**Real-Time Location System Analysis and Applications for Smart Cities**

Casey Martinez

Kyle Thomas

MSDS 7333 – Quantifying the World – Case Study 1 (Unit 2)

01/17/2019

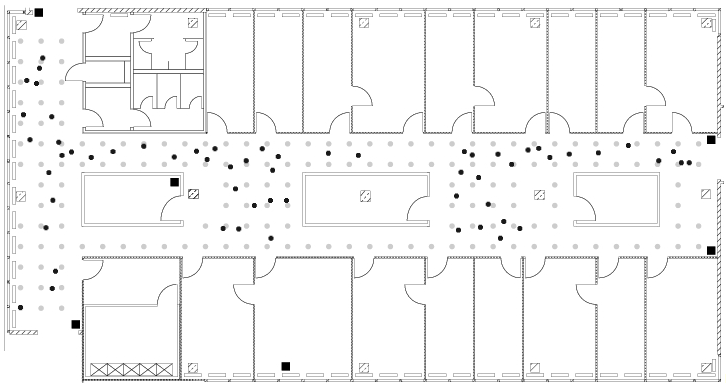
# Introduction

With the continued development of Internet of Things (IoT) devices capable of recording and sharing data to the network in real-time, truly smart spaces can be developed. These sensors and networks exchange information using indoor radiofrequency Wi-Fi and/or outdoor GPS which can be used to create Real-Time Location Systems (RTLS). These RTLS systems can provide great value in every industry from utilities services to transportation and manufacturing. For example, energy-efficient buildings can track occupants and automatically adjust lighting and climate controls when people are not around to need them.

This is possible with passive and semi-passive tags and readers that can track their relationship to one another. Using parameters such as relative signal strength (RSS) the location of the tags can be calculated. In a controlled test environment, RSS decays linearly with the log of the distance according to the Friis Free Space Equation (Fig. 1.2). In practice, the unique characteristics of the space, and people in them, add significant noise.

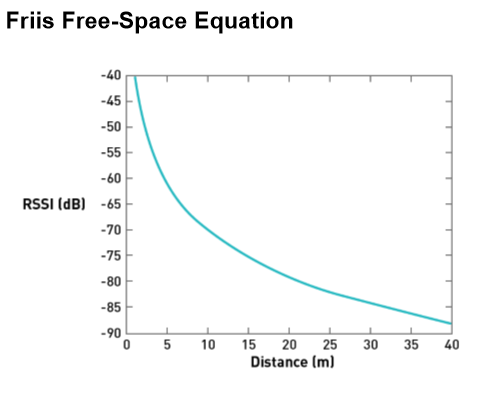
In this case study, Wi-Fi access points are setup on a floor of a building shown below as black squares (Figure 1.1). The database method was used for location prediction using a handheld receiver, measurements were taken of RSS from all access points in the area along with location. The space was mapped out systematically as shown in the grey dots below. This training dataset was used to analyze the RSS for each access point and create a model to predict location. Test data was also obtained for model validation which is shown below as black circles.

Fig. 1.1: Floor Plan of the Test Environment



*In this floor plan, the fixed access points are denoted by black square markers, the offline/training data were collected at the locations marked by grey dots, and the online measurements were recorded at randomly selected points indicated with black dots. The grey dots are spaced one meter apart.*

**Fig 1.2**



# Background

The analysis of conducted herein utilizes the Wi-Fi data gathered from the University of Mannheim [1]. More specifically, the data contains information on RSS as measured from a hand-held device from 166 known locations on the same floor in the building. In addition to signal strength and the x-y location of each of the 166 points there is also information on the time each reading was take, the MAC address of the Wi-Fi router, the orientation of the signal, and the MAC address of the scanner (hand-held device). The information from the data set can be used to create a RTLS. More specifically, it can be used to create and Indoor Positioning System (IPS).

There are many applications for IPS in Smart Buildings and Smart Cities. An IPS can be used to track valuable assets to ensure that they are in the correct location or prevent theft. Furthermore, they can be used to for access management to secure areas by ensuring that the WiFi signal near a secure area is coming from an approved device or person. There are also numerous life safety application for an IPS. Such a system could be used to track employees' locations in a dangerous situation such as a fire or lock down event so that resources can effectively help those that are in danger.

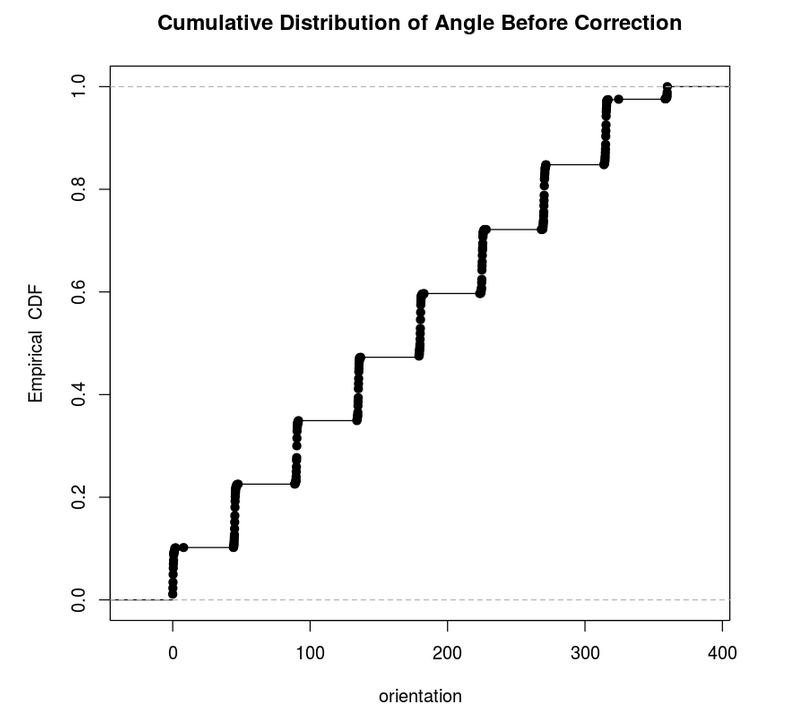
For an IPS, or any RTLS, to be useful they must provide reliable and actionable intelligence in a timely manner. Therefore, before any model is developed an analysis of the data must be done to ensure that the model is being built with reliable data. This analysis includes an examination of each feature in the dataset and checking for anomalies that might confound additional analysis or modeling. Of special interest is an analysis of signal strength as it will be the response variable in the for the location prediction algorithm.

# Methods

The data used for this analysis existed as a flat text file that required significant processing before it could be used for analysis and location prediction. This process consisted of some basic data cleansing techniques, mostly string processing, so that each feature could be directly related to a MAC address. Further processing was done to convert features that were numeric in nature to a numeric format from the base character data from the text file.

Once data had been converted into a numeric format basic data exploration could begin to see if further processing was necessary. During the exploratory data analysis it was discovered that the orientation feature was continuous in nature. According to the documentation, this feature should be an enumerated point with eight distinct values [1]. These values were converted to the eight values that were expected based on the documentation, and as such have been converted from their raw format into a more useful feature that can be used in location prediction. Figure 3.1 and 3.2 show how this feature was changed.

*Figure 3.1 - Empirical CDF of orientation before correction. Notice how there is evidence of nine distinct values, with some noise between individual levels.*



*Figure 3.2 Boxplots of Orientation After Correction. There are now only eight values with no noise between values. This matches the expectations set out in the documentation.*

