SliceDepth

February 5, 2020

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

[3]:
```
#Check pandas version to make sure 0.25.1 and the worksheet is working
pd.__version__
```

[3]:
```
'0.25.1'
```

[4]:
```
%matplotlib inline
```

How to use this worksheet

1.Point Misty at something interesting

There needs to be different levels of depth or something interesting in the frame.

2.Get Depth Data

The data is pulled in using either Misty API Explorer or Postman.
GET http://<your Misty robots IP address>/api/cameras/depth

The data should be in json format:

```
{
    "result": {
        "height": 240,
        "image": [ "NaN", ...],
        "width": 320
    },
    "status": "Success"
}
```

3. Save the data as a .json file
4. Update the script below to reference your file
5. Run each of the cells

later: Take a fisheye picture and compare

[5]:
```
# Read in a json file

#depth = pd.read_json('../coderepo/<your file name here>.json')
data = pd.read_json('../coderepo/BB8_HW.json')

#Transpose the DataFrame to get the column labels to contain height width, and␣
 ↪image
dataT = data.T

print(" The height and width should be (240, 320):", dataT['height'].result,␣
 ↪dataT['width'].result);
dataT
```

```
 The height and width should be (240, 320): 240 320
```

[5]:

	height	image	width
result	240	[NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, ...	320
status	Success	Success	Success

[87]:
```
#Next Extract the Image
sf = dataT['image'].result;

#Turn the result into a DataFrame
depth = pd.DataFrame(sf);

#Rename the volumn values
depth.rename(columns={0:'Values'}, inplace=True);

#Check to see the count of NaN values in the Image - Count all NaN's
NaN_cnt = (depth.Values == 'NaN').sum();

coverage = round(((76800-NaN_cnt)/76800)*100,2) #percent of non-NaN "good" values
print("Depth point coverage is: ", coverage, "%");


if coverage < 50:
    print("Data not so good - you may not have valid data in all cells")
    print("You may have problems running the rest of the cells")
```

```
Depth point coverage is:  41.88 %
Data not so good - you may not have valid data in all cells
You may have problems running the rest of the cells
```

