Principles of Engineering: Lab Two: 3D Scanner

Sunny Shroff and Shane Kelly

September 28, 2015

1 Design Goals

We began this lab with two servos, an infrared sensor, an Arduino, and the goal of pushing ourselves into encountering unique and challenging problems along the way. With this challenge in mind we created two fairly concrete design goals. The our first design goal was to capture a full 360 degree scan of an object rather than the single face that this lab seemed to suggest. Our second goal was to create a genuinely cool looking 3D scanning process that would create a fun and exciting interaction between the user and the machine. According to these two goals we sketched out the design in Figure 1.

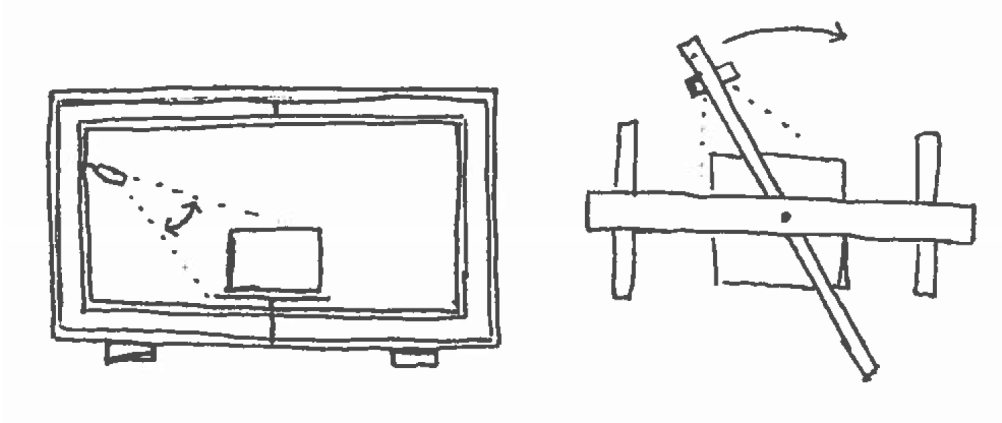

Figure 1: Initial sketches

The object would sit on a stationary platform in the center of the device. The IR sensor would be mounted to the inner frame which spins around this center platform and gives the IR sensor full view of the scanned part. This design would give us the classic and futuristic 3D scanner experience as well as the full 360 degree view of whatever object we scan.

2 Mechanical Structure

We began the construction of our scanner by building our inner and outer frames. When the two frames were built we worked on a method of attaching them that would keep friction low, while keeping the inner frame stable when spinning. To accomplish this, we connected the inner and outer frames in two points along the same vertical axis, around which the IR sensor rotates. The first connection point can be seen in Figure 2. While the servo sits in a pocket in the outer frame, the horn is connected to the top center of the inner frame, and drives its rotation.

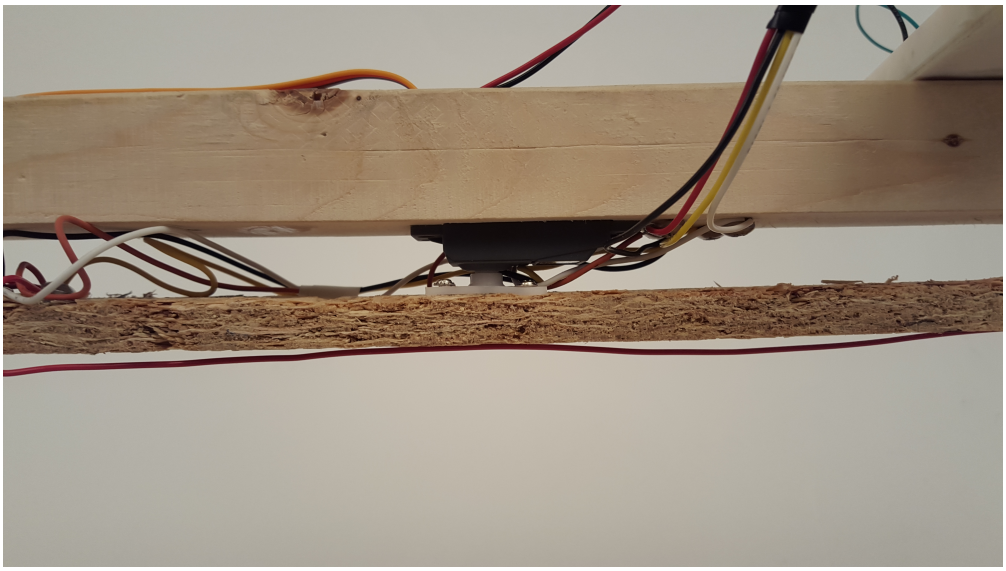

Figure 2: Top Servo Embedded in Outer Frame

The second connection point is located at the bottom center of the frames. As shown in Figure 3, there is a long bolt that goes through the outer and inner frames. This bolt is stationary when the inner frame turns, and thus is what our table is mounted on. To minimize friction, we pressed two bearings into the inner frame, through which the bolt is run. The bearing also sits against the outer frame such that it supports the weight of the outer frame. This ensures that there is no downward force on the servo.

Figure 3: Axle Stackup

Once the inner frame was rotating around the center platform, we designed two 3D printed parts: one to mount the servo to the inner frame and another to hold the infrared sensor perpendicular to the face of the servo horn. Figure 4 shows the assembly of our 3D printed parts mounted with the tilt servo and IR sensor.

