AR ASSISTED AP PLACEMENT

INVENTORS:

JOSHUA F. REDMORE
SHAFI KAHN

Description

Use AR to overlay a channel-specific antenna pattern on an access point (AP) in order to guide installation. The AR device will capture the channel and band in-use by the AP and apply the correct pattern. Additionally, the user can select any channel or band in order to compare performance and coverage.

An additional use case can be to plan for a future install, by overlaying these patterns on a sticker or some other AR trigger that can substitute for the AP. The user could select from a wide range of APs in order to choose the most appropriate one for the specific installation location.

Function flow

1. AR application launched
2. Select SSID (optionally – read the currently associated SSID)
3. AR app recognizes the AP (via markers or straight-up device recognition)
4. AR app polls the system to discover what channel the SSID is on
5. AR app loads the channel-specific RF pattern and overlays the image onto the AP

Optional steps:

1. User could select the channel (and, by implication, the band) they would like to see, for potential comparison / optimization.
2. User could place a sticker where the proposed AP placement will be, in order to cycle through different channels or even different APs, in order to establish the best device and orientation for a specific deployment.

Background

We have a library of antenna patterns from various Wi-Fi devices - primarily Wi-Fi enabled cable modems. Each antenna pattern describes where the radiated energy will be directed (Total Radiated Power - TRP) or in what direction the device is most sensitive (Total Isotropic Sensitivity - TIS). See page 4 of the attachment "TIS-ch1.pdf" for an example of the antenna patterns we capture.

Wi-Fi antennas are not isotropic and can be severely directional. This information is invisible to the customer, yet critical to their QoE. Additionally, antenna patterns will vary significantly between 2.4, 5, and 6 GHz radios. Because of these factors, how / where / what direction you install your APs will directly affect the performance of the WLAN.

You may have an installation with IoT devices on 2.4 GHz (say, to gain the extra range to be able to reach your Wi-Fi doorbell or sprinkler controller) and primary user devices like TVs and Laptops on 5 GHz. This system will show you the patterns on all channels and bands to let you select the best orientation of the AP.

CTIA Report (SNS_W-LAN 11n 2.4 GHz_ch1_tot)

Test Information

Test Method:	Sensitivity Mobile Phone
Test Condition:	FS: Free Space
EUT Identification:	MAC Address: 0A:19:70:C3:C3:00
Radio Link:	W-LAN 11n 2.4 GHz; Channel 1 (2412.000 MHz)
Test Time:	Start: 7/12/2021 11:58:05 AM; Stop: 7/12/2021 12:27:00 PM
WLAN Connectors:	In: RF3 COM (35.0 dB), Out: RF3 OUT (16.0 dB)
Cal Data Phi:	20.58 dB (Chamber Cal MA1 HOR, WLAN_CMW RF1OUT to MA1 HOR)
Cal Data Theta:	19.83 dB (Chamber Cal MA1 VER, WLAN_CMW RF1OUT to MA1 VER)

OTA Evaluation Results

Total Isotropic Sensitivity	-61.41 dBm
Minimum Isotropic Sensitivity	-68.31 dBm
Directivity	6.90 dBi
NHPIS 45°	-59.54 dBm
NHPIS 45° / TIS	-1.87 dB
NHPIS 45° / TIS	65.04 %
NHPIS 30°	-57.64 dBm
NHPIS 30° / TIS	-3.77 dB
NHPIS 30° / TIS	41.98 %
NHPIS 22.5°	-56.39 dBm
NHPIS 22.5° / TIS	-5.02 dB
NHPIS 22.5° / TIS	31.48 %
UHIS	-60.71 dBm
UHIS / TIS	-0.70 dB
UHIS / TIS	85.19 %
LHIS	-53.12 dBm
LHIS / TIS	-8.29 dB
LHIS / TIS	14.81 %
PIGS (0-120°)	-61.25 dBm
PIGS / TIS	-0.16 dB
PIGS / TIS	96.36 %
Front/Back Ratio	18.80
PhiBW	92.3 deg
PhiBW Up	77.6 deg
PhiBW Down	14.7 deg
ThetaBW	29.9 deg
ThetaBW Up	15.1 deg
ThetaBW Down	14.8 deg
Boresight Phi	120 deg
Boresight Theta	30 deg
Maximum Sensitivity	-45.40 dBm
Minimum Sensitivity	-68.31 dBm
Average Sensitivity	-61.88 dBm
Max/Min Ratio	22.91 dB
Max/Avg Ratio	16.48 dB
Min/Avg Ratio	-6.43 dB
Worst Single Value	-23.58 dBm
Worst Position	Azi = 0 deg; Elev = 0 deg; Pol = Phi
Best Single Value	-67.48 dBm