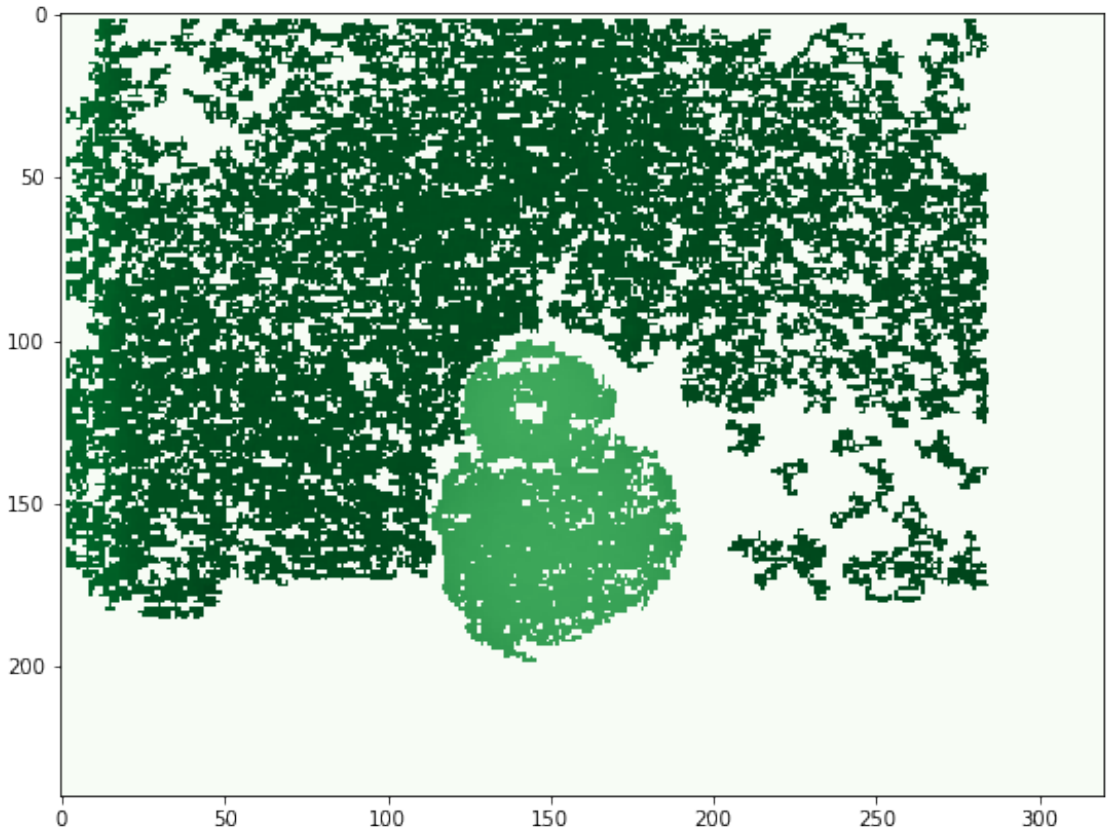
[88]:
```
depth.describe()
```

[88]:
```
        Values
count    76800
unique    4695
top        NaN
freq     44636
```

[99]:
```
#Replace all of the NaN's with 0
# "fo" is just an intermediate holding variable
fo = depth.replace('NaN',0)

#Check to make sure there are no NaNs
numberNaN = (fo.Values == 'NaN').sum()

#Reshape the array
data = np.array(fo).reshape((240,320))
data.shape

#Check to make sure all NaN's were replaced
print(" There are " + str(numberNaN) + " NaN's, and the array is " +str(data.
 ↪shape) + " of type " + str(type(data)) )
```

```
There are 0 NaN's, and the array is (240, 320) of type <class 'numpy.ndarray'>
```

[104]:
```
#Plot the figure with a couple of overlays

plt.rcParams['figure.figsize'] = [12, 7]
#plt.imshow(data, cmap=plt.get_cmap('gray'));
#plt.imshow(data, cmap=plt.get_cmap('gray'));
plt.imshow(data, cmap=plt.get_cmap('Greens'));

#Some other plot methods
#plt.rcParams['figure.figsize'] = [9, 7]
#plt.contour(data)
#plt.gca().invert_yaxis() #need to rotate or flip it so that 0 is at the top

#plt.rcParams['figure.figsize'] = [9, 7]
#plt.pcolormesh(data);
```