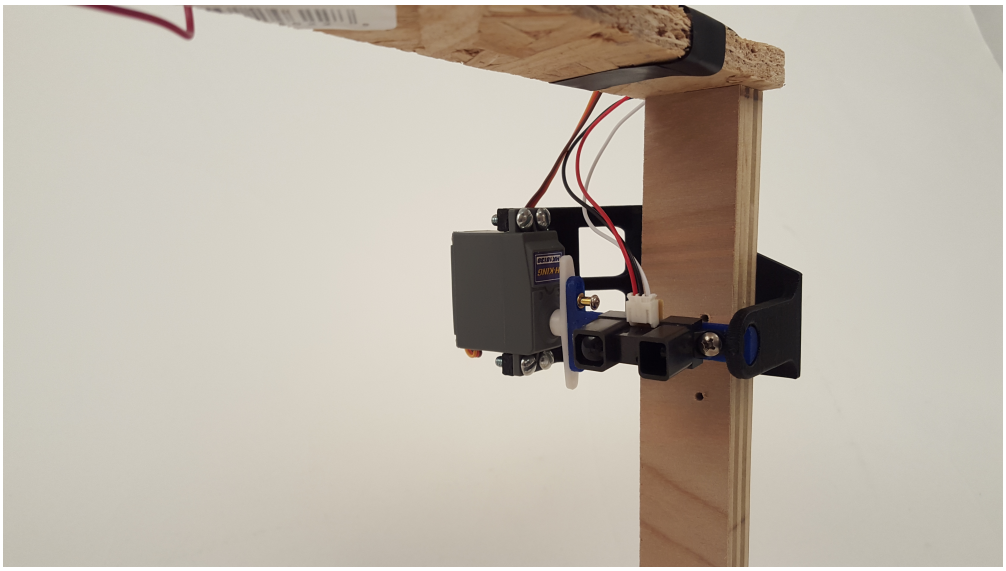

Figure 4: 3D Printed Assembly for Tilt Servo and IR Sensor

3 Full 360 Degree Scanning

3.1 Continuous Rotation Servo

To stay aligned with our design goals for we planned for the inner frame to have limitless rotation around the scanned part. In order to accomplish this we hacked the internals of one of the servos we were given to remove their limited range of motion. This hack required grinding down the mechanical stop and removing the motion-limited potentiometer from the servo. We also bypassed the servo's circuit board completely and instead soldered two new leads directly to the motor. This step allowed us to treat the servo as a normal DC motor with a gear box. This means that we could spin the motor with unlimited rotation, but we could no longer get a position readout from the motor.

3.2 Knowing Theta

In order to make up for the lack of potentiometer readout from our newly hacked motor, we added a physical switch in our system that, when pushed, lets us know that the inner frame was at that certain position, $\theta = 0$. We then spun the inner frame around in a full circle starting and ending with the position at the physical switch. We kept the inner frame moving at a constant velocity between these two points, which means that we could extrapolate every piece of data that we collected along the way to cover the space between the two touches of the switch. Figure 5 shows the screw that made contact with our aluminum pad button every 360 degrees.

3.3 Wiring

With the infrared sensor, tilt servo, and aluminum switch all mounted on the continuously spinning inner frame, we realized that the wires leading to the Arduino would get tangled around the axis of rotation if we spun too much. In order to remedy this problem we used long, flexible, stranded wire around the axis of rotation, which allowed us more degrees of turn. We also decided to make

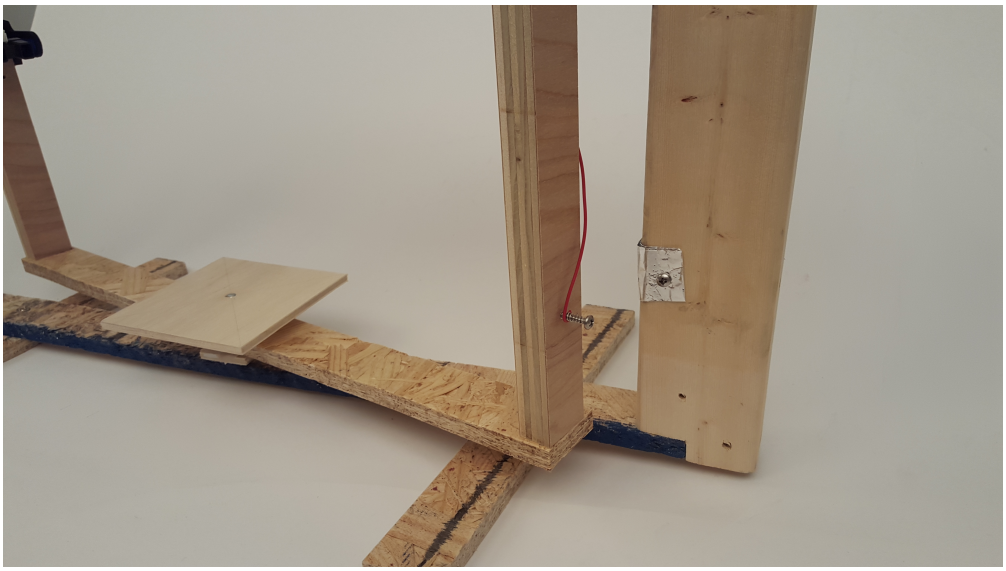

Figure 5: Aluminum Contact Switch

alternating clockwise and counterclockwise rotations around the part. This way, the wires never had to wrap around the axis more than once.

