Lab 2: DIY 3D Scanner

ENGR2210: Principles of Engineering

Byron Wasti | Luke Morris

February 18, 2015

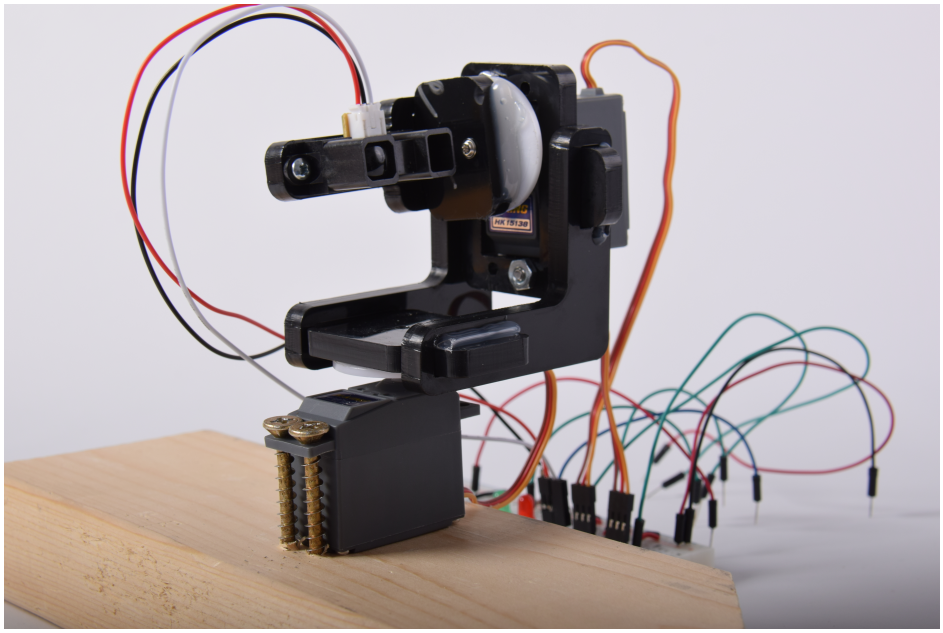

Figure 1: Scanner

1 Introduction

In this lab we used an infrared distance sensor and two hobby servo motors in order to scan a three dimensional object. Specifically, we scanned the first letters of one of our names: L. The mechanical system consists of a pan/tilt mechanism built of laser cut delrin. The electrical system consists of an Arduino used to collect physical data and a computer to process data for graphical representation.

2 Video

Click here to view our 3D scanner working.

3 Design

We used a delrin sheet we found in the POE lab to build the pan/tilt mechanism.

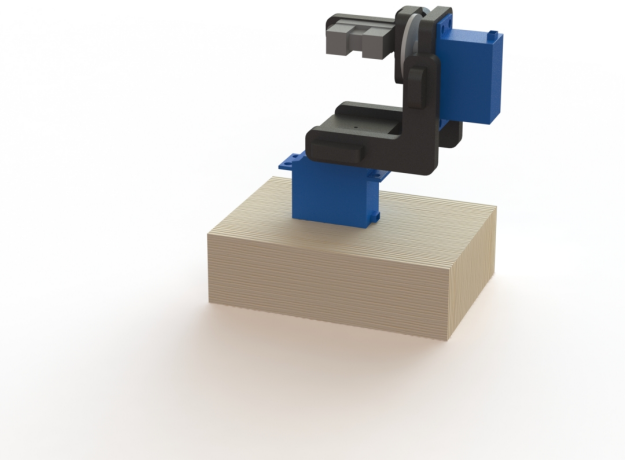

Figure 2: Render

There are six delrin pieces joined with mortise and tenon press fit joints and hot glue. We used the accompanying servo hardware to fasten the circular servo horn to the servo output. The horns were then hot glued to delrin plates with pre-drilled holes to ensure alignment. The distance sensor is fastened to a delrin arm with screws. The entire assembly is fixed to a 2x4 block with wood screws through the bottom servo mounting holes.

To make post processing as easy as possible, it was important for us to keep the axes of rotation of the pan tilt mechanism aligned at all times. This allows us to do a straight conversion from spherical coordinates to Cartesian coordinates for plotting without any complex geometry. We achieved this by using SolidWorks sketches and equations during design to make sure all components updated to maintain alignment when changes were made.