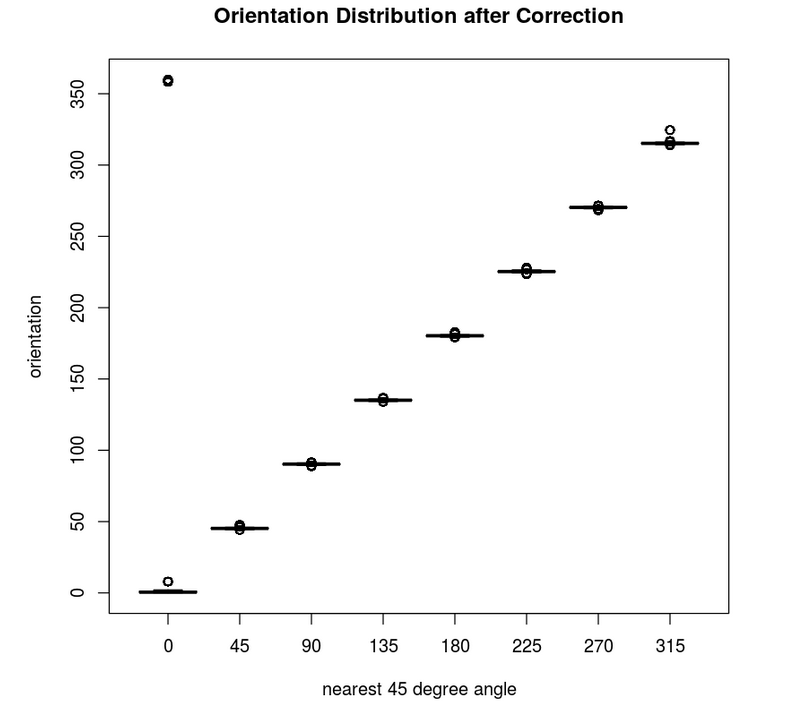
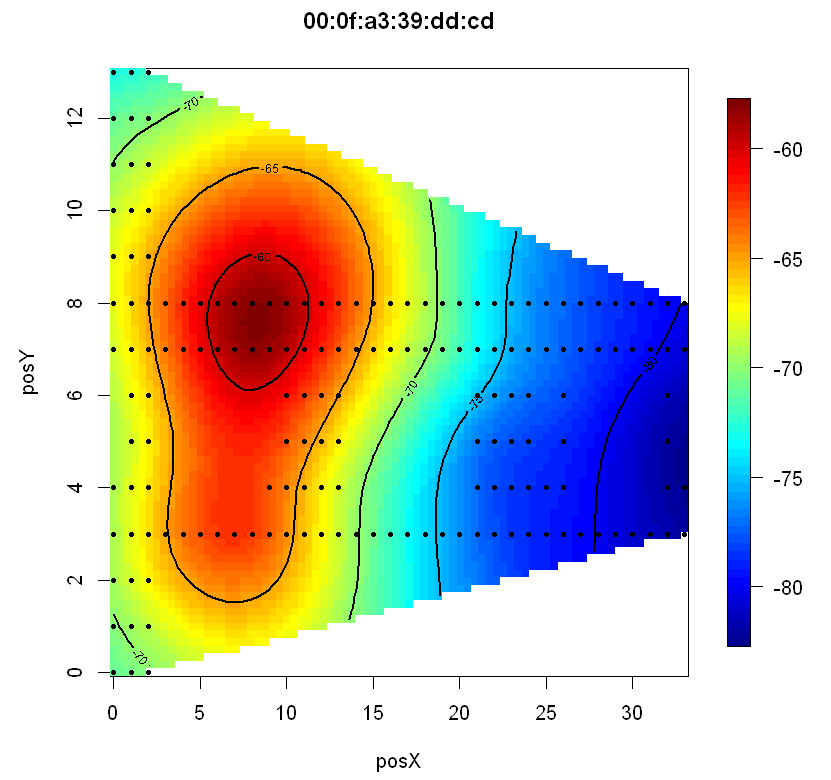
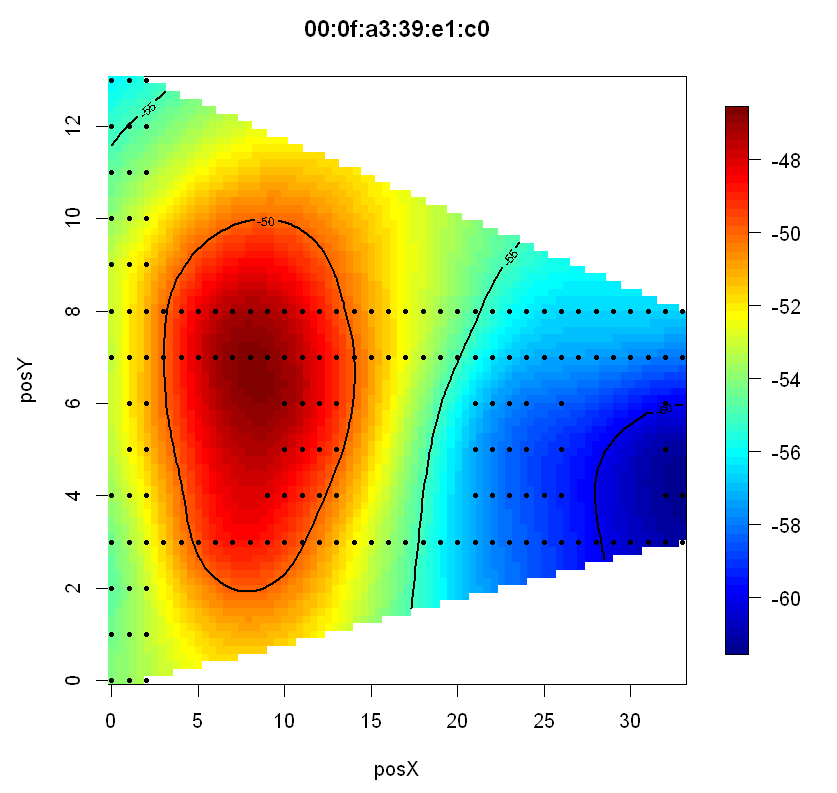
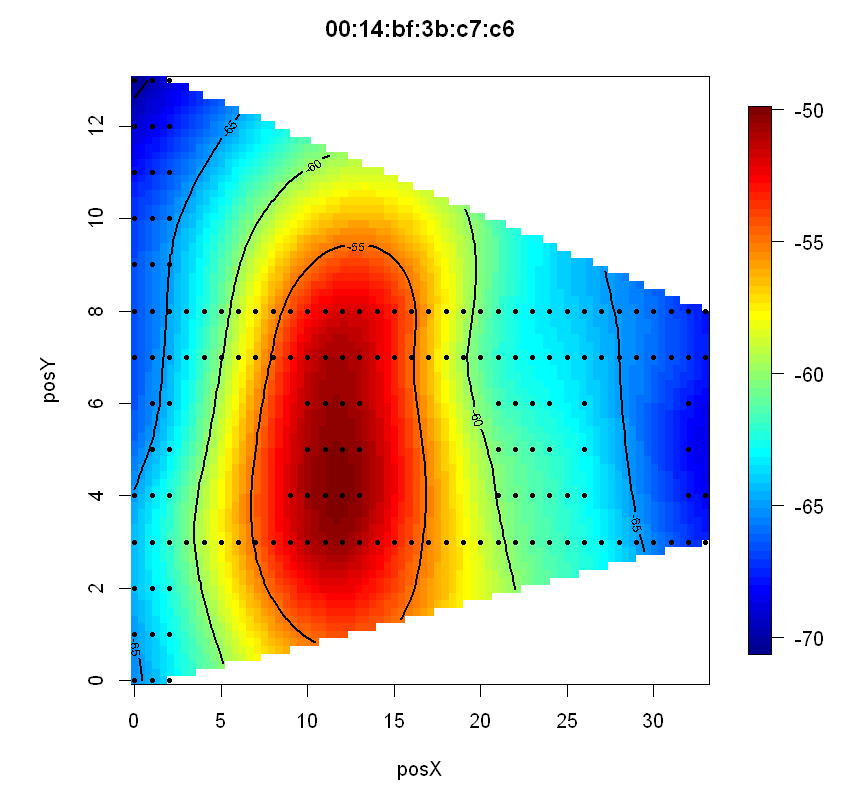
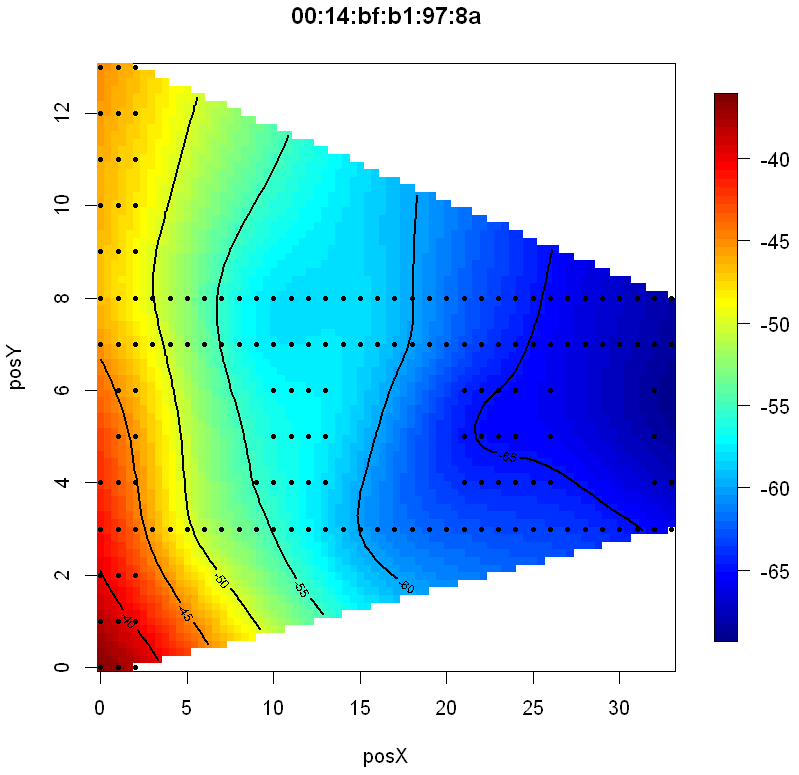
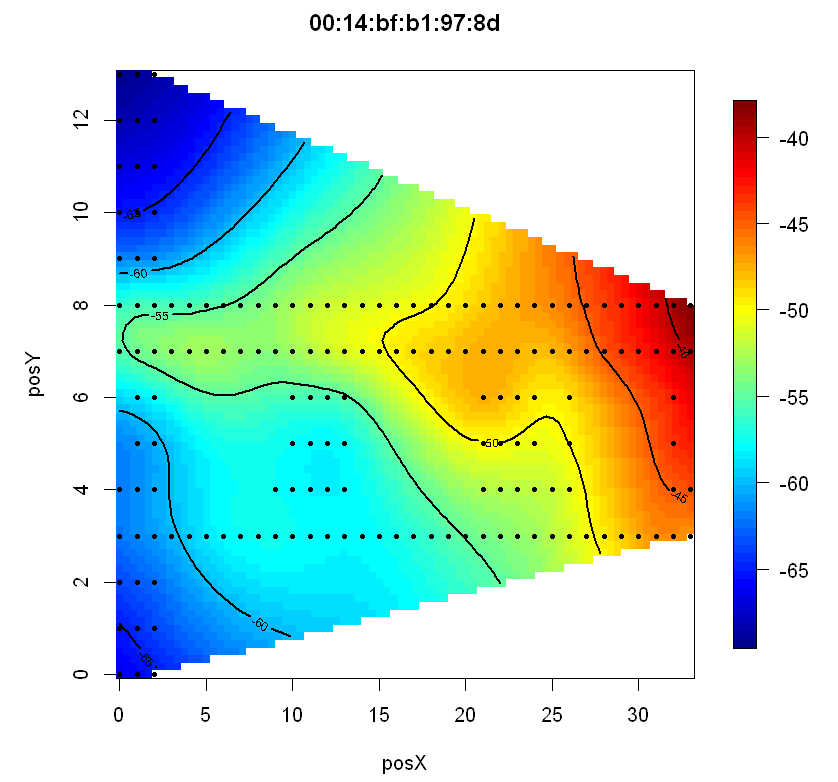
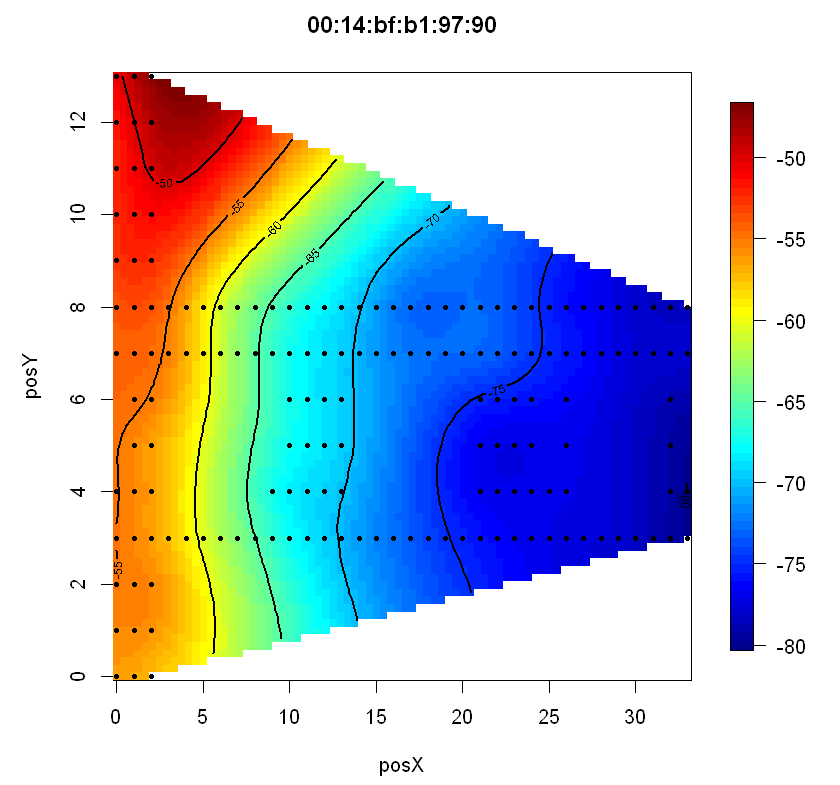