3.4 Controlling the Continuous Servo

When we removed the potentiometer from the servo and soldered leads directly to the motor, we could no longer use the conventional method of controlling it. In addition we had the added requirement of switching the direction of rotation to avoid wire entanglement. In order to be able to control the hacked servo in both directions, we used a half bridge driver. Figure 6 shows a simplified illustration of how a half bridge works, and how it allows current to flow through a load in both directions.

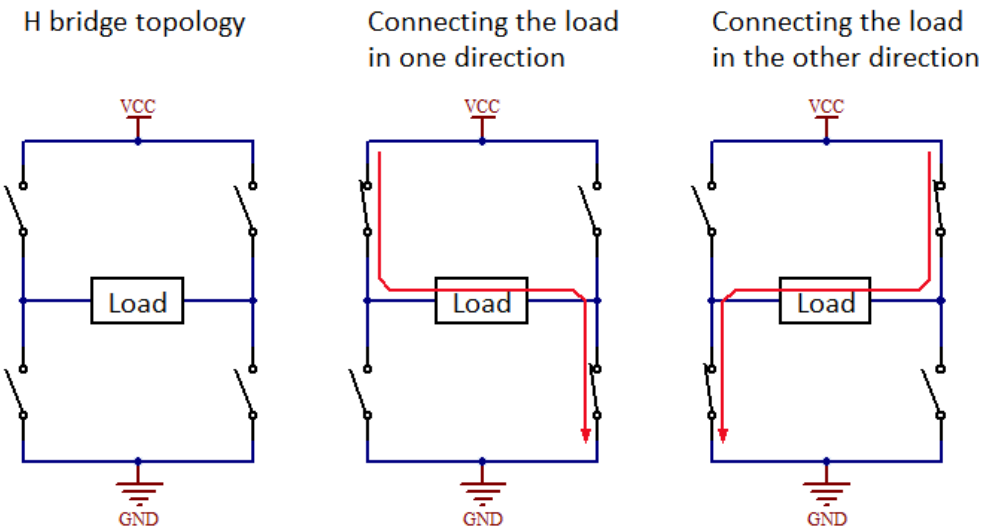

Figure 6: A simple representation of a h-bridge

The half bridge driver we used was contained in a chip. To control the hacked servo using the Arduino, we wrote to the input pins using PWM, and were able to control the hacked servo's speed.

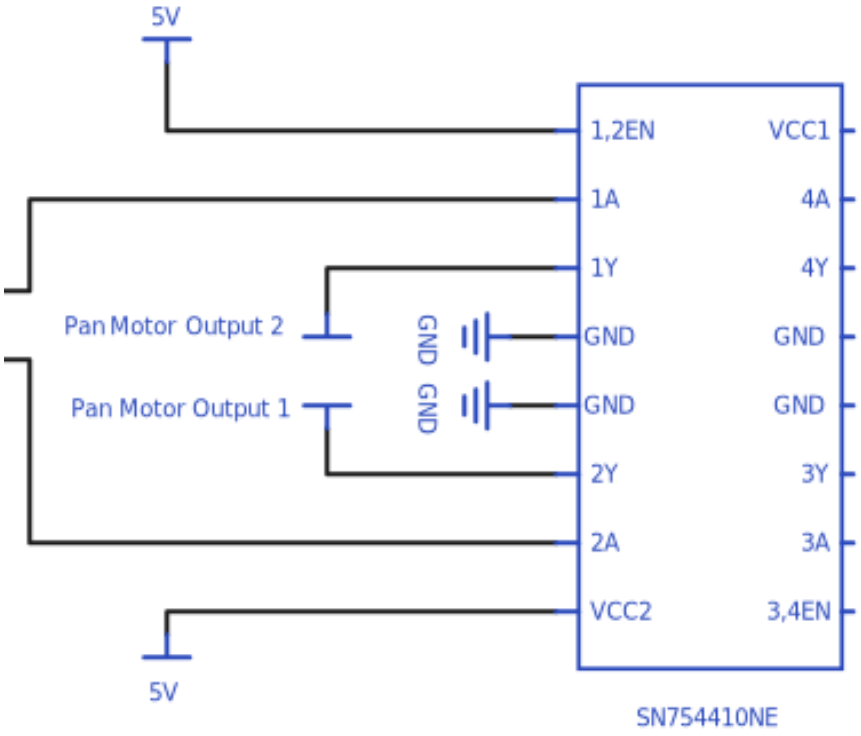

Figure 7: A circuit diagram of our half bridge driver setup. The two lines coming in from the left carry the PWM signals from the Arduino.

4 Software and Logic

4.1 IR Calibration

The Sharp infrared sensors that we were given to use in this lab work best in the 15cm to 100cm range and return anywhere from 0v to 5v based on the distance that they read, which is given to us by the Arduino as an integer from 0 to 1023. We needed to turn this arbitrary integer into a distance readout in centimeters for accurate scan data. We set up our IR sensor along a ruler with a plank of wood at the end. We progressively moved our plank of wood closer and closer, noting the wood's distance and the IR readout at each 1cm increments. We took this data and mapped it to a second degree polynomial line of best fit, which we then used in our code to transform future IR readouts into cm. The data we collected and the line of best fit for our data can be seen in Figure 8.