Best Position	Azi = 120 deg; Elev = 30 deg; Pol = Phi

SNS_W-LAN 11n 2.4 GHz_ch1_tot

Azimuth (deg)	Elevation 0 deg (dBm)	Elevation 30 deg (dBm)	Elevation 60 deg (dBm)	Elevation 90 deg (dBm)	Elevation 120 deg (dBm)	Elevation 150 deg (dBm)
0.00	-62.23	-64.91	-64.20	-60.88	-56.45	-50.72
30.00	-62.23	-64.66	-63.20	-56.56	-52.28	-49.12
60.00	-62.23	-64.20	-63.95	-58.26	-51.62	-50.66
90.00	-62.23	-62.20	-63.45	-59.91	-54.75	-52.34
120.00	-62.23	-68.31	-62.37	-60.62	-55.16	-54.26
150.00	-62.23	-66.83	-60.90	-58.73	-53.01	-50.41
180.00	-62.23	-64.00	-62.31	-56.41	-53.58	-50.68
210.00	-62.23	-63.37	-63.76	-57.63	-49.38	-50.31
240.00	-62.23	-66.76	-67.25	-62.48	-56.65	-45.40
270.00	-62.23	-66.66	-67.16	-60.76	-57.06	-54.65
300.00	-62.23	-65.12	-65.43	-61.04	-53.87	-51.15
330.00	-62.23	-62.77	-62.29	-58.76	-54.45	-51.48
360.00	-62.23	-64.91	-64.20	-60.88	-56.45	-50.72

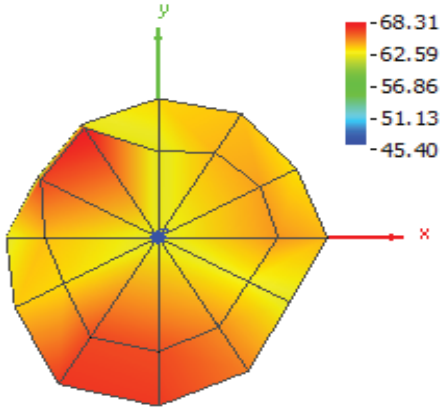
SNS_W-LAN 11n 2.4 GHz_ch1_phi

Azimuth (deg)	Elevation 0 deg (dBm)	Elevation 30 deg (dBm)	Elevation 60 deg (dBm)	Elevation 90 deg (dBm)	Elevation 120 deg (dBm)	Elevation 150 deg (dBm)
0.0	-23.58	-62.48	-61.98	-57.48	-52.48	-40.98
30.0	-23.58	-62.98	-60.98	-48.98	-32.98	-43.48
60.0	-23.58	-61.98	-59.98	-53.98	-48.48	-46.98
90.0	-23.58	-59.98	-59.48	-57.48	-47.98	-43.48
120.0	-23.58	-67.48	-60.98	-57.48	-51.48	-49.98
150.0	-23.58	-66.48	-59.98	-57.98	-51.48	-47.98
180.0	-23.58	-62.98	-61.48	-51.48	-48.98	-50.48
210.0	-23.58	-60.48	-59.48	-54.98	-45.98	-49.48
240.0	-23.58	-62.48	-60.48	-56.48	-49.48	-36.98
270.0	-23.58	-62.98	-63.48	-55.48	-49.48	-47.48
300.0	-23.58	-59.48	-51.98	-49.48	-50.98	-43.98
330.0	-23.58	-53.48	-43.98	-53.48	-50.48	-45.48
360.0	-23.58	-62.48	-61.98	-57.48	-52.48	-40.98