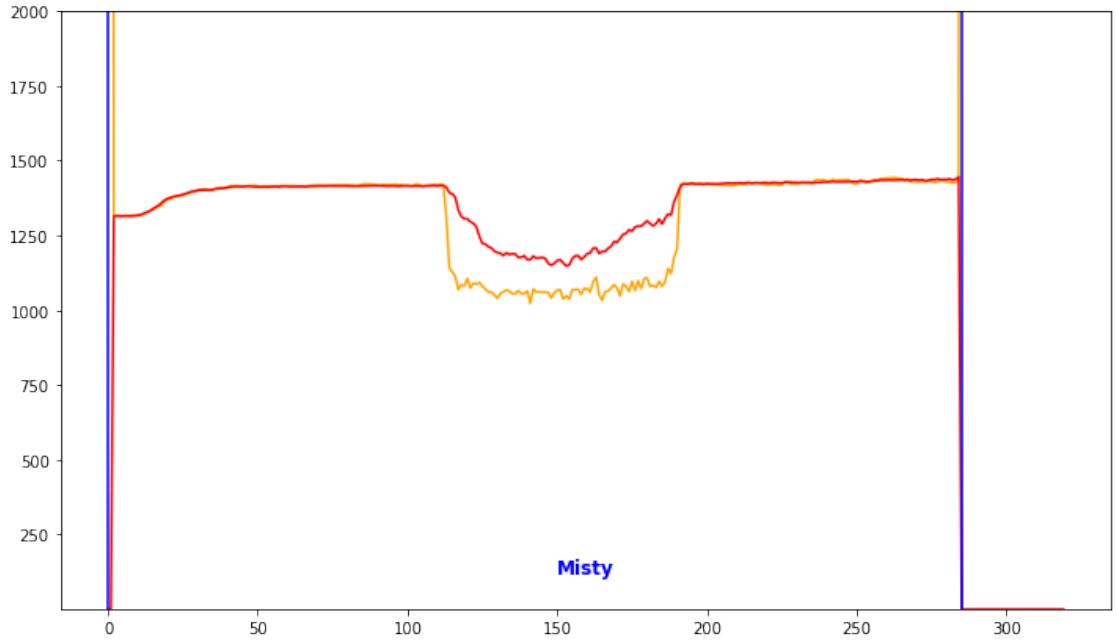
[60]:
```
#Create the column arrays for the sum results to be stored in
#There are 320 columns so create zero arrays of length 320

#Create Column Arrays for the entire picture
colDepthSum = np.empty(320)
colDepthCount = np.empty(320)
columnDepth = np.empty(320)

#Create Slice Arrays to only cover a middle portion of the picture
Slice_sum = np.empty(320)
Slice_cnt = np.empty(320)
sumSliceDistance = np.empty(320) #Array will contain averaged slice distance␣
 ↪values

for i in range(320):
    colDepthSum[i]=0
    colDepthCount[i]=0
    columnDepth[i]=0
    Slice_sum[i] = 0
    Slice_cnt[i] = 0
```

```
    sumSliceDistance[i] = 10000; #Set each element out of way 10000=10meters -␣
 ↪larger than max expected measured distance
```

[106]:
```
#Populate the array of 320 columns with values
# - the total depth coverage (average of 240 elements in the 320 columns)
# - the depth of just a slice of the image, defined by two points that indicate␣
 ↪the element line

depthSum = 0;         #sum total of all depth points
depthCount = 0;       #count of all non-zero points
tempDepth = 0;        #temporary variable for holding value

#Lines are rows between 1 and 240
#topline is less than bottom line -- since first pixel is top left
topline = 150; #change these for lines - 120 is the middle row
botline = 160;

toprow = 320*topline; #convert the row line to overall depth array position
botrow = 320*botline;

for i in range(76800): #76800 is the number of array points returned from the␣
 ↪depth picture
    if fo.Values[i] != 0:
        tempDepth = fo.Values[i]    #don't really need this step
        depthCount = depthCount +1
        depthSum = depthSum + tempDepth

        indes=i%320 #There are 320 colums - find correct column by remainder of␣
 ↪position index

        if i >= toprow and i<= botrow: #Check column for the Far Left FL␣
 ↪variables
            Slice_sum[indes] = Slice_sum[indes] + fo.Values[i]
            Slice_cnt[indes] = Slice_cnt[indes] + 1


        colDepthSum[indes] = colDepthSum[indes] + fo.Values[i]
        colDepthCount[indes] = colDepthCount[indes] + 1


print("Number of non-zero values:", depthCount, 'which is', round(depthCount/
 ↪76800*100,1) ,'% coverage')
print("Average overall depth:", round(depthSum/depthCount,4))
```