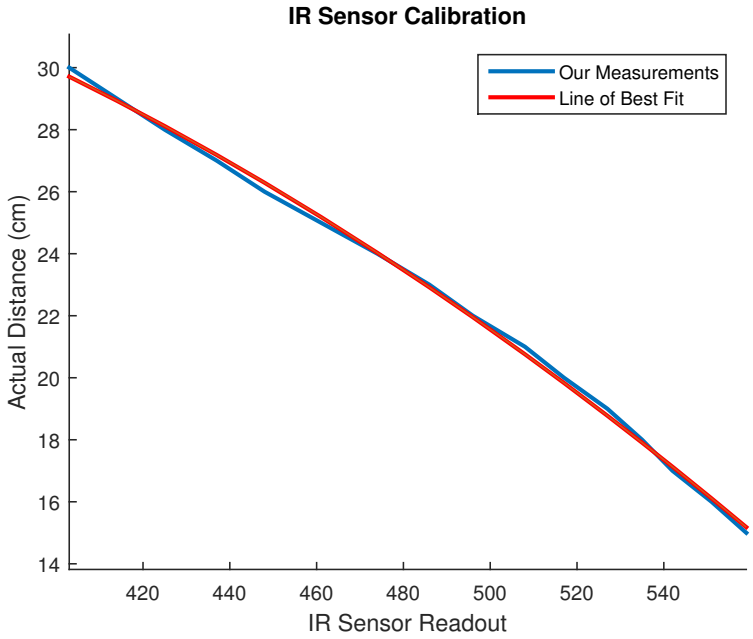

Figure 8: IR Sensor Calibration Data and Calculated Line of Best Fit

4.2 Serial Communication

We used serial communication between Arduino and our computers in order to move the collected scan data to be analyzed in Matlab. The serial port is opened at a given cycle rate and then the Arduino can print to the serial line in order to pass data to the computer.

```
// Define serial refresh rate
#define PC_BAUD 250000

// Define serial port
HardwareSerial & pcSer = Serial;

// Begin serial at refresh rate
  pcSer.begin(PC_BAUD);

void sendToSerial(float dist, int p, boolean d) {
  // Send distance, phi, and direction through serial delimited by commas
  pcSer.print(dist, 6);
  pcSer.print(", ");
  pcSer.print(p);
  pcSer.print(", ");
  pcSer.println(d);
}
```

We then took that data into Matlab and stored it into vectors for later use. To do this we followed a similar process with opening the serial port at a certain rate of refresh. Once the serial stream was opened we used regular expressions to parse our variables out of the serial-sent strings and appended them to the appropriate vectors.

```
% Define serial input at usb port COM4
    s = serial('COM4');

% Define the serial refresh rate and timeout limit
```

```
set(s, 'BaudRate', 250000, 'Timeout', .005);

% Open the serial port for reading
fopen(s);

% Retrieve a line of data from serial
serialData = fscanf(s);

% Scan the string received through serial for our three
% important variables (distance, phi, dir)
data = sscanf(serialData, '%f, %d, %d');

% Store the data in the appropriate vector
distance(i) = data(1);
phi(i) = data(2);
dir(i) = data(3);
```

4.3 Running Average

When plotted directly, the raw readings from the IR sensor are very jagged. Even while taking a reading of a single, stationary point, it is necessary to take multiple readings and use those to find some sort of more accurate reading overall. However, due to the continuous nature of our scanner, it would not have been possible to take multiple readings of the same point on an object. Instead, we implemented a running average, which reassigns a point's value based on its average with some number of previous points. While this works well for smoothing out IR sensor readings, it is not advantageous when scanning objects that have sharp edges, since those edges will become rounded.

4.4 Sweeping Theta

Through the use of the aluminum switch, our scanner knows when it has completed a full rotation. It also knows what direction it is going in on any given rotation, and communicates this information to MATLAB over serial. With this information, it is possible to assign specific θ values to each distance value. We did this by finding where the scanner switched directions (effectively where it has finished a rotation) and finding the number of readings that have occurred since the last switch. We then create a θ vector of the same length and linearly spacing θ from 0 to 2π or 2π to 0 over that length, depending on the direction of the rotation.

4.5 Coordinate System

With the geometry of our scanner, we found that it was easiest to use cylindrical coordinates to describe a point on our object. Figure 9 outlines the geometry of our scanner.

The θ component of the cylindrical coordinate system is given by the sweep of theta described above, so from the distance reading the r and z coordinates can be calculated. With the constants $xoffset$ and $yoffset$