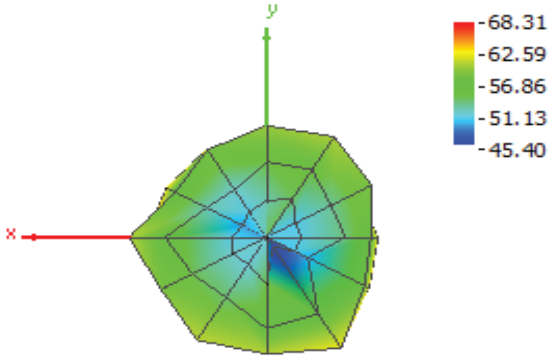
SNS_W-LAN 11n 2.4 GHz_ch1_theta

Azimuth (deg)	Elevation 0 deg (dBm)	Elevation 30 deg (dBm)	Elevation 60 deg (dBm)	Elevation 90 deg (dBm)	Elevation 120 deg (dBm)	Elevation 150 deg (dBm)
0.0	-62.23	-61.23	-60.23	-58.23	-54.23	-50.23
30.0	-62.23	-59.73	-59.23	-55.73	-52.23	-47.73
60.0	-62.23	-60.23	-61.73	-56.23	-48.73	-48.23
90.0	-62.23	-58.23	-61.23	-56.23	-53.73	-51.73
120.0	-62.23	-60.73	-56.73	-57.73	-52.73	-52.23
150.0	-62.23	-55.73	-53.73	-50.73	-47.73	-46.73
180.0	-62.23	-57.23	-54.73	-54.73	-51.73	-37.23
210.0	-62.23	-60.23	-61.73	-54.23	-46.73	-42.73
240.0	-62.23	-64.73	-66.23	-61.23	-55.73	-44.73
270.0	-62.23	-64.23	-64.73	-59.23	-56.23	-53.73
300.0	-62.23	-63.73	-65.23	-60.73	-50.73	-50.23
330.0	-62.23	-62.23	-62.23	-57.23	-52.23	-50.23
360.0	-62.23	-61.23	-60.23	-58.23	-54.23	-50.23