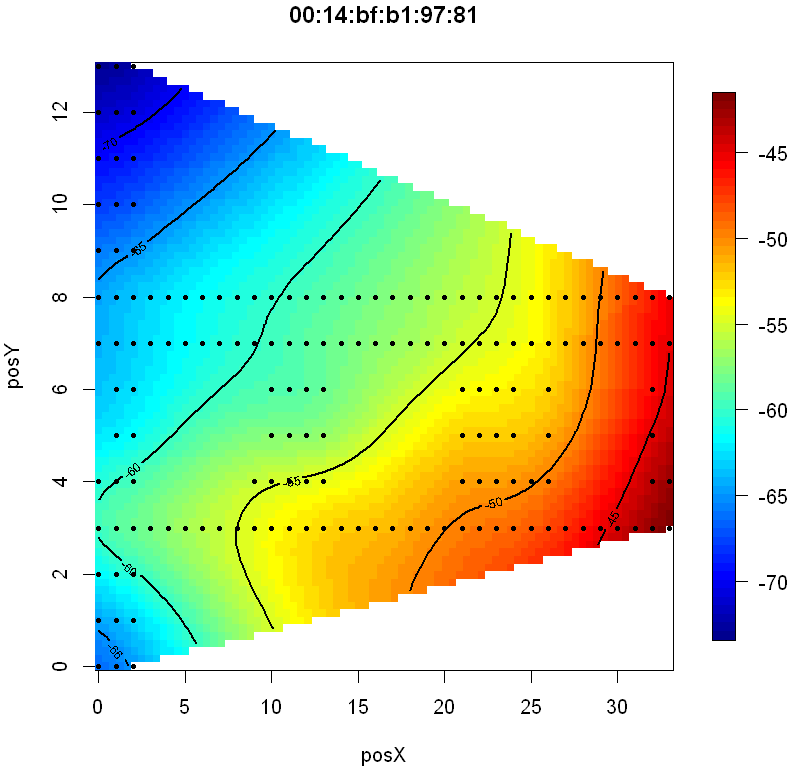
Figure 3.3 shows the effect of device orientation on RSS for the same access point. There are large differences in RSS based on device orientation. This difference indicates that orientation will play a large role in the analysis of RSS and must be considered when building the IPS algorithm. While examining RSS as a function of device orientation it was discovered that two MAC addresses had a similar signature etc. These two MAC addresses were 00:0f:a3:39:e1:c0 and 00:0f:a3:39:dd:cd.

*Fig. 3.3: Spatial distribution of RSS by MAC address*



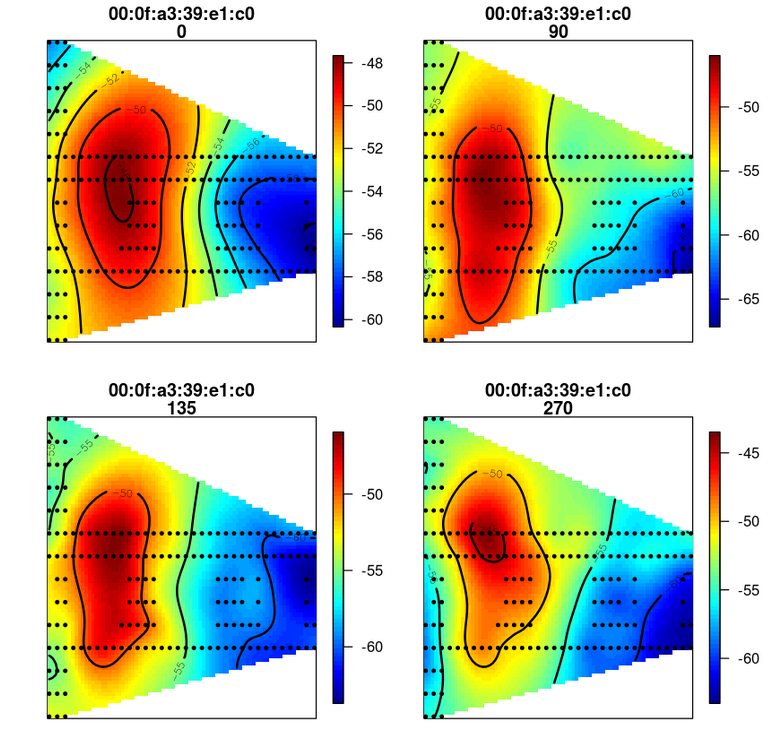




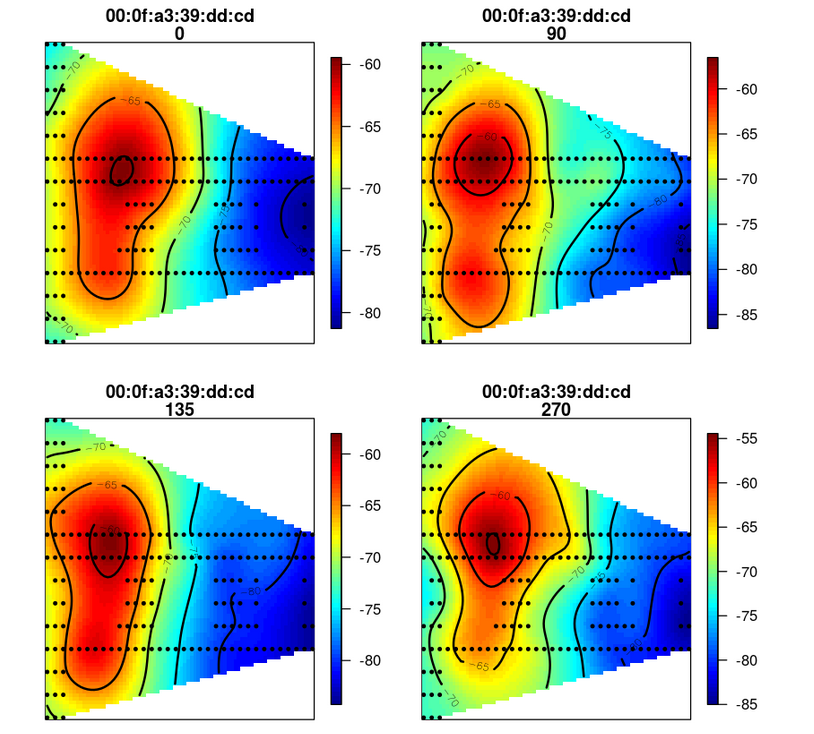


The IPS will rely heavily on the signal strength and therefore a thorough analysis of this variable is warranted. Specifically, there were concerns related to the two access points as they shared very similar RSS heatmaps at the same angles [1]. Figure 3.4 shows a heatmap of RSS by four different orientations associated with MAC 00:0f:a3:39:e1:c0. Figure 3.5 shows a similar heatmap for MAC 00:0f:a3:39:dd:cd. Based on these eight heatmaps these two MAC addresses are correlated to the same access point.