$$r = xoffset - dcos\phi \tag{1}$$

$$z = yoffset - dsin\phi \tag{2}$$

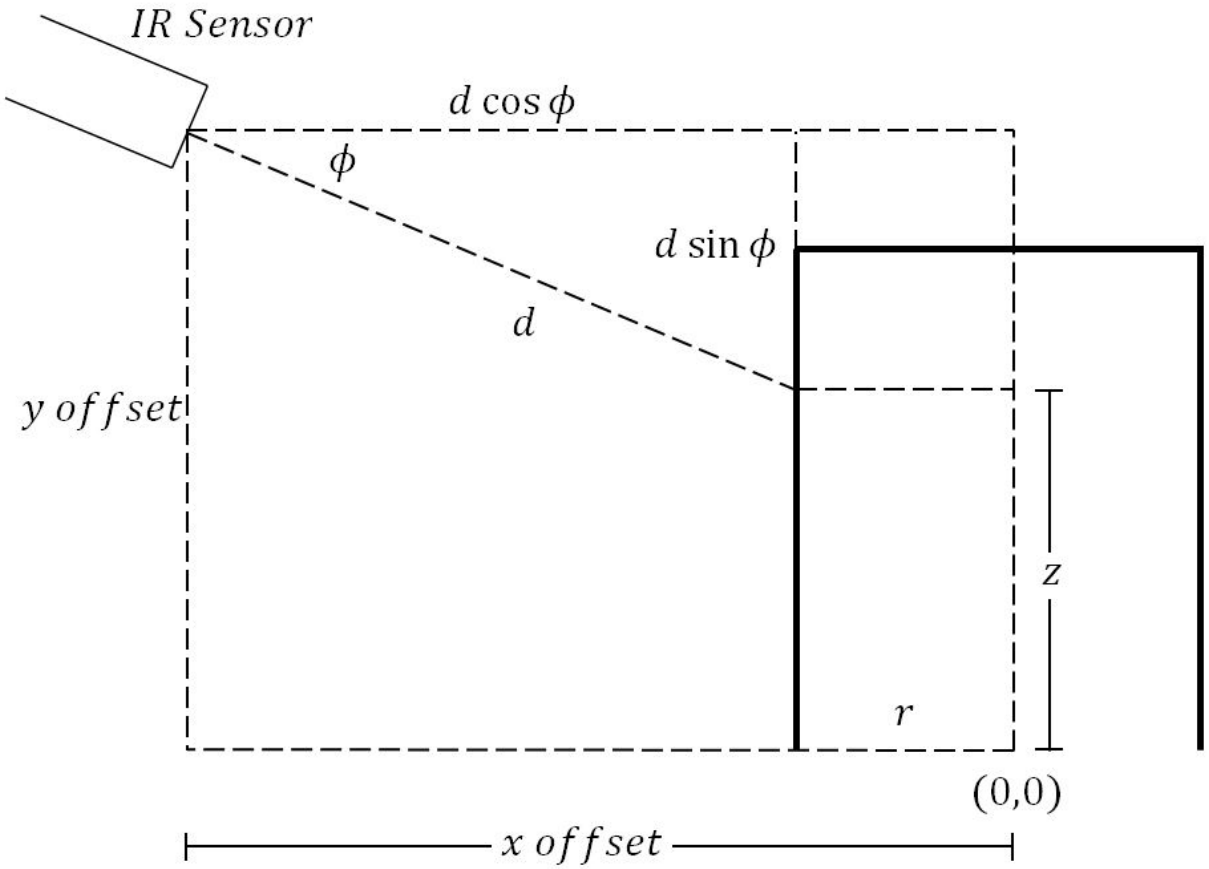

Figure 9: Our cylindrical coordinate system

5 Testing

When we first went to test our scanner for functionality, we reached for the nearest recognizable thing which happened to be a mug. This mug was one of the first things that we successfully scanned and it turned out quite recognizable.

Figure 10: Mug

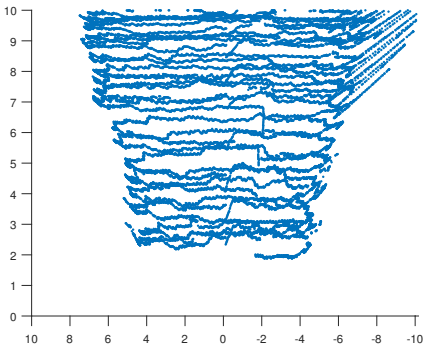

Figure 11: Mug Scan Front View

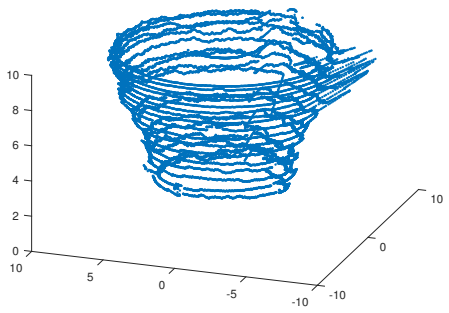

Figure 12: Mug Scan Isometric View

6 Limitations

Due to the nature of our scanner, we are forced to collect data continuously as we sweep the IR sensor across the object. This speeds up our scan time, but causes our data smoothing algorithms to round features and make edges seem less sharp. In addition, the size of the object you can scan is very limited, as the IR sensor has a small range of accuracy compared to the size of our structure.

7 Appendix A: Final 3D Scanner Picture and Video

The final status of our 3D scanner can we seen below in Figure 13 as well as a video of it in action at https://www.youtube.com/watch?v=yRaSRJSL8skfeature=youtu.be

