #reads scanner data
    with open(fn, 'r') as file_obj:
        scanner_data = json.load(file_obj)
    data_points=np.array(scanner_data)

    return data_points


def get_angles(indices, anglemin, anglemax, steps):
    """
    >>> get_angles(np.array([28]), h_degrees_min, h_degrees_max, h_points)
    array([ 99.])
    >>> get_angles(np.array([9]), v_degrees_min, v_degrees_max, v_points)
    array([ 77.625])

    """
    angles = anglemin + (indices.astype(float) / steps) * (anglemax - anglemin)
    return angles

def get_cartesian(h_pos_servos, v_pos_servos, distances):
    # get the angles from servo positions
    h_rads = radians(get_angles(h_pos_servos, h_degrees_min, h_degrees_max, h_points))
    v_rads = radians(get_angles(v_pos_servos, v_degrees_min, v_degrees_max, v_points))

    # adjust for camera offset
    distance_adj = distances - camera_distance_from_center

    x = distance_adj*sin(v_rads)*cos(h_rads)
    y = distance_adj* sin(v_rads)*sin(h_rads)
    z = distance_adj*cos(v_rads)

    return np.array([x, y, z]).T


def plot_points(x, y, z):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
```

```
    ax.scatter(x, y, z, alpha=0.07)
    ax.set_xlabel('X Position (inches)')
    ax.set_ylabel('Y Position (inches)')
    ax.set_zlabel('Z Position (inches)')
    plt.show()

def plot_heat(x, y, z, log=False):
    xi = np.linspace(-15, 15, 50)
    zi = np.linspace(-15, 15, 50)
    yi = griddata(x, z, y, xi, zi, interp='linear')

    if log:
        plt.pcolor(xi, zi, yi, norm=LogNorm(vmin=yi.min(), vmax=yi.max()))
    else:
        plt.pcolor(xi, zi, yi)
    cbar = plt.colorbar()

    ax = plt.gca()
    ax.set_xlabel('X Position (inches)')
    ax.set_ylabel('Z Position (inches)')
    cbar.set_label('Y Position (inches)')

    plt.show()

def plot_bool(x, y, z):
    xi = np.linspace(-15, 15, 50)
    zi = np.linspace(-15, 15, 50)
    yi = griddata(x, z, y, xi, zi, interp='linear')
    yi = yi <= cut_off

    plt.pcolor(xi, zi, yi)
    cbar = plt.colorbar()

    ax = plt.gca()
    ax.set_xlabel('X Position (inches)')
    ax.set_ylabel('Z Position (inches)')
    cbar.set_label('Y Position (inches)')

    plt.show()

def main():
    #Creates 3d plot from data points
    points = read_data(filename)
    distances = points[:,0]
    h_pos_servo = points[:,1]
    v_pos_servo = points[:,2]
    cartesian= get_cartesian(h_pos_servo, v_pos_servo, distances)
    cartesian = cartesian[cartesian[:, 1] <= thresh_distance]
    x, y, z = cartesian[:, 0], cartesian[:, 1], cartesian[:, 2]
    # plot_bool(x, y, z)
    plot_points(x, y, z)

if __name__ == '__main__':
    main()
```

plot.py