*Fig. 3.4 Spatial distribution of RSS by angle for MAC 00:0f:a3:39:e1:c0. The title of each plot shows the MAC address of the access point and the orientation of the signal.*



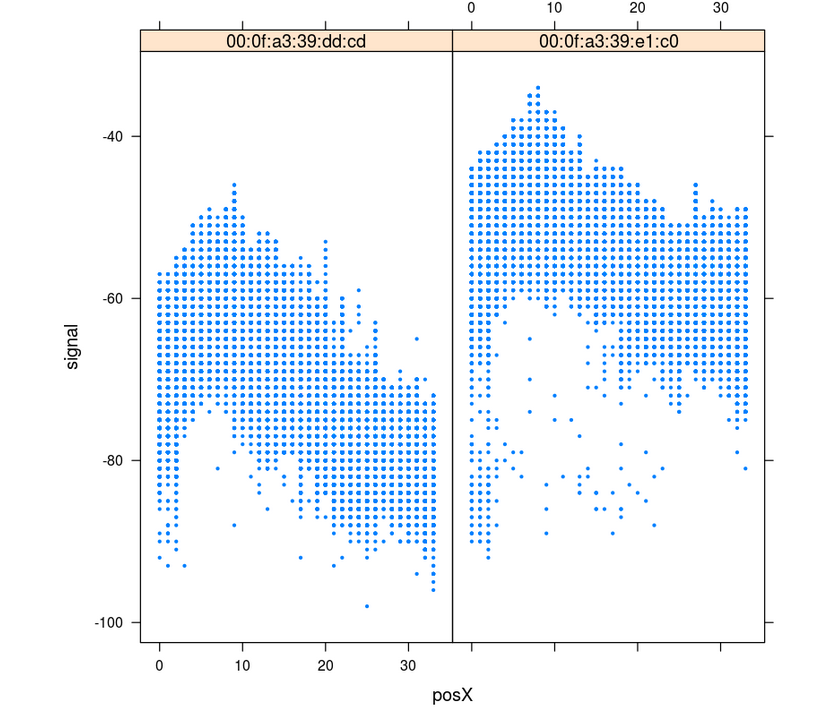
*Figure 3.5 Spatial distribution of RSS by angle for MAC 00:0f:a3:39:dd:cd. The title of each plot shows the MAC address of the access point and the orientation of the signal.*



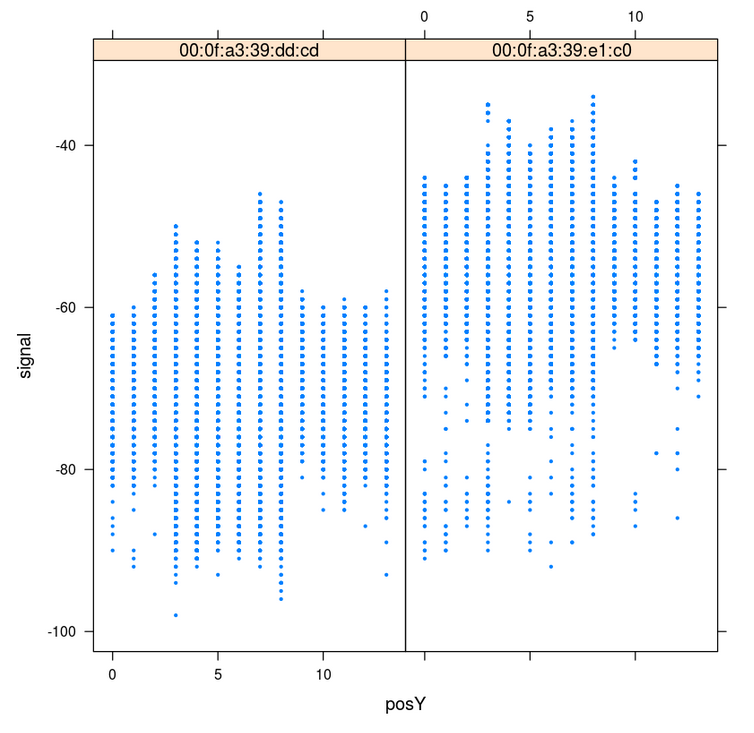
This leads to several interesting questions. Should one of these two MACs be tossed out? If so, which one should be removed from the data set? Alternatively, will use of both MAC addresses in the dataset improve the predictions of the IPS? If one of the two MAC addresses should be dropped it should be based on consideration of signal strength and the overall utility of the MAC address to the IPS prediction algorithm.

An analysis of signal strength for these two MAC addresses are shown in Figure 3.6 and Figure 3.7. These two scatter plots show the signal strength of each MAC address in relation to the X and Y coordinates of the floor space respectively. These plots reveal that MAC 00:0f:a3:e1:c0 has a much stronger signal than 00:0f:a3:39:dd:cd for both X and Y coordinates. It is worth noting that the distribution of the signal strength is similar, but shifted based on the magnitude of the signal strength. Note that stronger signal strength is indicated by values closer to zero (or lower absolute values), which weaker signal strength is indicated by values further from zero (or higher absolute values) [1].

*Figure 3.6: Scatter Plot of RSS vs X Position by MAC Address.*

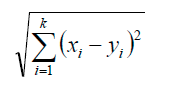


*Figure 3.7: Scatter Plot of RSS vs Y Position by MAC Address.*



As the end goal is make a IPS consideration also needs to be given to each access point’s impact on the prediction algorithm. For this analysis the algorithm used to predict location is “K Nearest Neighbors” or KNN. KNN works by comparing a new observation to known data points. In this case, signal strength and angle will be used to determine the nearest neighbors. Then, the closest “K” number of neighboring points are assed. In this analysis, closeness is defined by Euclidian Distance. The formula for Euclidian Distance is shown Figure 3.8 [6] below:

Figure 3.8: Euclidean Distance Formula



The aggregated results of the known locations are then used to predict the new location based on signal strength and angle. A KNN analysis was conducted with both MAC address, each of the MAC address separately, and a weighted KNN to determine the best model to use for the IPS. The best model will the model with the lowest error using cross-validation. Cross validation was used to ensure that the model was not overfit and would generalize well. In this analysis an 11 fold cross validation was utilized [1].

First, a KNN analysis was completed using all access points. The KNN algorithm is straight forward, however there was a challenge in choosing the correct number for K such that error was minimized. To do this, an algorithm was used that iterated between 1 and 20 as values of K. The algorithm calculates the error for each value of K and returns an array of the errors. This allows for the error to be plotted as a function of the number of neighbors used in the algorithm. The best value of K is the one with the lowest error.

Will removing one of the MACs improve the prediction power of the KNN algorithm? In order to test this idea, the same algorithm is used but on two different sets of data where only one of the two identified MACs has been removed from the test data. First the analysis was re-run while dropping MAC 00:0f:a3:39:dd:cd from the data set. The KNN analysis was conducted once more after dropping MAC address 00:0f:a3:39:e1:c0 (and keeping 00:0f:a3:39:dd:cd). After searching once again over a wide variety of neighbors an optimal solution was found for each data set.

The process of predicting the new observation’s spatial coordinates is based on calculating the distance between the RSS of the new observation and all the reference observations in the database. The neighboring reference observations, in RSS space, are then chosen and used to predict the new observation’s spatial coordinates. The distance measured from the new observation to the neighboring reference points can also be used to weight the neighbors so that the closer neighbors will have a larger impact on the prediction.

# Results

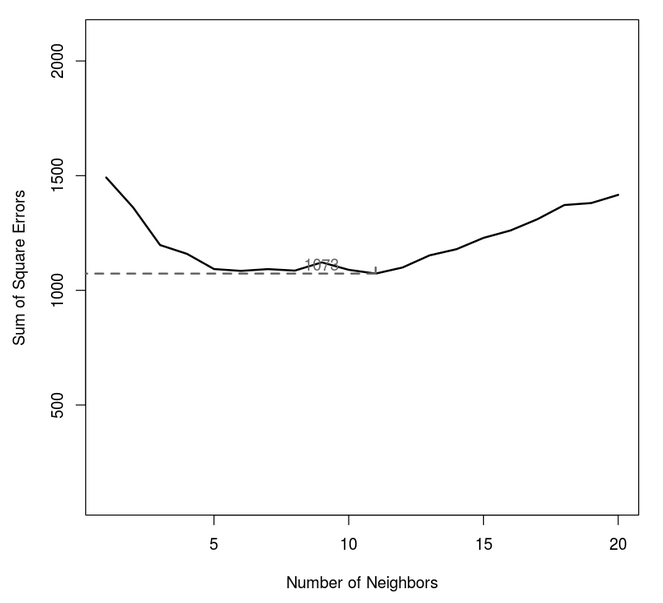
When using all MAC address the value of K that minimizes the error is four with a minimum Sum of Square error of 1147. Figure 3.10, below, shows the error by the number of neighbors used in the KNN algorithm.