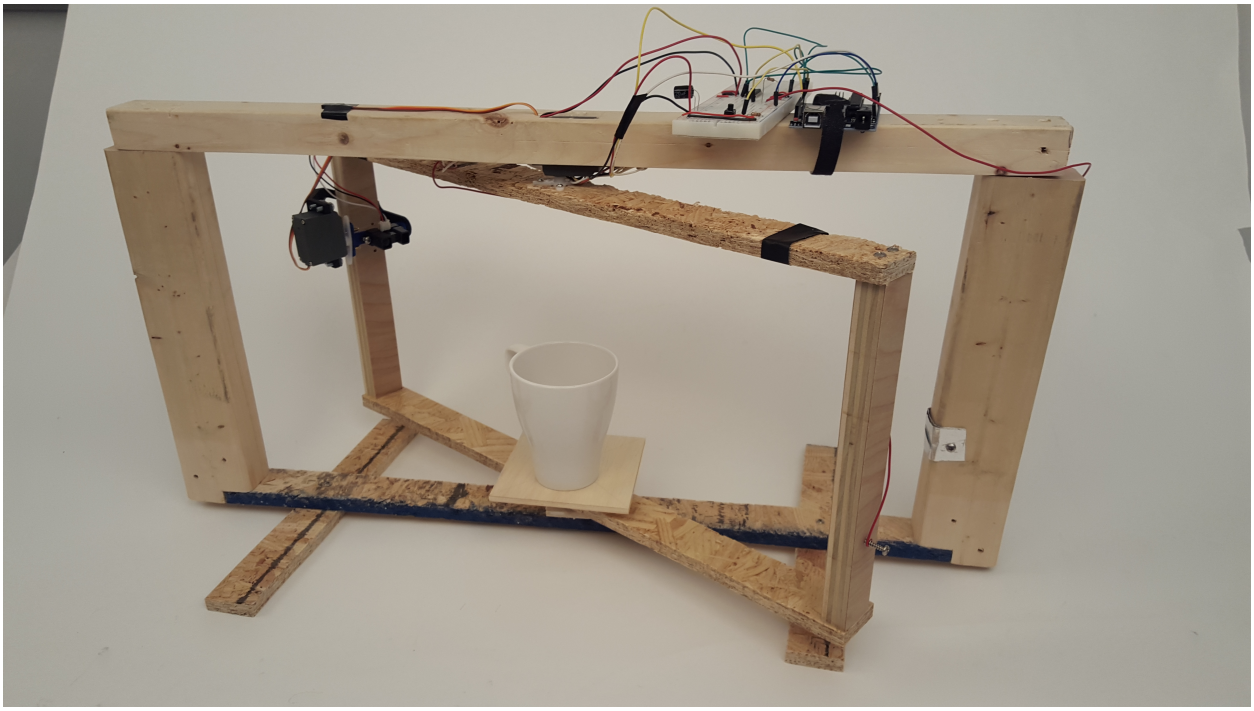

Figure 13: Final Image of Our 3D Scanner

8 Appendix B: Circuit Diagram

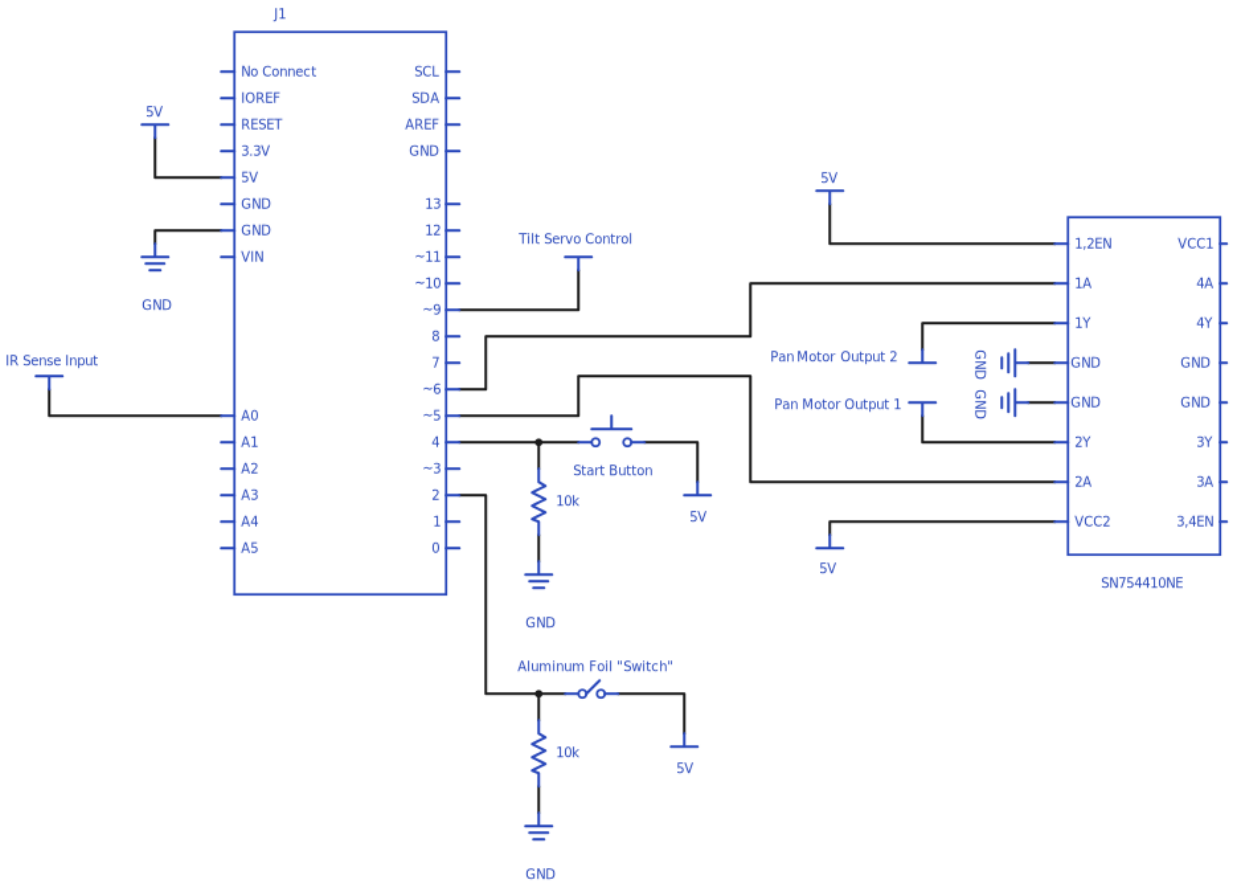

Figure 14: Full circuit diagram

9 Appendix C: Source Code

9.1 serialSend.ino

Controls the all physical objects on our scanner and sends all collected data across the serial port to a computer where the data is analyzed in Matlab.

```
// Include packages
#include <Arduino.h>
#include <stdint.h>
#include <Servo.h>

// Define constants
#define PC_BAUD 250000
float x2 = -.00015422;
float x1 = .0552;
float b = 32.4954;

// Define serial port
HardwareSerial & pcSer = Serial;
```

```
14
15 // Define servo
16 Servo tiltServo;
17
18 // Define pin numbers
19 int button = 2;
20 int startButton = 4;
21 int panServoOne = 5;
22 int panServoTwo = 6;
23 int tiltServoPin = 9;
24
25 // State variables
26 unsigned long long int currentTime;
27 unsigned long long int prevTime = 0;
28 boolean start = 0;
29 boolean buttonState = 0;
30 boolean prevButtonState = 0;
31 boolean dir = 0;
32 float lastButtonTime = 0.0;
33 float distance;
34 int buttonCount = 0;
35 int phi = 150;
36
37 void setup() {
38   // Begin serial at refresh rate
39   pcSer.begin(PC_BAUD);
40
41   // Define pin modes
42   pinMode(startButton, INPUT);
43   pinMode(button, INPUT);
44   pinMode(panServoOne, OUTPUT);
45   pinMode(panServoTwo, OUTPUT);
46
47   // Set servo pin number and starting position
48   tiltServo.attach(tiltServoPin);
49 }
50
51 void loop() {
52   // Wait for the start button to be pressed before running code
53   if (!start) {
54     start = digitalRead(startButton);
55   }
56   // Collect data until phi hits a certain value
57   else if (phi >= 110) {
58     // Get current time in microseconds
59     currentTime = millis() + 1000;
60
61     // Check the state of the button
62     buttonState = digitalRead(button);
63
64     // Set the pan servo to spin in the correct direction
65     panServo(dir, !dir);
66
67     // Set the tilt servo to the correct position
68     tiltServo.write(phi);
69
70     // Collect data or switch directions based on button input
71     switch(buttonCount) {
72       case 1:
```

```
        collectData();
        break;
      case 2:
        panServo(0, 0);
        dir = !dir;
        phi -= 2;
        buttonCount = 0;
        delay(750);
        break;
    }

    // Check for the aluminum foil button to make contact
    if (buttonState == 0 && prevButtonState == 1 && currentTime-lastButtonTime>1000)
    {
        buttonCount ++;
        lastButtonTime = currentTime;
    }

    // Define the previous state of the button
    prevButtonState = buttonState;
  }
  else {
    pcSer.print("This means we are done with serial.");
    delay(10000);
  }
}

void panServo(boolean dir1, boolean dir2) {
  // Takes in 0 or 1 for each parameter and sets servo power