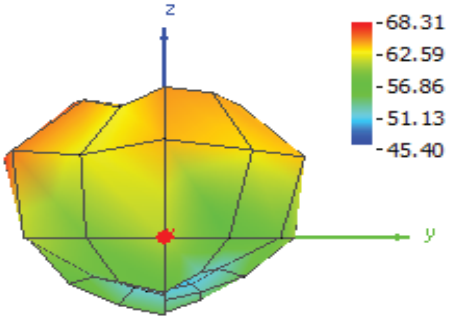
Theta = 0, Phi = 0



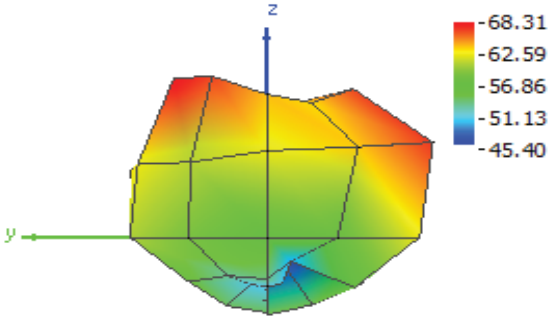
Theta = 180, Phi = 0



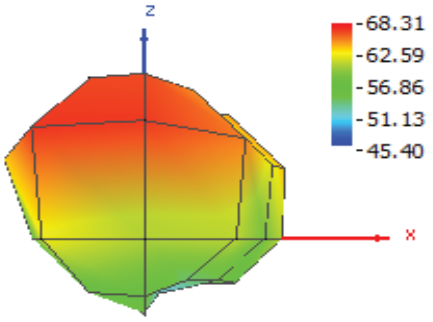
Theta = 90, Phi = 0



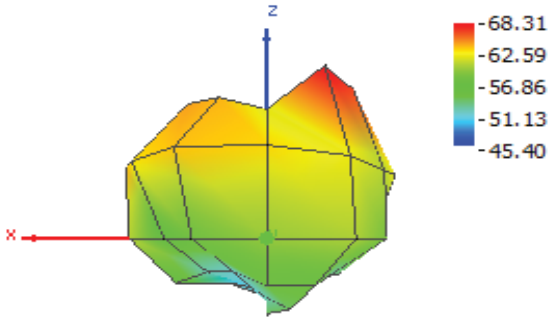
Theta = 90, Phi = 180



Theta = 90, Phi = 270



Theta = 90, Phi = 90



```
#!/Users/dlavender/anaconda/bin/python

# Author: Dan Lavender
# Date:   Feb 19, 2015
# Desc:   Convert x,y,z formatted piles for 3D printer plotting.
#
#         Files are formatted as collections of x,y,z values enclosed in
#         double quotes. The x,y,z values are comma delimited.
#         Each record is prepended by three commas.
#         Each record, collection set contains 25 xyx tuples.
#         An example record is:
#
#          ,,,"0.00, 0.00, 35.73","0.00, 0.00, 35.73","0.00, 0.00, 35.73","0.00,
0.00, 35.73","0.00, 0.00, 35.73","0.00, 0.00, 35.73","0.00, 0.00, 35.73","0.00,
0.00, 35.73","0.00, 0.00, 35.73","0.00, 0.00, 35.73","0.00, 0.00, 35.73","0.00,
0.00, 35.73","0.00, 0.00, 35.73","0.00, 0.00, 35.73","0.00, 0.00, 35.73","0.00,
0.00, 35.73","0.00, 0.00, 35.73","0.00, 0.00, 35.73","0.00, 0.00, 35.73","0.00,
0.00, 35.73","0.00, 0.00, 35.73","0.00, 0.00, 35.73","0.00, 0.00, 35.73","0.00,
0.00, 35.73","0.00, 0.00, 35.73"
#
#         The output format is a series of x,y,z points and a pair of endpoints for

#         the line segment
#         Example output records for the "_Point" data:
#             _Point 8.09, 2.17, 31.26
#             _Point 8.33, 4.81, 35.89
#         Example output record for the "_Line" data:
#             _Line 8.09, 2.17, 31.26 8.33, 4.81, 35.89

# Sample command line call:
#         python ./Transform3DPrint.py -i PointCloud2.txt -o outfile.dat -p yes

import numpy as np
import random
import matplotlib.pyplot as plt
import pylab
import pandas as pd
from mpl_toolkits.mplot3d import Axes3D
import matplotlib as mpl

import sys
import argparse # enables parsing the command line



from matplotlib.ticker import LinearLocator, FormatStrFormatter
```

```
class attr: # from command line args
   inFile = "undefined"
   outFile = "undefined"
   outFp = 0
   inFp = 0
   plot_yn = "No"
# End of class attr

#
# The key function for the script. Does the input, transform, output, and plotting
#
def doTransform():
   myList = list()
   pointList = list()

   # If plotting selected, Set up the environment for plotting
   if (attr.plot_yn == 'YES'):
      fig = plt.figure()
      ax = fig.gca(projection='3d')

   # create list, remove the prepended three commas; remove the double quotes,
   # split the remaining string into individual values on the commas
   for i in attr.inFp:
      myList.append(i[3:-1].replace('"','').rstrip().split(','))

   # Process the current, transformed data record from the input file
   for curLine in myList:
      pointList = list()

      # Write the points list to the outfile
      for k in range(0, len(curLine), 3):
         outStr =  curLine[k] + "," + curLine[k + 1] + "," + curLine[k + 2] + "\n"
         outStr =  "_Point " + outStr.replace(', ', ',')
         attr.outFp.write(outStr)

         # Save the points for plotting the polygons