4 Calibration

We mapped the distance sensor output voltage to a range of distances in 1cm increments from 16cm to 52cm. We did this using a ruler and a flat piece of wood which we moved away from the sensor slowly.

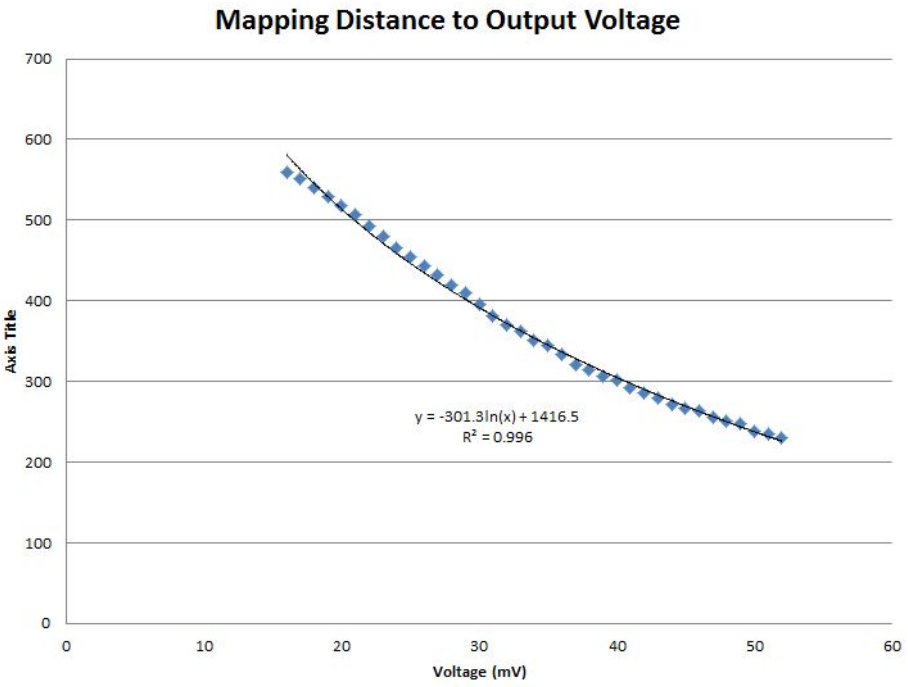

Figure 3: Calibration Plot

Using Microsoft Excel, we fit the data with a logarithmic equation. However, since the data has distance on the Y axis, the actual equation is exponential. Voltage maps to distance by the equation:

$$D = e^{\frac{v - 1416.4651589499}{-301.2902022911}}$$

4.1 Error Analysis

To verify the mapping function, we moved the entire setup to a different location (back of the classroom) and took a series of measurements with the IR sensor. We recorded the actual distance with a ruler to a sheet of plywood and the output voltage of the sensor at each location. We then calculated the measured distance with the IR sensor using our mapping function.