*Figure 3.10: Error plot for each iteration of K in the KNN algorithm using all MAC address.*



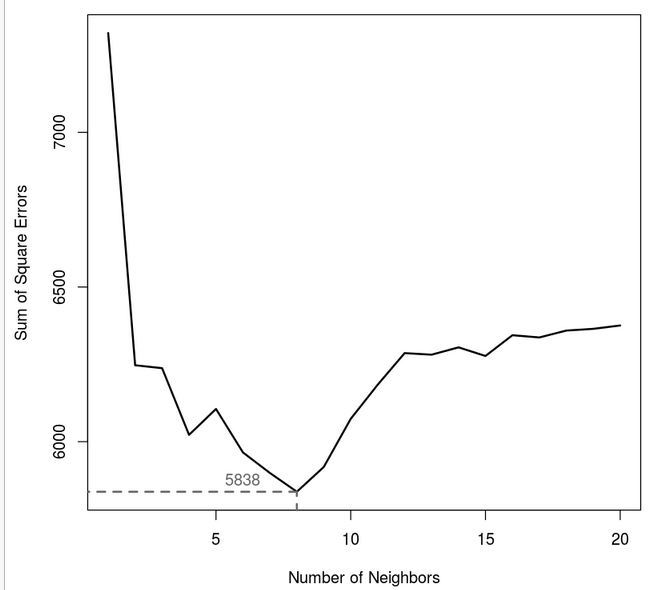
The optimal solution found for data set that contained MAC address 00:0f:a3:39:e1:c0 had a minimum Sum of Square Error of 1073 with a value of K equal to 11. This is an improvement over using all MAC addresses. Figure 3.11 shows the error by number of neighbors for the analysis without for MAC address 00:0f:a3:39:e1:c0.

*Figure 3.11: Error plot for each iteration of K in the KNN algorithm after dropping MAC address 00:0f:a3:39:dd:cd from the data set.*



The optimal solution found for data set that contained MAC address 00:0f:a3:39:dd:cd had a minimum Sum of Square Error of 5838 with a value of K equal to eight. This represents the most error prone model examined thus far. Figure 3.12 shows the error by number of neighbors for the analysis without MAC address 00:0f:a3:39:e1:c0.

*Figure 3.12: Error plot for each iteration of K in the KNN algorithm after dropping MAC address 00:0f:a3:39:e1:c0 from the data set.*



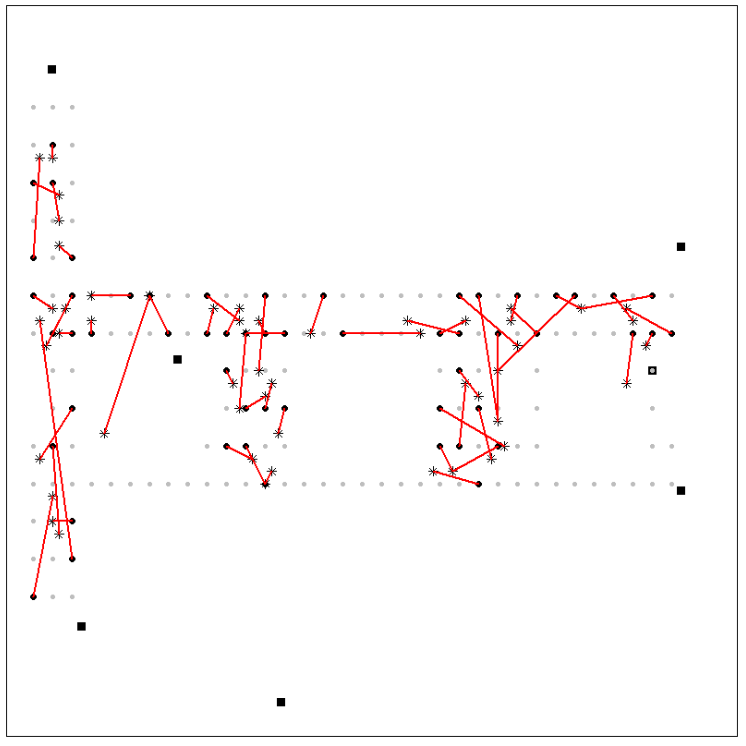
After conducting all three KNN analysis a clear winner has emerged in terms of predictability for the IPS. The best solution was found by keeping MAC address 00:0f:a3:39:e1:c0 in the data set while dropping MAC address 00:0f:a3:39:dd:cd. Using both MAC addresses was better than using 00:0f:a3:39:dd:cd alone, but was not as accurate as dropping the same MAC address from the dataset completely. Table 3.1 shows a quick comparison of the KNN results.

*Table 3.1: A comparison of KNN models by MAC address.*

|  |  |  |
| --- | --- | --- |
| MAC ADDRESS | Optimal Value of K | Sum of Square Error (lowest) |
| BOTH | 4 | 1147 |
| 00:0f:a3:39:dd:cd | 8 | 5838 |
| 00:0f:a3:39:e1:c0 | 11 | 1073 |

The calculated RSS distance was used in the weighted average of the k nearest neighbors X and Y positions but there was little change in the results. The added benefits of the weighted average should be significant based on the relationship between RSS and distance described earlier. Constructive and destructive signal interference can add significant noise to the RSS and distance relationship. Below is a plot of the unweighted predicted locations and the corresponding actual positions. The red line connects them and shows how the floor geometry can impact the accuracy. Note that the middle of the floor seems to show the best accuracy where the left and right sides have longer red lines.

*Fig. 3.11 Floor Plan of Predicted and Actual*



# Conclusion

The approach described here constitute an accurate framework in creating an Indoor Positioning System. The relationships between access points, relative signal strength, and device orientation were all explored here. The analysis of access points and IPS accuracy suggest the access points with a large overlap in coverage can decrease the accuracy of the IPS by an estimated 6.9 percent.

There are many areas of future investigation for creating a more robust IPS. There was some evidence that a log transformation of the RSS data could prove to be useful [1]. Transforming the RSS variable and rerunning the analysis for comparison might yield a more accurate model. In addition, this analysis calculated distance between points using Euclidian distance. However, there are other distance metrics that could be used, such as Manhattan distance. This could be especially interesting as the 166 measurement sites resemble a city block grid. Manhattan distance might provide additional insights as it is based on X and Y grid distances that resemble city blocks.

Once the final model is chosen there are many ways in which an IPS can be used in Smart Building applications. Tags could be attached to important equipment and tracked throughout a building. And IPS could also be used to automatically track when employees have entered and left a building to generate logs of access and drive efficient use of the building. In theory they could even be used to track how long an employee is at their desk to track productivity or billing hours for a project. Whatever the use maybe, the model used to predict location must be accurate and fast to provide meaningful and accurate insights. The methods discussed in this analysis focused primarily on accuracy, further study should be implemented with a focus on computational performance before a final model is select.

# References

1. Nolan, D. and Lang, D. T. “Data Science in R.” CRC Press, 2015 (Chapter 1)
2. <http://rdatasciencecases.org/>
3. Ault, A., Zhong, X., and Coyle, E. J. (2005), [*K-Nearest-Neighbor Analysis of Received Signal Strength Distance Estimation Across Environments.*](https://s3-us-west-2.amazonaws.com/smu-mds/prod/Quantifying+the+World/Course+Materials/WiNMee_Ault.pdf) *Technical Report. Purdue University: Center for Wireless Systems and Applications,=.*
4. Madigan, D., Ju, W. H., Krishnan, P., Krishnakumar, A. S., and Zorych, I. (2006), ["Location Estimation in Wireless Networks: A Bayesian Approach."](https://s3-us-west-2.amazonaws.com/smu-mds/prod/Quantifying+the+World/Course+Materials/A16n210.pdf) *Statistica Sinica*, 16, 495-522.
5. Tarrio, P., Bernardos, A. M., and Casar, J. R. (2011), ["Weighted Least Squares Techniques for Improved Received Signal Strength Based Localization."](https://s3-us-west-2.amazonaws.com/smu-mds/prod/Quantifying+the+World/Course+Materials/sensors-11-08569.pdf) *Sensors*, 11(9), 8569-8592.
6. Sayad, Saed PhD. K Nearest Neighbors – Classification. <https://www.saedsayad.com/k_nearest_neighbors.htm>