  analogWrite(panServoOne, dir1*255);
  analogWrite(panServoTwo, dir2*255);
}

void collectData() {
  // Read the IR sensor and convert to cm
  distance = analogRead(A0);
  distance = distance*(50/float(1023.0));
  distance = -1.94512951295*distance + 68.33512195;

  // Alternate distance equation based on material of scanned object
  //  distance = sq(distance)*x2 + distance*x1 + b;

  // Push data through serial
  sendToSerial(distance, phi, dir);
}

void sendToSerial(float dist, int p, boolean d) {
  // Send distance, phi, and direction through serial delimited by commas
  pcSer.print(dist, 6);
  pcSer.print(", ");
  pcSer.print(p);
  pcSer.print(", ");
  pcSer.println(d);
}
```

9.2 serialRead.m

Reads the data sent through serial from the Arduino and creates a rolling average on our distance variable.

```
function [distance, phi, dir] = serialRead()
    % Define serial input at usb port COM4
    s = serial('COM4');

    % Define the serial refresh rate and timeout limit
    set(s, 'BaudRate', 250000, 'Timeout', .005);

    % Open the serial port for reading
    fopen(s);

    % Make a cleanup function to clear and close the serial port
    function cleanup(s)
        fclose(s); % close serial port
        delete(s); % delete port information
        clear s % remove port data from workspace
        disp(['Cleaned Up. Last data: ' num2str(serialData)]);
    end

    % Call the cleanup function when code is finished executing
    finishUp = onCleanup(@() cleanup(s));

    % Define looping variables
    receivingData = true;
    i = 0;

    while(receivingData)
        % If data is available through serial
        if(get(s, 'BytesAvailable') >= 1)
            % Add one to the index
            i = i + 1;

            % Retrieve a line of data from serial
            serialData = fscanf(s);

            % Scan the string received through serial for our three
            % important variables (distance, phi, dir)
            data = sscanf(serialData, '%f, %d, %d');

            % If the data receive through serial is empty, then stop serial
            % communication
            if (isempty(data))
                receivingData = false;
                disp('Serial collection terminated.');
            else
                % Store the data in the appropriate vector
                distance(i) = data(1);
                phi(i) = data(2);
                dir(i) = data(3);
            end
        % If no data is available, state the index
        else
            disp(['No current incoming data at index ' num2str(i)]);
        end
    end
```

```
55
56      % Create a rolling average for the distance variable
57      lor = 20;
58      for i = 1:length(distance)-lor
59          newDistance(i) = sum(distance(i:i+lor))/(lor+1);
60      end
61
62      % Plot the collected data
63      plotIRData(newDistance, phi, dir);
64  end
```