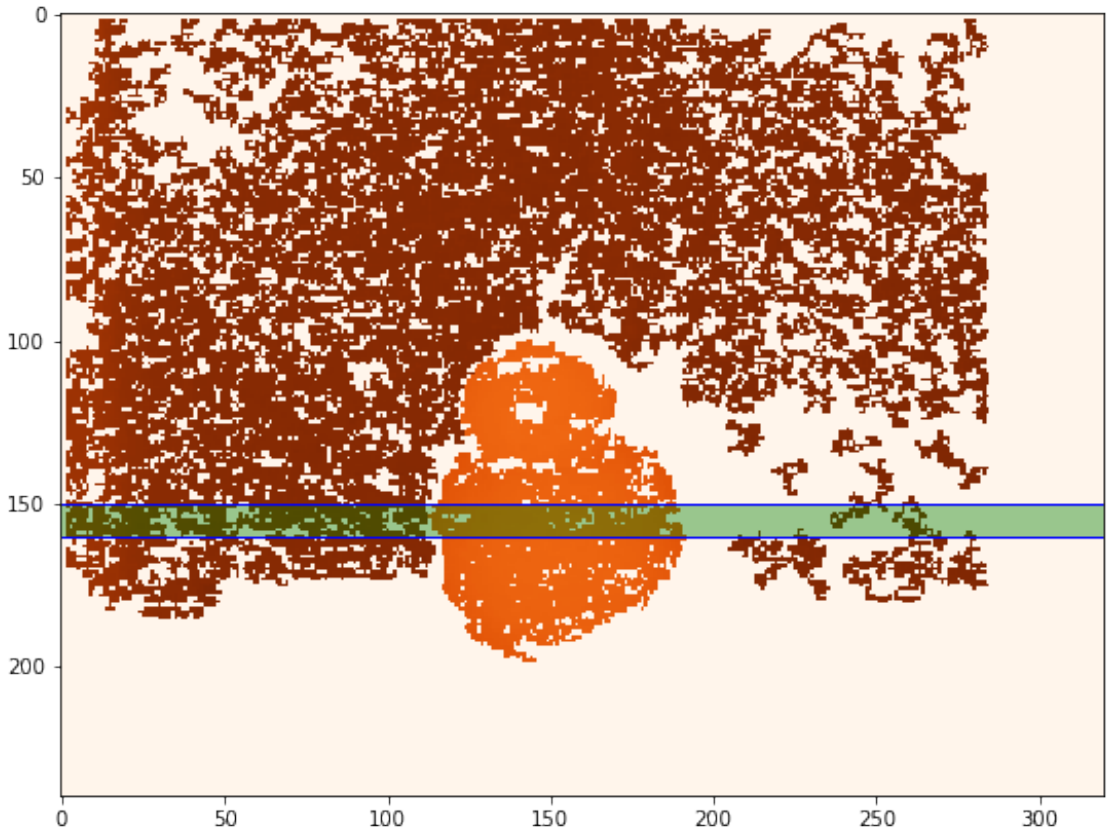
```
Number of non-zero values: 32164 which is 41.9 % coverage
Average overall depth: 1346.175
```

[107]:
```
#Calculate the average for the slice
for i in range(320):
    if Slice_cnt[i] > 0:
        sumSliceDistance[i] = Slice_sum[i]/Slice_cnt[i]

#Calculate the average for each column over the ENTIRE depth picture
for i in range(320):
    if colDepthCount[i] !=0:
        columnDepth[i] = colDepthSum[i]/colDepthCount[i]

#Plot both the average and slice arrays
fig = plt.figure()
ax = plt.subplot(111)

plt.plot(sumSliceDistance, 'orange')
plt.plot(columnDepth, 'r')
ax.set_ylim(1,2000);

#two vertical lines at 0 and 320 - the limits of the sensor
plt.plot([0, 0], [0, 2100], 'b');
#plt.plot([320, 320], [0, 2100], 'b'); #320 is the max - but the sensor does not␣
 ↪appear to go over that far
plt.plot([285, 285], [0, 2100], 'b'); #285 is where the sensor appears to stop

#place "Misty" where is placed and looking from
plt.annotate('Misty', xy=(150, 120), c='b', fontsize = 'large', fontweight =␣
 ↪'bold');
#Change the axis to not show the 10000 out of the way points.
```



