POE Lab 2 - 3D Scanner

Voltage to Distance

Scrape distance calibration data off of our Github repository and remove header/blank rows

In[1]:=
```
distanceCSV =
  Import["https://raw.githubusercontent.com/HALtheWise/POE-lab2/master/docs/
    zeroPassDistances.csv", "Csv"];
distanceData = distanceCSV[[2 ;; Length[distanceCSV] - 1]];
```

Fit the analog voltage read vs the distance to a 3rd order least squares function

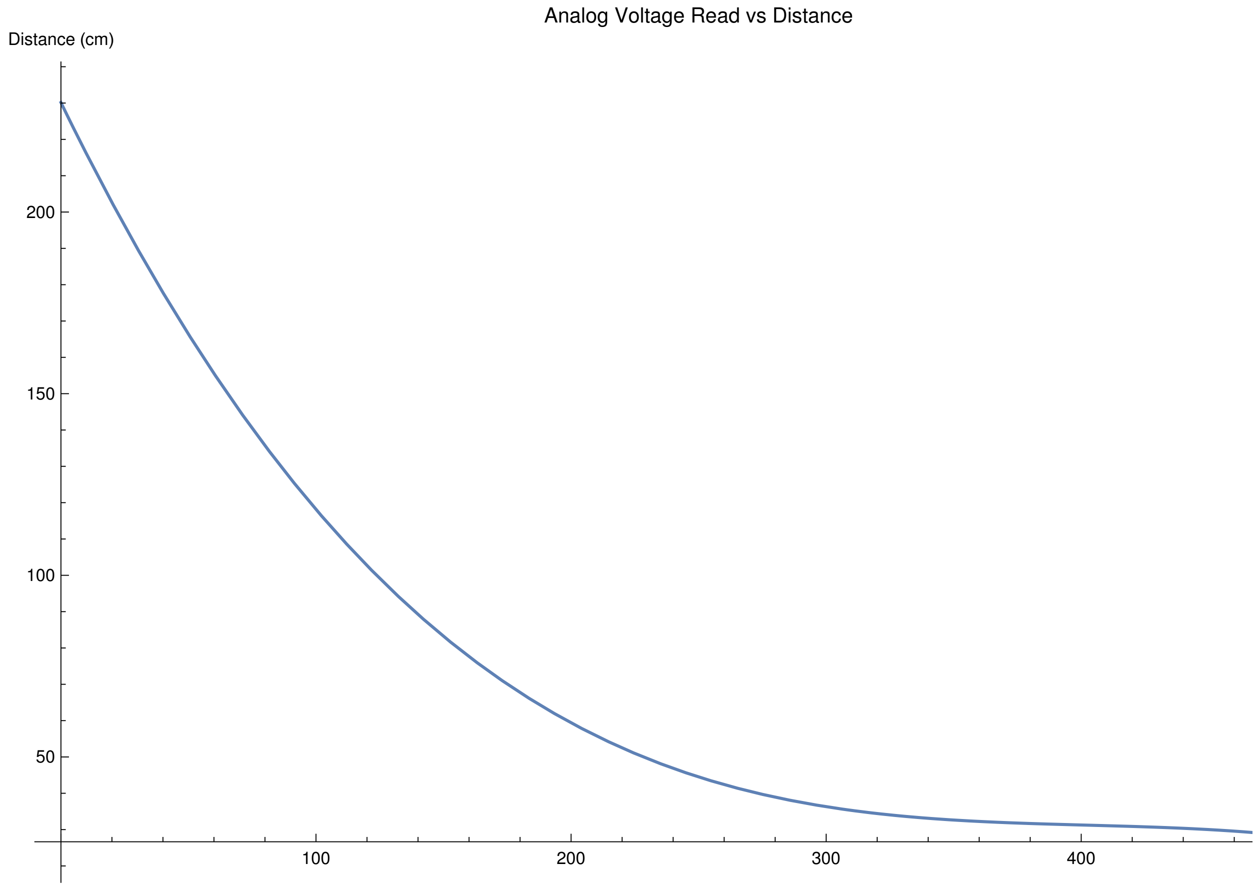
In[3]:=
```
Fit[Map[Reverse, distanceData], {1, x, x^2, x^3}, x]
```