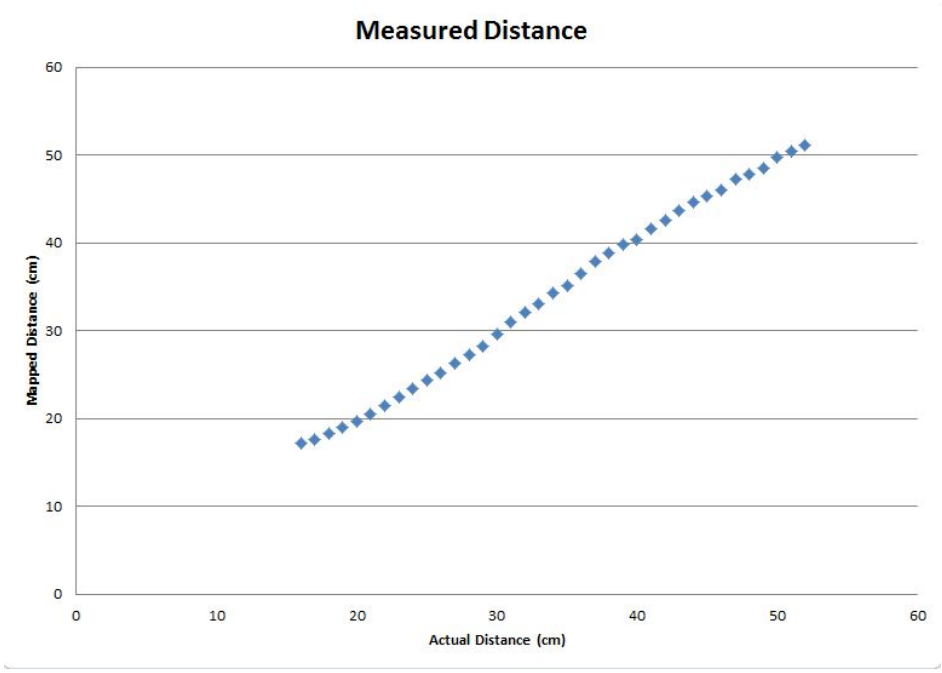

Figure 4: Mapped Distance vs. Actual Distance

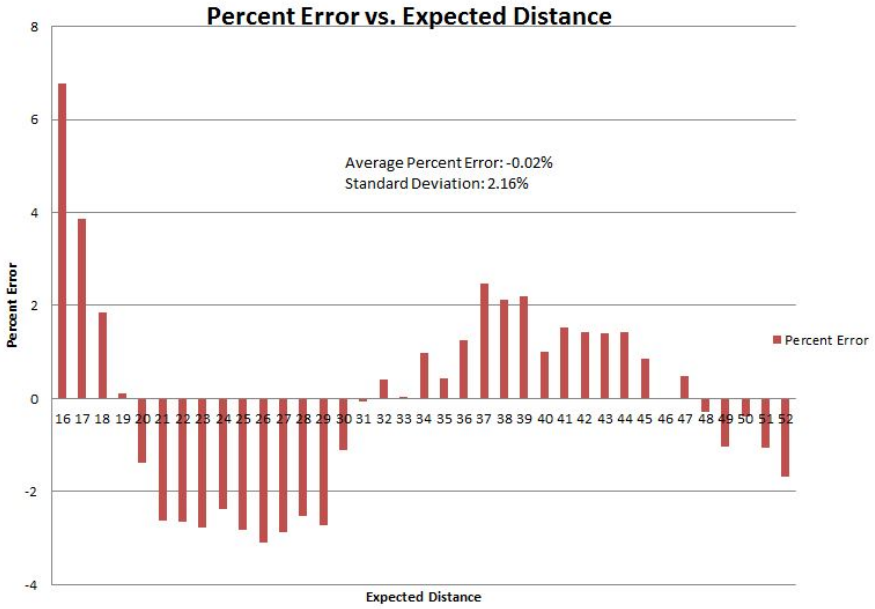

Figure 5: Mapping Percentage Error

Figure 5 shows the percent error from our new measurements and the actual distance to the board. The average percent error is essentially zero at -0.02%. The standard deviation of the error is 2.16%. The maximum error occurs closest to the sensor and is 6.77%. This is a relatively small error and therefore we believe that our mapping function is sufficiently accurate to do a 3D scan.

5 Circuit

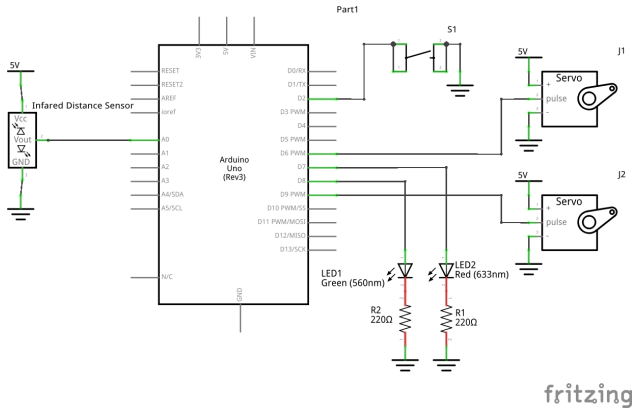

Figure 6: Circuit diagram

The circuit for our 3D scanner is fairly simple, with the addition of a kill-button which stops the 3D scanner and graphs whatever data had been collected so far, and two debugging lights. The debugging lights correspond to the pinging, red being ping and green being pong. If both lights turn off, then either the system is hanging or something went wrong.