[109]:
```
# Re-plot the entire DepthPicture (with points) and inlay lines
# that show the positions of consolidated points, and the depth
# level of each of the 12 points

plt.rcParams['figure.figsize'] = [12, 7]
plt.imshow(data, cmap=plt.get_cmap('Oranges')); #Oranges instead of Greys for␣
 ↪BB8 - plus it makes blue lines show up better

c_width = botline-topline;
c_aveline = (botline+topline)/2;

#plot horizontal lines
plt.plot([0, 319], [botline, botline], 'b', linewidth='1'); # plot([x1,x2],␣
 ↪[y1,y2], color='b')
plt.plot([0, 319], [topline, topline], 'b', linewidth='1');
plt.plot([0, 319], [c_aveline, c_aveline], 'g-', alpha=0.4, linewidth=c_width*1.
 ↪5); #plot average thick line

print("The Green line shows where the depth data is constrained to for␣
 ↪sumSliceDistance. Columns in this range are averaged.")
```

```
The Green line shows where the depth data is constrained to for
sumSliceDistance. All columns in this range are averaged.
```



[110]:
```
#Print out the array to see it
sumSliceDistance
```

[110]:
```
array([10000.        , 10000.        ,  1316.791397  ,  1317.287085  ,
        1314.57711419,  1314.00211688,  1316.04034562,  1315.89886793,
        1316.08882121,  1315.9565413 ,  1317.46511556,  1318.95094559,
        1321.74398   ,  1326.32651937,  1330.92850857,  1336.6634848 ,
        1342.6612496 ,  1347.10288097,  1351.73148742,  1363.19175923,
        1372.11255353,  1371.74790906,  1378.97465839,  1381.24877   ,
        1383.33429129,  1384.81336139,  1388.17377086,  1390.64913179,
        1394.65142966,  1397.07189433,  1402.59988519,  1403.92190125,
        1404.81241955,  1404.89812733,  1400.304422  ,  1400.49849211,
        1406.47787629,  1407.08019029,  1408.35373032,  1409.29806207,
        1411.32812364,  1416.57879808,  1415.01070789,  1416.2388325 ,
        1415.26522818,  1412.89036821,  1414.4925475 ,  1416.38801086,
        1415.68459412,  1417.18694657,  1415.320805  ,  1412.36814647,
        1413.408617  ,  1413.745835  ,  1415.30646815,  1414.0363764 ,
        1414.538265  ,  1410.93590882,  1416.91324038,  1416.15690121,
        1416.02789111,  1415.63495857,  1415.808284  ,  1415.39590576,
        1413.2158125 ,  1412.95617714,  1412.96409371,  1416.28753324,
```

```
1417.50369865,  1417.898178  ,  1417.87851344,  1417.34608433,
1417.621498  ,  1417.82646897,  1418.50686706,  1418.967458  ,
1418.79529032,  1418.74203906,  1418.15693939,  1418.52194425,
1419.16169091,  1417.64994438,  1418.09495548,  1417.10668273,
1418.78334576,  1420.4444385 ,  1422.10493355,  1421.66098741,
1421.39693088,  1420.06071065,  1419.21702941,  1417.90251406,
1420.58720829,  1419.87501306,  1419.38619265,  1419.88180167,
1421.35021054,  1419.136492  ,  1419.7069373 ,  1416.76896519,
1417.66220742,  1415.68757312,  1418.41242815,  1421.45275226,
1419.46522167,  1415.69753968,  1416.59159806,  1417.62996   ,
1418.17873811,  1420.50029324,  1418.86609606,  1420.43242273,
1419.70017833,  1309.50867833,  1141.06404733,  1127.76873455,
1112.212949  ,  1068.91458572,  1085.33989068,  1081.74151641,
1105.84528842,  1074.14928182,  1090.34764572,  1086.66696737,
1093.16788221,  1080.46373175,  1071.1616461 ,  1060.58192011,
1060.59069937,  1051.51839908,  1039.66294186,  1054.99999053,