Out[3]= $230.143 - 1.4452\,x + 0.00354446\,x^2 - 2.93605 \times 10^{-6}\,x^3$

Plot the data to see how inputs voltages correspond to output voltages

In[44]:=
```
Plot[230.143 - 1.4452 x + 0.00354446 x^2 - 2.93605 × 10^-6 x^3,
 {x, 0, 500}, PlotLabel → "Analog Voltage Read vs Distance",
 AxesLabel → {"Analog Voltage Read", "Distance (cm)"}]
```

Out[44]=



Make Functions

voltageToDistance takes input analog voltage read and returns distance in cm

```
voltageToDistance[x_] = (230.1430411867322` - 1.4451995839029892` x +
   0.0035444572391777783` x^2 - 2.9360497052173657`*^-6 x^3);
```

anglesToPan takes in the servo angles and converts to pan in degrees

```
anglesToPan[servo1_, servo2_] := Module[{}, (N@servo1 - servo2) / 2]
```

anglesToPan takes in the servo angles and converts to pan in degrees

```
anglesToTilt[servo1_, servo2_] := Module[{}, (N@servo1 + servo2) / 2]
```

panTiltDistance takes in the servo positions and voltage and outputs the distance, tilt, and pan

In[9]:=

```
panTiltDistance[{servo1_, servo2_, voltage_}] := Module[{pan, tilt, distance},
  pan = anglesToPan[servo1, servo2];
  tilt = anglesToTilt[servo1, servo2];
  distance = voltageToDistance[voltage];
  Return[{distance, tilt, pan}]
 ]
```

toCartesian takes in the distance, tilt, and pan and outputs Cartesian coordinates

In[10]:=

```
ClearAll@toCartesian;
toCartesian[{distance_, tilt_, pan_}] :=
 Module[{radPan, radTilt, basePoint, panRotation, tiltRotation},
  radPan = pan * π/180;
  radTilt = tilt * π/180;
  basePoint = distance * {1, 0, 0};
  panRotation = RotationMatrix[radPan, {0, 0, 1}];
  tiltRotation = RotationMatrix[radTilt, {0, 1, 0}];
  tiltRotation.panRotation.basePoint
 ]
```

Open Serial device and start reading data

Open Arduino serial port

In[71]:=

```
serial = DeviceOpen["Serial", {"/dev/ttyUSB1", "BaudRate" → 19 200}]
```

Out[71]=

```
DeviceObject[ Class: Serial   ID: 2
              Status: Connected (/dev/ttyUSB1) ]
```

Clear the rawdata list to start capturing fresh scan data

In[92]:=
```
rawdata = {};
```

Send a byte that contains 1 to the Arduino. This tells the Arduino to change state and start the scan procedure.

In[93]:=
```
DeviceWrite[serial, 1];
```

Read a set number of data from the serial buffer to rawdata.
Magic numbers for various grid sizes as to not over-read serial buffer:
1 degree = 3734
5 degree = 268