6 Code

6.1 Introduction

The goal of our code is to control the servo motors, take data, and plot the data in 3D. There are many options available, however we imposed upon ourselves the goal of ensuring that there is no data loss and that there is the ability to have live 3D plotting. This section will first go over the interfacing between the computer and the Arduino, and then will go over the individual code.

6.2 The Ping

One simple way of communicating between an Arduino and a computer is to have the Arduino spam the serial port of the computer and have the computer read the serial input at any interval. This method works to a certain extent, as the Arduino Uno's output buffer holds 64 bytes of information and, at least in our case, we are outputting 9 bytes for each data point (in the format distance,angle1,angle2).

What this means is that by just spamming the serial port, the Arduino could be ahead of the computer by about 7 data points, which in most instances is fine. However, attempting to implement live 3D plotting slows the computer down dramatically, and therefore spamming the port is no longer an option. So we decided to set up a ping between the computer and the Arduino, such that the Arduino could never get ahead of the python code.

The ping part of the code can be seen in the python code as:

```
s.write('a')
tmp = s.readline()
```

with the corresponding Arduino code being:

```
incomingByte = Serial.read()
...
Serial.print(sensorValue);
Serial.print(",");
Serial.print(bpos);
Serial.print(",");
Serial.println(tpos);
```

The Arduino pauses until it can read something in the input buffer, and then sends out the data. This means that there is no data loss throughout the entire process, even if the computation of plotting 3D in real time takes a while. The process can be improved by actually using the output buffer of the Arduino to it's fullest extent, and allowing for 7 data transmissions to be stored there before the Arduino pauses. This could be implemented in future iterations.

6.3 Arduino

We used an Arduino script to manually move the servos to determine the necessary sweeping range for the servos. We did this to ensure the swept range of the servos would encompass the letter but also not scan too much of the surrounding environment to save time. We also used this code to see a live feed of the distance sensor output voltage to map from voltage to distance in centimeters for plotting.

The data collection code consists of sweeping the two servos between their respective minimum and maximum angles (determined using above mentioned code) in 1 degree increments. The sweep servo moves through the entire range, then the pan servo moves up one degree, and the sweep servo moves back along its range. At each increment, the Arduino takes 5 IR readings with a pause of 15 milliseconds. It returns the average of these values.

6.4 Python

6.5 Introduction

Where the Arduino handles reading the IR distance sensor and the servos, the python code running on the computer handles everything else. When the program first starts a blurb pops up waiting to get the go-ahead to try and connect to the Arduino. If the connection is deemed successful, and the computer and Arduino are pinging happily together, the program begins to run. With live plotting that can be turned on and off, and the ability to stop the code at any time and still plot all of the data that had been collected, the python code makes working with the 3D scanner a breeze.

6.6 Plotting and Filtering

Plotting was handled by the MatPlotLib python library, which allows for easy 3D plotting. While the program is running there is a live 3D plot with a few optimizations to make it easier to process, and once the program is done a better version is plotted which allows for us to pan around the 3D scatter plot.

One of the largest issues was correcting for errors in our collected data, one of which was a slight slant in all of the 3D plots along the Y-axis. This was fixed by adding a correction factor to our spherical-to-cartesian equations for the Y-axis (in the form of an extra 8 degrees being added to the phi angle). The other errors that are apparent is the random noise that is collected. We decided to filter out such noise by comparing each point's relative distance to every other point,

and then cutting out any outliers. Of course this is computationally expensive, and we only use such post-processing on data that has the background already filtered out.

7 Results

7.1 One Dimension

Moving iteratively, we then took a one dimensional scan of a small sheet of plywood. Because we had already written the functionality to do a 3D scan, we simply tilted our entire mechanism up and took the first pan sweep of data.