# Code

|  |
| --- |
| Significant amounts of the code shown below was borrowed from the Nolan and Lang book (reference number 1). Most the data cleaning comes directly from the book. We repurposed the code to answer the questions we had about the data.  # Nolan, D. and Lang, D. T. “Data Science in R.” CRC Press, 2015 (Chapter 1)    # Reading in data and verify it is the same set as the book  # OfflinePath = 'C:/Users/casey/Dropbox/SMU\_DataScience/MSDS\_7333\_QuantifyingTheWorld/Homework/CaseStudy1/offline.final.trace.txt'  # OfflinePath = "[/home/kyle/Documents/thomaskh522@gmail.com/SMU/Quant](mailto:/home/kyle/Documents/thomaskh522@gmail.com/SMU/Quant) the World/RTLS\_CaseStudy/Data/offline\_data.txt"  OfflinePath = '/home/kyle\_thomas/Documents/For\_Others/ME/SMU/RTLS\_CaseStudy/Data/offline\_data.txt'    # onlinePath = 'C:/Users/casey/Dropbox/SMU\_DataScience/MSDS\_7333\_QuantifyingTheWorld/Homework/CaseStudy1/online.final.trace.txt'  # onlinePath = '[/home/kyle/Documents/thomaskh522@gmail.com/SMU/Quant](mailto:/home/kyle/Documents/thomaskh522@gmail.com/SMU/Quant) the World/RTLS\_CaseStudy/Data/online\_data.txt'  onlinePath = '/home/kyle\_thomas/Documents/For\_Others/ME/SMU/RTLS\_CaseStudy/Data/online\_data.txt'    readData =  function(filename,  subMacs = c(  "00:0f:a3:39:e1:c0",  "00:0f:a3:39:dd:cd",  "00:14:bf:b1:97:8a",  "00:14:bf:3b:c7:c6",  "00:14:bf:b1:97:90",  "00:14:bf:b1:97:8d",  "00:14:bf:b1:97:81"  ))  {  txt = readLines(filename)  lines = txt[substr(txt, 1, 1) != "#"]  tmp = lapply(lines, processLine)  offline = as.data.frame(do.call("rbind", tmp),  stringsAsFactors = FALSE)    names(offline) = c(  "time",  "scanMac",  "posX",  "posY",  "posZ",  "orientation",  "mac",  "signal",  "channel",  "type"  )    # keep only signals from access points  offline = offline[offline$type == "3",]  submac = names(sort(table(offline$mac), decreasing = TRUE))[1:7]    # drop scanMac, posZ, channel, and type - no info in them  dropVars = c("scanMac", "posZ", "channel", "type")  offline = offline[,!(names(offline) %in% dropVars)]      # drop more unwanted access points  offline = offline[offline$mac %in% subMacs,]    # convert numeric values  numVars = c("time", "posX", "posY", "orientation", "signal")  offline[numVars] = lapply(offline[numVars], as.numeric)    offline$posX = round(offline$posX, 0)  offline$posY = round(offline$posY, 0)    offline$posXY = paste(offline$posX, offline$posY, sep = "-")    # convert time to POSIX  offline$rawTime = offline$time  offline$time = offline$time / 1000  class(offline$time) = c("POSIXt", "POSIXct")    # round orientations to nearest 45  offline$angle = roundOrientation(offline$orientation)    return(offline)  }    processLine = function(x) {  tokens = strsplit(x, "[;=,]")[[1]]    if (length(tokens) == 10)  return(NULL)  tmp = matrix(tokens[-(1:10)], ncol = 4, byrow = TRUE)  cbind(matrix(  tokens[c(2, 4, 6:8, 10)],  nrow = nrow(tmp),  ncol = 6,  byrow = TRUE  ), tmp)  }    roundOrientation = function(angles) {  refs = seq(0, by = 45, length = 9)  q = sapply(angles, function(o)  which.min(abs(o - refs)))  c(refs[1:8], 0)[q]  }    offline = readData(OfflinePath)  # online = readData(OnlinePath1)    #pdf(file = "Geo\_BoxplotSignalByMacAngle.pdf", width = 7)  oldPar = par(mar = c(3.1, 3, 1, 1))    library(lattice)  bwplot(  signal ~ factor(angle) | mac,  data = offline,  subset = posX == 2 & posY == 12,  layout = c(2, 4)  )    par(oldPar)  dev.off()    # summary(offline$signal)    #pdf(file = "Geo\_DensitySignalByMacAngle.pdf", width = 8, height = 12)  oldPar = par(mar = c(3.1, 3, 1, 1))    densityplot(  ~ signal | mac + factor(angle),  data = offline,  subset = posX == 24 & posY == 4,  bw = 0.5,  plot.points = FALSE  )    par(oldPar)  dev.off()    #offline = offline[ offline$mac != "00:0f:a3:39:dd:cd", ]    offline$posXY = paste(offline$posX, offline$posY, sep = "-")    byLocAngleAP = with(offline,  by(offline, list(posXY, angle, mac),  function(x)  x))    signalSummary =  lapply(byLocAngleAP,  function(oneLoc) {  ans = oneLoc[1,]  ans$medSignal = median(oneLoc$signal)  ans$avgSignal = mean(oneLoc$signal)  ans$num = length(oneLoc$signal)  ans$sdSignal = sd(oneLoc$signal)  ans$iqrSignal = IQR(oneLoc$signal)  ans  })    offlineSummary = do.call("rbind", signalSummary)    #pdf(file = "Geo\_BoxplotSignalSDByAvg.pdf", width = 10)  # oldPar = par(mar = c(3.1, 3, 1, 1))    # breaks = seq(-90, -30, by = 5)  # bwplot(sdSignal ~ cut(avgSignal, breaks = breaks),  # data = offlineSummary,  # xlab = "Mean Signal", ylab = "SD Signal")    # par(oldPar)  # dev.off()    #pdf(file = "Geo\_ScatterMean-Median.pdf", width = 10)  oldPar = par(mar = c(4.1, 4.1, 1, 1))  submac = names(sort(table(offline$mac), decreasing = TRUE))[1:7]    with(  offlineSummary,  smoothScatter((avgSignal - medSignal) ~ num,  xlab = "Number of Observations",  ylab = "mean - median")  )  abline(h = 0, col = "#984ea3", lwd = 2)    lo.obj =  with(offlineSummary,  loess(diff ~ num,  data = data.frame(  diff = (avgSignal - medSignal),  num = num  )))    lo.obj.pr = predict(lo.obj, newdata = data.frame(num = (70:120)))  lines(  x = 70:120,  y = lo.obj.pr,  col = "#4daf4a",  lwd = 2  )    par(oldPar)  dev.off()    library(fields)  plotAllHeatMaps = function() {  for (thismac in submac) {  print(thismac)    oneAPAngle = subset(offlineSummary, mac == thismac &  angle == 0)    smoothSS = Tps(oneAPAngle[, c("posX", "posY")],  oneAPAngle$avgSignal)    vizSmooth = predictSurface(smoothSS)    plot.surface(vizSmooth, type = "C")    points(oneAPAngle$posX,  oneAPAngle$posY,  pch = 19,  cex = 0.5)    title(thismac)    }      }    plotAllHeatMaps()      # removed the angle from subset. This will combine all the angles    plotAllHeatMaps = function() {  for (thismac in submac) {  print(thismac)    oneAPAngle = subset(offlineSummary, mac == thismac)    smoothSS = Tps(oneAPAngle[, c("posX", "posY")],  oneAPAngle$avgSignal)    vizSmooth = predictSurface(smoothSS)    plot.surface(vizSmooth, type = "C")    points(oneAPAngle$posX,  oneAPAngle$posY,  pch = 19,  cex = 0.5)    title(thismac)    }      }    plotAllHeatMaps()      library(fields)    oneAPAngle = subset(offlineSummary, mac == submac[5] & angle == 0)    smoothSS = Tps(oneAPAngle[, c("posX", "posY")],  oneAPAngle$avgSignal)    vizSmooth = predictSurface(smoothSS)    plot.surface(vizSmooth, type = "C")    points(oneAPAngle$posX,  oneAPAngle$posY,  pch = 19,  cex = 0.5)    # overall signal strength  library(fields)    allAPAngle = subset(offlineSummary, angle == 0)    smoothSS = Tps(allAPAngle[, c("posX", "posY")],  allAPAngle$avgSignal)    vizSmooth = predictSurface(smoothSS)    plot.surface(vizSmooth, type = "C")    points(allAPAngle$posX,  allAPAngle$posY,  pch = 19,  cex = 0.5)        unique(offlineSummary$mac)    macc0 = submac[1]  maccd = submac[2]    surfaceSS = function(data, mac, angle = 45) {  require(fields)  oneAPAngle = data[data$mac == mac & data$angle == angle,]  smoothSS = Tps(oneAPAngle[, c("posX", "posY")],  oneAPAngle$avgSignal)  vizSmooth = predictSurface(smoothSS)  plot.surface(  vizSmooth,  type = "C",  xlab = "",  ylab = "",  xaxt = "n",  yaxt = "n"  )  points(oneAPAngle$posX,  oneAPAngle$posY,  pch = 19,  cex = 0.5)  title(paste(c(mac, angle), sep = " "))  }    parCur = par(mfrow = c(2, 2), mar = rep(1, 4))    mapply(  surfaceSS,  mac = submac[rep(c(1, 2, 3, 4, 5, 6, 7), each = 4)],  angle = rep(c(0, 135), 2),  data = list(data = offlineSummary)  )    # par(parCur)    # plots for all angles of 00:0f:a3:39:e1:c0  parCur = par(mfrow = c(2, 2), mar = rep(2, 4))    mapply(  surfaceSS,  mac = submac[rep(c(1, 2), each = 4)],  angle = rep(c(0, 