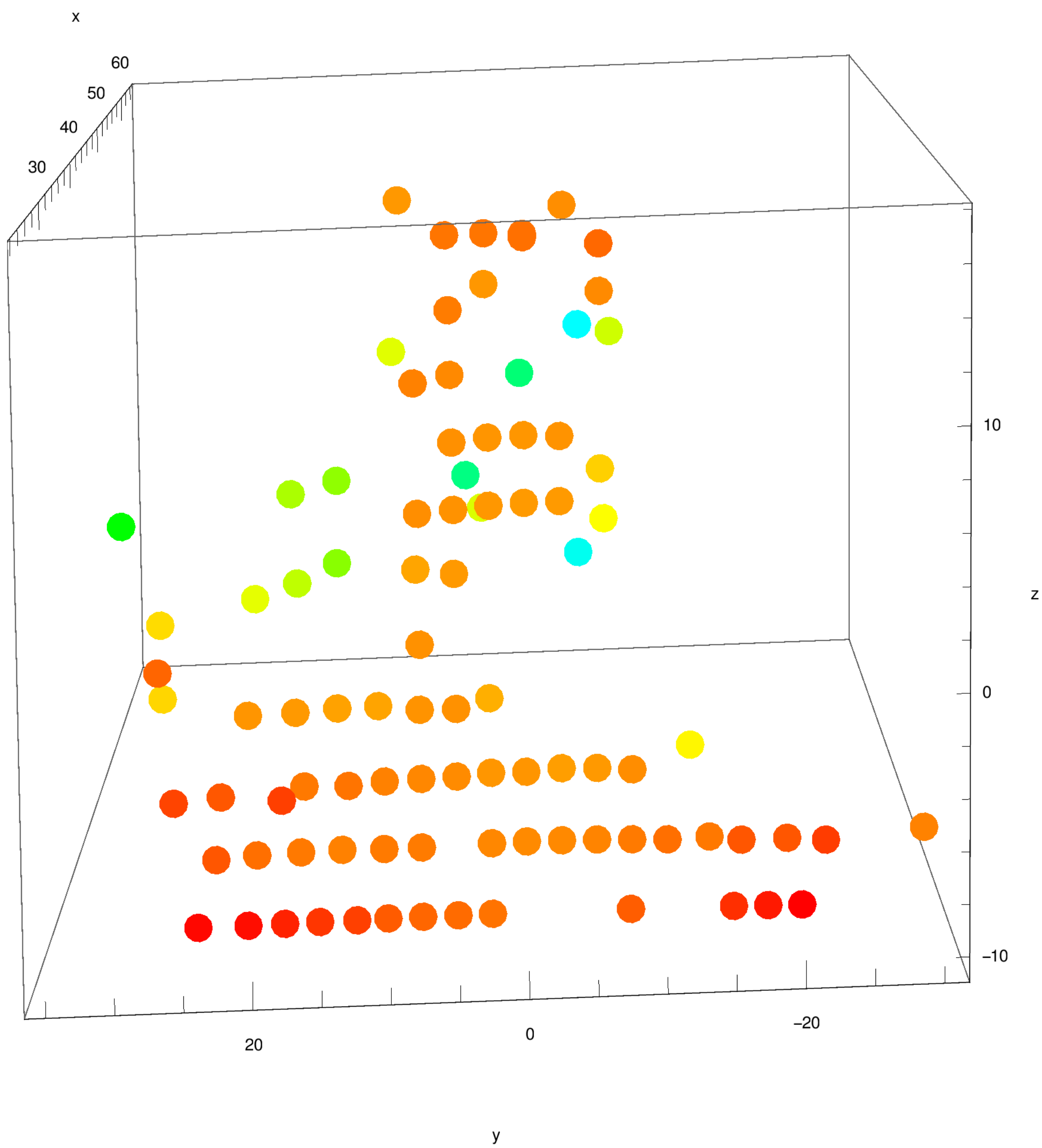
```
Do[AppendTo[rawdata, ToExpression[StringSplit[
    FromCharacterCode[DeviceReadBuffer[serial, "ReadTerminator" → 10]], ","]]], 268]
Dynamic@Dimensions@rawdata
```

Out[97]=
```
{268, 3}
```

Filter the raw data and make a 3d plot.

In[98]:=
```
points = panTiltDistance /@ rawdata;
points = Select[points, 30 < #[[1]] < 80 &];
points = toCartesian /@ points;
ListPointPlot3D[points, ImageSize → Large,
 AxesLabel → {"x", "y", "z"}, PlotStyle → PointSize[.03],
 ColorFunction → Function[{x, y, z}, Hue[x / 2]], AspectRatio → 1]
```

Out[101]=



Optional command to export good data

```
Export["filename", points, "Data"]
```

In[66]:=
```
Length[rawdata]
```

Out[66]= 100

Close serial device

In[23]:=
```
DeviceClose[serial]
```