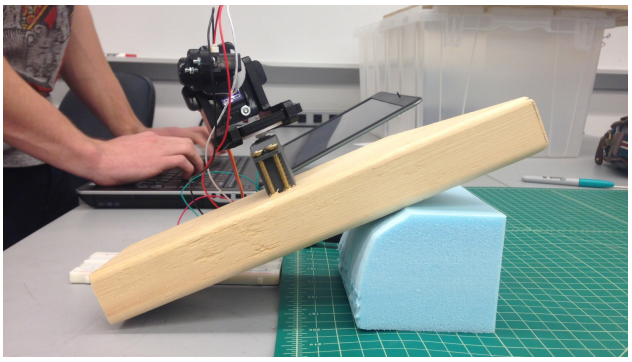

Figure 7: 1D Set-Up

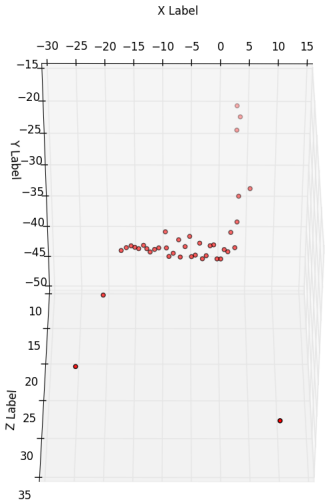

Figure 8: One dimensional scan, top view

Figure 8 demonstrates a one-dimensional scan of a block of wood standing up on the table. As seen in the scatter plot, there is a clear line where the IR distance scanner picked up the wooden block, and the rest is noise or the background.

7.2 Three Dimensions

The following images are of different filtering techniques we used to help clean up the data.