         pointList.append([float(curLine[k]), float(curLine[k + 1]),
float(curLine[k + 2])])

      # Write the line segments list to the outfile
      lineLen = len(curLine)
      for k in range(0, lineLen - 6, 3):
         outStr = curLine[k] + "," + curLine[k + 1] + "," + curLine[k + 2] + " " +
\
                     curLine[k + 3] + "," + curLine[k + 4] + "," + curLine[k + 5] +
"\n"
         outStr = "_Line " + outStr.replace(', ', ',')
         attr.outFp.write(outStr)

      # Write the last line segment for the current list of x,y,z tuples to the
```

```
outfile
      # outStr = "_Line " + curLine[lineLen - 6] + "," + curLine[lineLen - 5] + ","
+ \
                # curLine[lineLen - 4] + " " + curLine[lineLen - 3] + \
                # "," + curLine[lineLen - 2] + "," + curLine[lineLen - 1] + "\n"
      outStr = curLine[lineLen - 6] + "," + curLine[lineLen - 5] + "," + \
                curLine[lineLen - 4] + " " + curLine[lineLen - 3] + \
                "," + curLine[lineLen - 2] + "," + curLine[lineLen - 1] + "\n"
      outStr = "_Line " + outStr.replace(', ', ',')
      attr.outFp.write(outStr)

      # prep the data for plotting by putting in a Pandas DataFrame
      df = pd.DataFrame(data=pointList)

      # create the lists of x, y, and z values
      x = df[0]; y = df[1]; z = df[2]

      # Plot the data in 3D using matplotlib
      if (attr.plot_yn == 'YES'):
         ax.plot(x, y, z, label='Closed polygons')

   # Show the generated plots.
   if (attr.plot_yn == 'YES'):
      # Set up the environment for plotting
      # fig = plt.figure()
      # ax = fig.gca(projection='3d')

      # prep the data for plotting by putting in a Pandas DataFrame
      # df = pd.DataFrame(data=pointList)

      # create the lists of x, y, and z values
      # x = df[0]; y = df[1]; z = df[2]

      # Plot the data in 3D using matplotlib
      # ax.plot(x, y, z, label='Closed polygons')
      plt.show()
# End of doTransform()

################################################################
# Main entry point into the program
################################################################

def main(argv):
   # Parse the command line arguments
   parser = argparse.ArgumentParser(description='Transform 3D sample data for
printing and visualizations.')
   parser.add_argument('-i','--inFile', help='Filename for input sample data',
required=True)
   parser.add_argument('-o','--outFile', help='Filename for writing transformed
data.', required=True)
```

```
   parser.add_argument('-p','--plot_yn', help="Yes(plot to screen) or No (don't
plot to screen).", required = "True")
   args = vars(parser.parse_args())

   # Save the command line arguments for later use
   attr.inFile = args['inFile']
   attr.outFile = args['outFile']
   attr.plot_yn = args['plot_yn'].upper()

   try:
      attr.inFp = open(attr.inFile, 'r')
      print attr.inFile, "opened successfully."

      attr.outFp = open(attr.outFile, 'w')
      print attr.outFile, "opened successfully."

      doTransform()
   except IOError:
      print "Error opening data files."

   attr.inFp.close()
   attr.outFp.close()
# main()

#
# Program entry point
#

if __name__ == "__main__":
   main(sys.argv[1:])

# End.
```