9.3 plotIRData.m

Analyzes the smoothed data and inputs it into our coordinate system in order to render the 3D object in a plot.

```
1   function res = plotIRData(dist, phi, dir)
2       close all;
3
4       %Assign variables to workspace variables
5       assignin('base', 'dist', dist);
6       assignin('base', 'dir', dir);
7       assignin('base', 'phi', phi);
8
9       %Initialize variables
10      xOffset = 21;
11      yOffset = 15.25;
12      distOffset = 0;
13      thetaOffset = 0;
14      prevDir = dir(1);
15      prevBreak = 1;
16      graphScale = 10;
17      lenDist = length(dist);
18
19      %Modify phi
20      horizontal = 104;
21      phi = phi - horizontal;
22      phi = degtorad(phi);
23
24      %Modify array lengths to match lenDist
25      dir = dir(length(dir) + 1 - lenDist : end);
26      phi = phi(length(phi) + 1 - lenDist : end);
27
28      theta = zeros(1, lenDist);
29
30      %Create theta according to direction of spins
31      for i = 1:lenDist
32          currDir = dir(i);
33
34          if ~isequal(prevDir, currDir)
35              if currDir
36                  theta(prevBreak: i - 1) = linspace(0+thetaOffset, 2*pi+thetaOffset,
    i - prevBreak);
37              else
38                  theta(prevBreak: i - 1) = linspace(2*pi, 0, i - prevBreak);
39              end
40
41              prevBreak = i;
```

```
42
43              prevDir = ~prevDir;
44          end
45      end
46
47      %Create theta for last spin
48      if currDir
49          theta(prevBreak + 1: end) = linspace(2*pi, 0, length(dir) - prevBreak);
50      else
51          theta(prevBreak + 1: end) = linspace(0+thetaOffset, 2*pi+thetaOffset, length
    (dir) - prevBreak);
52      end
53
54      %r is radial distance from center of table
55      dist = dist + distOffset;
56      r = xOffset - dist.*cos(phi);
57
58      %z is vertical distance from center of table
59      z = yOffset - dist.*sin(phi);
60
61      %Plot distance over time
62      figure();
63      plot(dist);
64
65      %Plot the 3D points of the object
66      figure();
67      [x, y, z] = pol2cart(theta, r, z);
68      plot3(x, y, z, '.');
69      xlim([-graphScale graphScale])
70      ylim([-graphScale graphScale])
71      zlim([0 2*graphScale])
72
73      %Convert degrees to radian
74      function rad = degtorad(deg)
75          rad = deg .* pi/180;
76      end
77  end
```

10 Appendix D: Various Scans

Figure 15: Elephant Mug

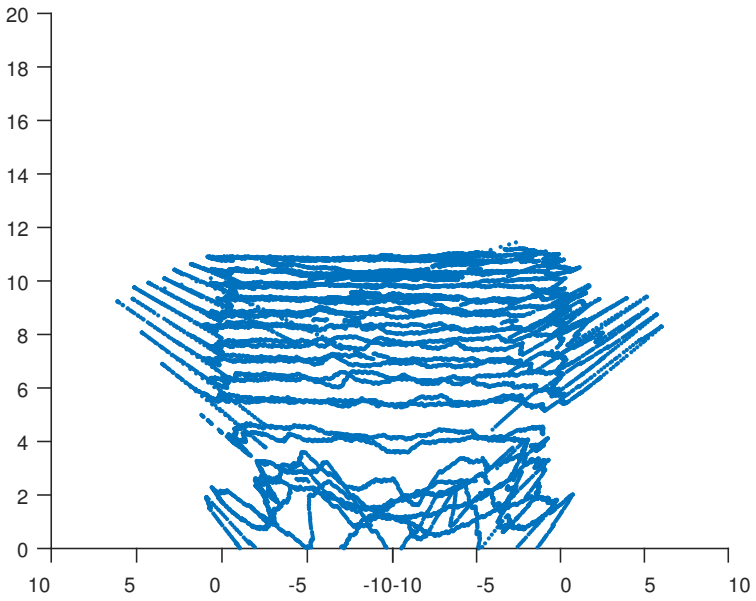

Figure 16: Elephant Mug Scan

Figure 17: Green Cup

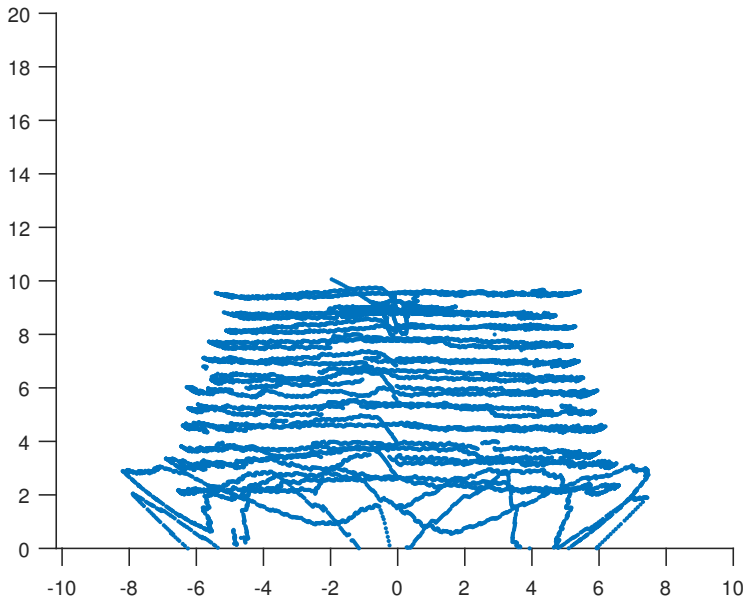

Figure 18: Upside Down Green Cup Scan