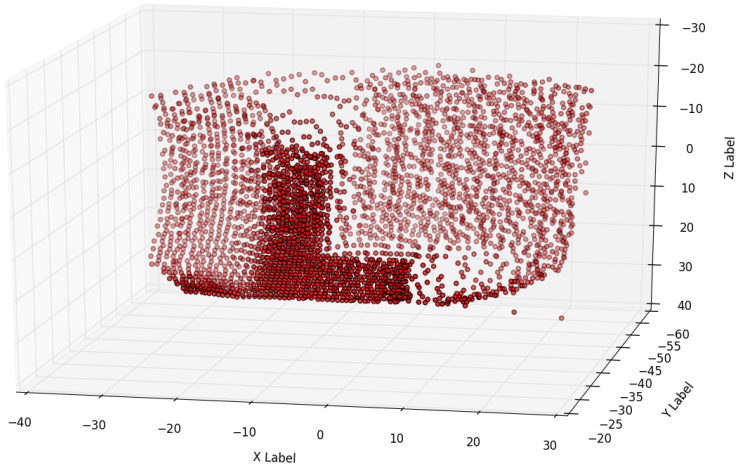

Figure 9: Raw 3D scan data

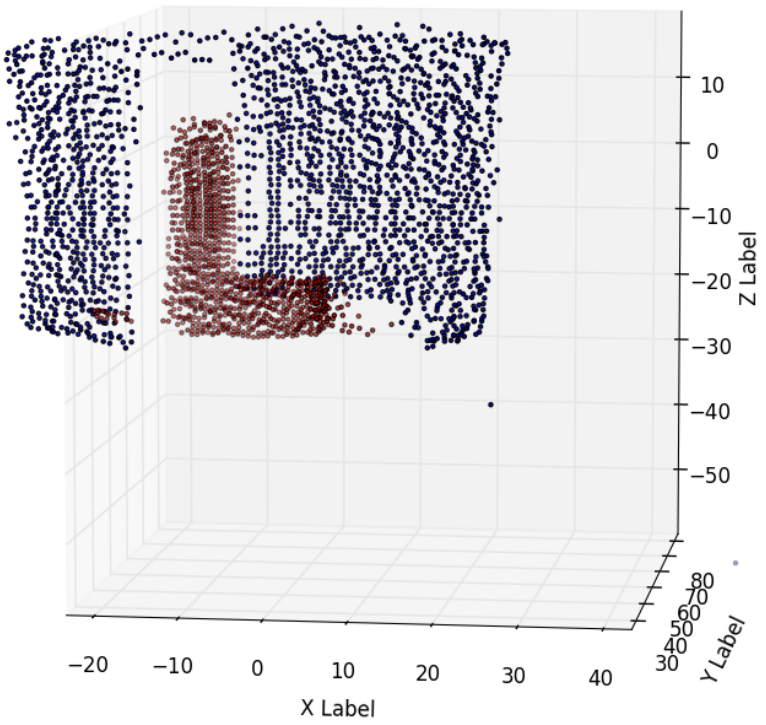

Figure 10: Data cleaned up and colored for distances

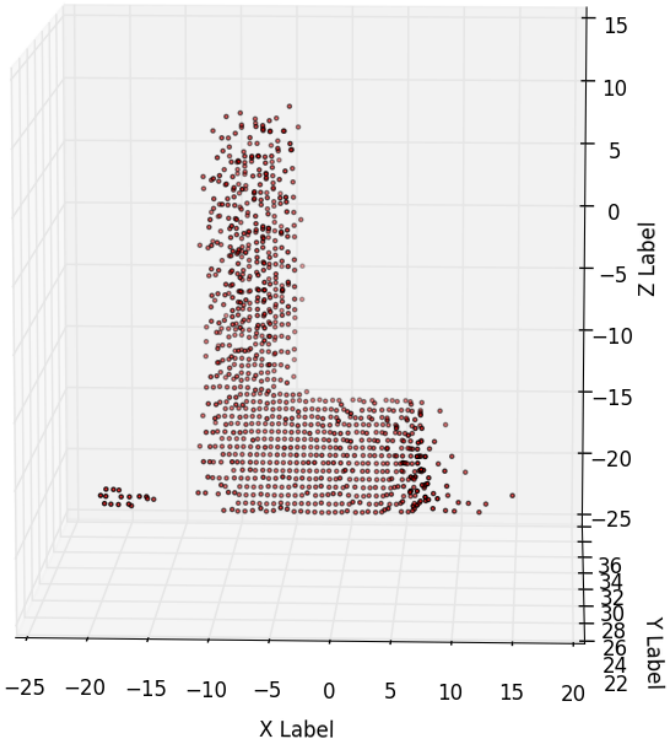

Figure 11: Filtered data to display just the letter and no background

8 Appendix

8.1 Code

8.2 Python Code

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from pylab import *
import numpy as np
import serial
from time import sleep

Time_Plot = False

def Coord_Transform_Single ( data):
    d = data[0]
    theta = data[1]
    phi = data[2]

    # Converts output of Arduino to distance based on
    # equation found from calibration
    d = np.exp( ( d - 1416.4651589499) / ( -301.2902022911))

    # Cuts out the points that are too far away/noisy
    if d > 60: return 0

    # Standard spherical to cartesian
    z = ( d * np.cos( theta ) )
    x = ( d * np.sin( theta ) * np.cos ( phi ) )
    y = -( d * np.sin( theta ) * np.sin ( phi + (pi/180 * 8)) )

    return x, y, z

def Plot3D(x, y, z):
    fig = plt.figure()
    ax = plt.axes(projection='3d')

    ax.scatter(x, y, z, c='r', marker='o')

    ax.set_xlabel('X Label')
    ax.set_ylabel('Y Label')
    ax.set_zlabel('Z Label')

    plt.show()

def Parse_Data ( data ):

    # Takes in string of data
    data = data.split(',')

    # Distance
    d = int(data[0])
```

```
    # Theta
    t = np.radians(int(data[1]))

    # Phi
    p = np.radians(int(data[2].strip('\n')))

    return d, t, p

def main():

    # Array that will store all of the data
    data = []
    DATA_EXPORT = []

    # Array that will store temporary data for
    # plotting in time
    x = []
    y = []
    z = []

    if Time_Plot:

        # Initialize plotting functions
        fig = plt.figure()
        ax = plt.axes(projection='3d')
        ax.disable_mouse_rotation()
        plt.ylim(-150,150)
        plt.xlim(-150,150)
        ax.set_zlim(-150,150)

    # Setting up the initialization for the Arduino
    raw_input("Is the 3D Scanner plugged in?")
    s = serial.Serial('/dev/ttyACM0',9600,timeout=50)
    print "Getting connection..."
    sleep(2)
    s.readline()
    raw_input('Start?')

    # Killing off any 'a's left
    s.readline()

    print "Beginning data collection"
    # After this point the Arduino will start scanning and sending back data
    while True:

        # Set up pinging between the two devices in order to not lose data
        s.write('a')
        tmp = s.readline()

        # TEMPORARY CODE
        #print tmp

        # Sets up ending the loop when the scan is finished
        if "STOP" in tmp: break
```

```
        # Start storing the values taken in
        data.append(tmp)

        # Live plotting
        if Time_Plot:
            tmpx, tmpy, tmpz = Coord_Transform_Single( Parse_Data( tmp))

            x.append(tmpx)
            y.append(tmpy)
            z.append(tmpz)