1061.32203595,  1067.5598788 ,  1063.81016617,  1055.41853805,
1054.94002923,  1065.41161875,  1055.24181444,  1055.2808571 ,
1064.339     ,  1023.34573683,  1070.89934076,  1061.27484986,
1060.61945668,  1060.90045225,  1059.9638165 ,  1059.1322965 ,
1040.58163621,  1057.92553242,  1066.84327541,  1067.6037253 ,
1037.07418084,  1047.87060969,  1037.18441947,  1067.868903  ,
1068.51951303,  1069.85824973,  1052.28802023,  1073.50539182,
1071.17332548,  1060.73601694,  1095.64928594,  1109.75328839,
1051.48064141,  1033.44667243,  1061.9028581 ,  1063.43103435,
1074.15094146,  1085.22142494,  1073.94939924,  1048.17919071,
1087.98173562,  1079.67688597,  1062.70446028,  1095.7548432 ,
1066.49115789,  1098.43986156,  1071.94907749,  1105.73633509,
1107.79177794,  1080.0937336 ,  1082.85510679,  1076.46201585,
1096.80899817,  1079.6122695 ,  1095.72588915,  1139.25932227,
1122.76341061,  1176.4211776 ,  1204.45568765,  1419.357052  ,
1421.88680111,  1421.5444625 ,  1422.87722125,  1422.82755   ,
1420.92968625,  1420.73930375,  1421.210875  ,  1419.39349   ,
1422.499553  ,  1421.43579   ,  1419.86706286,  1420.27384875,
1418.301906  ,  1418.48513   ,  1418.65786   ,  1418.6807875 ,
1415.353246  ,  1418.49354286,  1418.96061333,  1422.8536075 ,
1421.71291667,  1420.1369675 ,  1427.75940909,  1422.93805889,
1424.896835  ,  1424.82037167,  1423.46083833,  1420.8256475 ,
1419.57941714,  1420.63102   ,  1421.70012429,  1420.99192429,
1417.58408   ,  1420.10897   ,  1429.8392025 ,  1427.85786889,
1420.56698   ,  1422.763597  ,  1421.716169  ,  1425.595737  ,
1428.563866  ,  1424.11010875,  1425.18530714,  1422.40913778,
1434.35721611,  1435.913375  ,  1434.2108095 ,  1436.27515222,
1430.66850471,  1432.01896615,  1437.71565077,  1433.17963   ,
1435.09166222,  1435.63888833,  1436.16814167,  1436.66212842,
1432.29301625,  1422.86153   ,  1422.52869   ,  1435.42570833,
1433.57111   ,  1431.10816583,  1428.21881111,  1432.88613364,
```

```
 1436.70552059,  1434.67989867,  1440.67551462,  1440.34888895,
 1442.09819273,  1442.238501  ,  1443.04193033,  1442.62927029,
 1440.59014421,  1441.74873778,  1435.29893556,  1438.88527667,
 1427.921384  ,  1433.40870444,  1428.70690167,  1430.326466  ,
 1430.24276571,  1429.98033143,  1431.49227   ,  1430.64307   ,
 1427.2002875 ,  1429.9609    ,  1434.83638091,  1432.77281636,
 1429.47346889,  1428.28716111,  1426.73598   ,  1427.36444857,
 1430.19092   , 10000.        , 10000.        , 10000.        ,
10000.        , 10000.        , 10000.        , 10000.        ,
10000.        , 10000.        , 10000.        , 10000.        ,
10000.        , 10000.        , 10000.        , 10000.        ,
10000.        , 10000.        , 10000.        , 10000.        ,
10000.        , 10000.        , 10000.        , 10000.        ,
10000.        , 10000.        , 10000.        , 10000.        ,
10000.        , 10000.        , 10000.        , 10000.        ,
10000.        , 10000.        , 10000.        , 10000.        ])
```

[111]:
```
#Now that have the array - check to see if there is anything within 1meter of␣
 ↪Misty, and
depthThresholdtoWarn = 1000; #Set to 1000mm or 1m

if (min(sumSliceDistance) < depthThresholdtoWarn):
    print("Watch out!")
else:
    print("All Good! Keep Driving")
```

```
All Good! Keep Driving
```

[ ]:
```

```