            # It plots every 20 points collected
            if len(x) % 20 == 0:
                ax.scatter(y,x,z,c='r',marker='.',depthshade=False)
                x = []
                y = []
                z = []
                plt.draw()
                pause(0.01)

    print "Ended data collection"
    if Time_Plot: plt.close()

    x = []
    y = []
    z = []
    for i in data:

        # In case there was an error thrown by the Arduino
        # which is rare, but has happened
        try:
            tmpx, tmpy, tmpz = Coord_Transform_Single( Parse_Data( i ) )
            x.append(tmpx)
            y.append(tmpy)
            z.append(tmpz)
        except: continue

    Plot3D(z,y,x)

main()
```

8.3 Arduino Code

```
#include <Servo.h>

// for incoming serial data
int incomingByte = 0;

int sensorPin = A0;

// Setting up variables for taking average reading
int sensorValue = 0;
int sensorValue1 = 0;
int sensorValue2 = 0;
int sensorValue3 = 0;
int sensorValue4 = 0;
int sensorValue5 = 0;

// Creating the bottom and top servos
Servo bser;
Servo tser;

// Giving initial and end points for servos
const int bstart = 65;
const int bend = 125;
const int tstart = 52;
const int tend = 110;

// Setting up variables to hold servo positions
int bpos = bstart;
int tpos = tstart;

// Variable to control how many data points to take
int steps = 1;

// Sets up whether to start 3D Scanning
int enabled = 0;

// Sets whether to stop 3D scanning
int ENDER = 0;

// How long to wait for the servo to reach position
int delayer = 60;

// Typical Setup function
void setup() {

    // Open serial port at 9600 baud rate
    Serial.begin(9600);

    // Set up LED debugging
    pinMode(7, OUTPUT);
    pinMode(8,OUTPUT);
```

```
    // Set kill button with internal pullup
    pinMode(2, INPUT_PULLUP);

    // Set up both Servos
    bser.attach(9);
    tser.attach(6);
}

int Take_Data(){

    // Which way the gimbal is panning
    static int direction = 0;

    // Setting up movements
    if (direction == 0){
        bpos += steps;
        if (bpos >= bend){
            direction = 1;
            tpos += steps;
        }
    }
    else {
        bpos -= steps;
        if (bpos <= bstart){
            direction = 0;
            tpos += steps;
        }
    }

    // Return ending condition
    if (tpos > tend) return 1;

    // Write positions to servos
    bser.write(bpos);
    tser.write(tpos);

    delay(delayer);

    // Read sensor for distance
    sensorValue1 = analogRead(sensorPin);
    delay(15);
    sensorValue2 = analogRead(sensorPin);
    delay(15);
    sensorValue3 = analogRead(sensorPin);
    delay(15);
    sensorValue4 = analogRead(sensorPin);
    delay(15);
    sensorValue5 = analogRead(sensorPin);


    sensorValue = (sensorValue + sensorValue2 + sensorValue3 + sensorValue4 +
        sensorValue5)/5;
```

```
    // Continue doing processes
    return 0;
}

// Main function which just loops
void loop() {

    // Stops pinging LEDs
    digitalWrite(7, LOW);
    digitalWrite(8,LOW);

    // Sends data when pinged
    if (Serial.available() > 0) {

        // Debugging LED, set high when pinged
        digitalWrite(7, HIGH);

        // Flush out all of the "A's" that were being sent
        Serial.flush();

        // Set up enabled for better delay handling
        enabled = 1;

        // read the incoming byte:
        incomingByte = Serial.read();

        // Set ping out pin high
        digitalWrite(8,HIGH);

        // Write to serial
        if (ENDER == 1) {

            // End of sweep
            Serial.println("STOP");
            delay(1000);
        }
        else {

            // Print out data
            Serial.print(sensorValue);
            Serial.print(",");
            Serial.print(bpos);
            Serial.print(",");
            Serial.println(tpos);
        }

        // Go to next position, returns 1 if finished
        ENDER = Take_Data();

        // If button is pressed exit program
        if ( digitalRead(2) == 0 ) ENDER = 1;
    }
    else if (enabled == 0){
```

```
        // Just sends out 'a' over the serial
        // until ping-ed
        Serial.println('a');
        delay(1000);
    }

}
```