90, 135, 270), 2),  data = list(data = offlineSummary)  )    # install.packages('dplyr')  library(dplyr)  RSSmax = offlineSummary %>%  select(mac, posX, posY, avgSignal) %>%  group\_by(mac, posX, posY) %>%  # summarise(max(avgSignal), mean(posX), mean(posY))  summarise(max = max(avgSignal))    macLocations = RSSmax %>%  group\_by(mac) %>%  arrange(max)    macLocations    submac[-2]    # # offlineSummary = subset(offlineSummary, mac != submac[2])    # AP\_orig = matrix( c( 7.5, 6.3, 2.5, -.8, 12.8, -2.8,  # 1, 14, 33.5, 9.3, 33.5, 2.8),  # ncol = 2, byrow = TRUE,  # dimnames = list(submac[ -2 ], c("x", "y") ))      AP\_orig = matrix(  c(7.5, 6.3, 7.5, 6.3, 2.5,-.8, 12.8,-2.8,  1, 14, 33.5, 9.3, 33.5, 2.8),  ncol = 2,  byrow = TRUE,  # dimnames = list(submac[ -2 ], c("x", "y") ))  dimnames = list(submac, c("x", "y"))  )      AP\_orig    # added the location of the missing mac    AP = matrix(  c(7.5, 6.3, 32, 6, 2.5,-.8, 12.8,-2.8,  1, 14, 33.5, 9.3, 33.5, 2.8),  ncol = 2,  byrow = TRUE,  dimnames = list(submac, c("x", "y"))  )    AP    # Plot same mac for all angles    surfaceSS = function(data, mac, angle = 45) {  require(fields)  oneAPAngle = data[data$mac == mac & data$angle == angle,]  smoothSS = Tps(oneAPAngle[, c("posX", "posY")],  oneAPAngle$avgSignal)  vizSmooth = predictSurface(smoothSS)  plot.surface(  vizSmooth,  type = "C",  xlab = "",  ylab = "",  xaxt = "n",  yaxt = "n"  )  points(oneAPAngle$posX,  oneAPAngle$posY,  pch = 19,  cex = 0.5)  title(angle)  }    parCur = par(mfrow = c(4, 2), mar = rep(1, 1, 4))    mapply(  surfaceSS,  mac = submac[1],  angle = c(0, 45, 90, 135, 180, 225, 270, 315),  data = list(data = offlineSummary)  )      diffs = offlineSummary[, c("posX", "posY")] - AP[offlineSummary$mac,]    offlineSummary$dist = sqrt(diffs[, 1] ^ 2 + diffs[, 2] ^ 2)    xyplot(  signal ~ dist | factor(mac) + factor(angle),  data = offlineSummary,  pch = 19,  cex = 0.3,  xlab = "distance"  )    #pdf(file="Geo\_ScatterSignalDist.pdf", width = 7, height = 10)  oldPar = par(mar = c(3.1, 3.1, 1, 1))  library(lattice)  xyplot(  signal ~ dist | factor(mac) + factor(angle),  data = offlineSummary,  pch = 19,  cex = 0.3,  xlab = "distance"  )  par(oldPar)  dev.off()    macs = unique(offlineSummary$mac)    names(sort(table(offline$mac), decreasing = TRUE))[1:7]      # onlinePath = 'C:/Users/casey/Dropbox/SMU\_DataScience/MSDS\_7333\_QuantifyingTheWorld/Homework/CaseStudy1/online.final.trace.txt'  # onlinePath = '[/home/kyle/Documents/thomaskh522@gmail.com/SMU/Quant](mailto:/home/kyle/Documents/thomaskh522@gmail.com/SMU/Quant) the World/RTLS\_CaseStudy/Data/online\_data.txt'  submac = names(sort(table(offline$mac), decreasing = TRUE))[1:7]    online = readData(onlinePath, subMacs = submac)    length(unique(online$posXY))    tabonlineXYA = table(online$posXY, online$angle)  tabonlineXYA[1:6,]    keepVars = c("posXY", "posX", "posY", "orientation", "angle")  byLoc = with(online,  by(online, list(posXY),  function(x) {  ans = x[1, keepVars]  avgSS = tapply(x$signal, x$mac, mean)  y = matrix(  avgSS,  nrow = 1,  ncol = 7,  #ncol was 6  dimnames = list(ans$posXY, names(avgSS))  )  cbind(ans, y)  }))    onlineSummary = do.call("rbind", byLoc)            dim(onlineSummary)    names(onlineSummary)  m = 3  angleNewObs = 230  refs = seq(0, by = 45, length = 8)  nearestAngle = roundOrientation(angleNewObs)    if (m %% 2 == 1) {  angles = seq(-45 \* (m - 1) / 2, 45 \* (m - 1) / 2, length = m)  } else {  m = m + 1  angles = seq(-45 \* (m - 1) / 2, 45 \* (m - 1) / 2, length = m)  if (sign(angleNewObs - nearestAngle) > -1)  angles = angles[-1]  else  angles = angles[-m]  }  angles = angles + nearestAngle  angles[angles < 0] = angles[angles < 0] + 360  angles[angles > 360] = angles[angles > 360] - 360    offlineSubset =  offlineSummary[offlineSummary$angle %in% angles,]    reshapeSS = function(data,  varSignal = "signal",  keepVars = c("posXY", "posX", "posY")) {  byLocation =  with(data, by(data, list(posXY),  function(x) {  ans = x[1, keepVars]  avgSS = tapply(x[, varSignal], x$mac, mean)  y = matrix(  avgSS,  nrow = 1,  ncol = 7,  #ncol was 6  dimnames = list(ans$posXY,  names(avgSS))  )  cbind(ans, y)  }))    newDataSS = do.call("rbind", byLocation)  return(newDataSS)  }    trainSS = reshapeSS(offlineSubset, varSignal = "avgSignal")    selectTrain = function(angleNewObs,  signals = NULL,  m = 1) {  # m is the number of angles to keep between 1 and 5  refs = seq(0, by = 45, length = 8)  nearestAngle = roundOrientation(angleNewObs)    if (m %% 2 == 1)  angles = seq(-45 \* (m - 1) / 2, 45 \* (m - 1) / 2, length = m)  else {  m = m + 1  angles = seq(-45 \* (m - 1) / 2, 45 \* (m - 1) / 2, length = m)  if (sign(angleNewObs - nearestAngle) > -1)  angles = angles[-1]  else  angles = angles[-m]  }  angles = angles + nearestAngle  angles[angles < 0] = angles[angles < 0] + 360  angles[angles > 360] = angles[angles > 360] - 360  angles = sort(angles)    offlineSubset = signals[signals$angle %in% angles,]  reshapeSS(offlineSubset, varSignal = "avgSignal")  }    head(trainSS)[, 4:10]    train130 = selectTrain(130, offlineSummary, m = 3)    head(train130)    length(train130[[1]])    findNN = function(newSignal, trainSubset) {  diffs = apply(trainSubset[, 4:10], 1, #changed to 4:10 to get all MACs  function(x)  x - newSignal)  dists = apply(diffs, 2, function(x)  sqrt(sum(x ^ 2)))  closest = order(dists)  return(trainSubset[closest, 1:3])  }        onlineSummary[, 6:12]    predXY = function(newSignals,  newAngles,  trainData,  numAngles = 1,  k = 3) {  closeXY = list(length = nrow(newSignals))    for (i in 1:nrow(newSignals)) {  trainSS = selectTrain(newAngles[i], trainData, m = numAngles)  closeXY[[i]] =  findNN(newSignal = as.numeric(newSignals[i,]), trainSS)  }    estXY = lapply(closeXY,  function(x)  sapply(x[, 2:3],  function(x)  mean(x[1:k])))  estXY = do.call("rbind", estXY)  return(estXY)  }  estXYk3 = predXY(  newSignals = onlineSummary[, 6:12],  #changed to 6:12  newAngles = onlineSummary[, 4],  offlineSummary,  numAngles = 3,  k = 3  )    estXYk1 = predXY(  newSignals = onlineSummary[, 6:12],  #changed to 6:12  newAngles = onlineSummary[, 4],  offlineSummary,  numAngles = 3,  k = 1  )        floorErrorMap = function(estXY,  actualXY,  trainPoints = NULL,  AP = NULL) {  plot(  0,  0,  xlim = c(0, 35),  ylim = c(-3, 15),  type = "n",  xlab = "",  ylab = "",  axes = FALSE  )  box()  if (!is.null(AP))  points(AP, pch = 15)  if (!is.null(trainPoints))  points(trainPoints,  pch = 19,  col = "grey",  cex = 0.6)    points(  x = actualXY[, 1],  y = actualXY[, 2],  pch = 19,  cex = 0.8  )  points(  x = estXY[, 1],  y = estXY[, 2],  pch = 8,  cex = 0.8  )  segments(  x0 = estXY[, 1],  y0 = estXY[, 2],  x1 = actualXY[, 1],  y1 = actualXY[, 2],  lwd = 2,  col = "red"  )  }    trainPoints = offlineSummary[offlineSummary$angle == 0 &  offlineSummary$mac == "00:0f:a3:39:e1:c0" ,  c("posX", "posY")]    #pdf(file="GEO\_FloorPlanK3Errors.pdf", width = 10, height = 7)  oldPar = par(mar = c(1, 1, 1, 1))  floorErrorMap(estXYk3,  onlineSummary[, c("posX", "posY")],  trainPoints = trainPoints,  AP = AP)  par(oldPar)  dev.off()    #pdf(file="GEO\_FloorPlanK1Errors.pdf", width = 10, height = 7)  oldPar = par(mar = c(1, 1, 1, 1))  floorErrorMap(estXYk1,  onlineSummary[, c("posX", "posY")],  trainPoints = trainPoints,  AP = AP)  par(oldPar)  dev.off()    calcError =  function(estXY, actualXY)  sum(rowSums((estXY - actualXY) ^ 2))    actualXY = onlineSummary[, c("posX", "posY")]  sapply(list(estXYk1, estXYk3), calcError, actualXY)    calcError =  function(estXY, actualXY)  sum(rowSums((estXY - actualXY) ^ 2))    actualXY = onlineSummary[, c("posX", "posY")]  sapply(list(estXYk1, estXYk3), calcError, actualXY)    v = 11  permuteLocs = sample(unique(offlineSummary$posXY))  permuteLocs = matrix(permuteLocs, ncol = v,  nrow = floor(length(permuteLocs) / v))    onlineFold = subset(offlineSummary, posXY %in% permuteLocs[, 1])    reshapeSS = function(data,  varSignal = "signal",  keepVars = c("posXY", "posX", "posY"),  sampleAngle = FALSE,  refs = seq(0, 315, by = 45)) {  byLocation =  with(data, by(data, list(posXY),  function(x) {  if (sampleAngle) {  x = x[x$angle == sample(refs, size = 1),]  }  ans = x[1, keepVars]  avgSS = tapply(x[, varSignal], x$mac, mean)  y = matrix(  avgSS,  nrow = 1,  ncol = 7,  #ncol was 6  dimnames = list(ans$posXY,  names(avgSS))  )  cbind(ans, y)  }))    newDataSS = do.call("rbind", byLocation)  return(newDataSS)  }    # offline = offline[ offline$mac != "00:0f:a3:39:dd:cd", ]    keepVars = c("posXY", "posX", "posY", "orientation", "angle")    onlineCVSummary = reshapeSS(offline, keepVars = keepVars,  sampleAngle = TRUE)    onlineFold = subset(onlineCVSummary,  posXY %in% permuteLocs[, 1])    offlineFold = subset(offlineSummary,  posXY %in% permuteLocs[,-1])    estFold = predXY(  newSignals = onlineFold[, 6:12],  #changed to 6:12 to get all MACS  newAngles = onlineFold[, 4],  offlineFold,  numAngles = 3,  k = 3  )    onlineFold[, 6:12]    permuteLocs    actualFold = onlineFold[, c("posX", "posY")]  calcError(estFold, actualFold)    K = 20  err = rep(0, K)    for (j in 1:v) {  onlineFold = subset(onlineCVSummary,  posXY %in% permuteLocs[, j])  offlineFold = subset(offlineSummary,  posXY %in% permuteLocs[,-j])  actualFold = onlineFold[, c("posX", "posY")]    for (k in 1:K) {  estFold = predXY(  newSignals = onlineFold[, 6:12],  #changed to 6:12 to get all MACs  newAngles = onlineFold[, 4],  offlineFold,  numAngles = 3,  k = k  )  err[k] = err[k] + calcError(estFold, actualFold)  }  }    err    #pdf(file = "Geo\_CVChoiceOfK.pdf", width = 10, height = 6)  #oldPar = par(mar = c(4, 3, 1, 1))  plot(  y = err,  x = (1:K),  type = "l",  lwd = 2,  ylim = c(100, 2100),  xlab = "Number of Neighbors",  ylab = "Sum of Square Errors"  )    rmseMin = min(err)  kMin = which(err == rmseMin)[1]  segments(  x0 = 0,  x1 = kMin,  y0 = rmseMin,  col = gray(0.4),  lty = 2,  lwd = 2  )  segments(  x0 = kMin,  x1 = kMin,  y0 = 1100,  y1 = rmseMin,  col = grey(0.4),  lty = 2,  lwd = 2  )  text(  x = kMin - 2,  y = rmseMin + 40,  label = as.character(round(rmseMin)),  col = grey(0.4)  )  # par(oldPar)  # dev.off()    #mtext(kMin, side = 1, line = 1, at = kMin, col = grey(0.4))      estXYk6 = predXY(  newSignals = onlineSummary[, 6:11],  newAngles = onlineSummary[, 4],  offlineSummary,  numAngles = 3,  k = 6  )    calcError(estXYk6, actualXY)    predXY = function(newSignals,  newAngles,  trainData,  numAngles = 1,  k = 3) {  closeXY = list(length = nrow(newSignals))    for (i in 1:nrow(newSignals)) {  trainSS = selectTrain(newAngles[i], trainData, m = numAngles)  closeXY[[i]] = findNN(newSignal = as.numeric(newSignals[i,]),  trainSS)  }    estXY = lapply(closeXY, function(x)  sapply(x[, 2:3],  function(x)  mean(x[1:k])))  estXY = do.call("rbind", estXY)  return(estXY)  }    findWNN = function(newSignal, trainSubset) {  diffs = apply(trainSubset[, 4:9], 1,  function(x)  abs(x - newSignal))  dists = apply(diffs, 2, function(x)  (1 / x) / sum(1 / x))  closest = order(dists)  return(trainSubset[closest, 1:3])  }    findNN = function(newSignal, trainSubset) {  diffs = apply(trainSubset[, 4:9], 1,  function(x)  x - newSignal)  dists = apply(diffs, 2, function(x)  sqrt(sum(x ^ 2)))  trainSubset$dist = dists  closest = order(dists)  return(trainSubset[closest, 1:4])  }    offlineSummaryC0 = offlineSummary[offlineSummary$mac != '00:0f:a3:39:dd:cd', ]  dim(offlineSummaryC0)    online = readData(onlinePath, subMacs = submac[-2])    length(unique(online$posXY))    tabonlineXYA = table(online$posXY, online$angle)  tabonlineXYA[1:6,]    keepVars = c("posXY", "posX", "posY", "orientation", "angle")  byLoc = with(online,  by(online, list(posXY),  function(x) {  ans = x[1, keepVars]  avgSS = tapply(x$signal, x$mac, mean)  y = matrix(  avgSS,  nrow = 1,  ncol = 6,  #ncol was 6  dimnames = list(ans$posXY, names(avgSS))  )  cbind(ans, y)  }))    onlineSummary = do.call("rbind", byLoc)        dim(onlineSummary)    names(onlineSummary)  m = 3  angleNewObs = 230  refs = seq(0, by = 45, length = 8)  nearestAngle = roundOrientation(angleNewObs)    if (m %% 2 == 1) {  angles = seq(-45 \* (m - 1) / 2, 45 \* (m - 1) / 2, length = m)  } else {  m = m + 1  angles = seq(-45 \* (m - 1) / 2, 45 \* (m - 1) / 2, length = m)  if (sign(angleNewObs - nearestAngle) > -1)  angles = angles[-1]  else  angles = angles[-m]  }  angles = angles + nearestAngle  angles[angles < 0] = angles[angles < 0] + 360  angles[angles > 360] = angles[angles > 360] - 360    offlineSubsetC0 =  offlineSummaryC0[offlineSummaryC0$angle %in% angles,]    reshapeSS = function(data,  varSignal = "signal",  keepVars = c("posXY", "posX", "posY")) {  byLocation =  with(data, by(data, list(posXY),  function(x) {  ans = x[1, keepVars]  avgSS = tapply(x[, varSignal], x$mac, mean)  y = matrix(  avgSS,  nrow = 1,  ncol = 6,  #ncol was 6  dimnames = list(ans$posXY,  names(avgSS))  )  cbind(ans, y)  }))    newDataSS = do.call("rbind", byLocation)  return(newDataSS)  }    trainSS = reshapeSS(offlineSubsetC0, varSignal = "avgSignal")    selectTrain = function(angleNewObs,  signals = NULL,  m = 1) {  # m is the number of angles to keep between 1 and 5  refs = seq(0, by = 45, length = 8)  nearestAngle = roundOrientation(angleNewObs)    if (m %% 2 == 1)  angles = seq(-45 \* (m - 1) / 2, 45 \* (m - 1) / 2, length = m)  else {  m = m + 1  angles = seq(-45 \* (m - 1) / 2, 45 \* (m - 1) / 2, length = m)  if (sign(angleNewObs - nearestAngle) > -1)  angles = angles[-1]  else  angles = angles[-m]  }  angles = angles + nearestAngle  angles[angles < 0] = angles[angles < 0] + 360  angles[angles > 360] = angles[angles > 360] - 360  angles = sort(angles)    offlineSubsetC0 = signals[signals$angle %in% angles,]  reshapeSS(offlineSubsetC0, varSignal = "avgSignal")  }    train130 = selectTrain(130, offlineSummaryC0, m = 3)    head(train130)    length(train130[[1]])    findNN = function(newSignal, trainSubset) {  diffs = apply(trainSubset[, 4:9], 1,  function(x)  x - newSignal)  dists = apply(diffs, 2, function(x)  sqrt(sum(x ^ 2)))  closest = order(dists)  return(trainSubset[closest, 1:3])  }        predXY = function(newSignals,  newAngles,  trainData,  numAngles = 1,  k = 3) {  closeXY = list(length = nrow(newSignals))    for (i in 1:nrow(newSignals)) {  trainSS = selectTrain(newAngles[i], trainData, m = numAngles)  closeXY[[i]] =  findNN(newSignal = as.numeric(newSignals[i,]), trainSS)  }    estXY = lapply(closeXY,  function(x)  sapply(x[, 2:3],  function(x)  mean(x[1:k])))  estXY = do.call("rbind", estXY)  return(estXY)  }  estXYk3 = predXY(  newSignals = onlineSummary[, 6:11],  newAngles = onlineSummary[, 4],  offlineSummaryC0,  numAngles = 3,  k = 3  )    estXYk1 = predXY(  newSignals = onlineSummary[, 6:11],  newAngles = onlineSummary[, 4],  offlineSummaryC0,  numAngles = 3,  k = 1  )        floorErrorMap = function(estXY,  actualXY,  trainPoints = NULL,  AP = NULL) {  plot(  0,  0,  xlim = c(0, 35),  ylim = c(-3, 15),  type = "n",  xlab = "",  ylab = "",  axes = FALSE  )  box()  if (!is.null(AP))  points(AP, pch = 15)  if (!is.null(trainPoints))  points(trainPoints,  pch = 19,  col = "grey",  cex = 0.6)    points(  x = actualXY[, 1],  y = actualXY[, 2],  pch = 19,  cex = 0.8  )  points(  x = estXY[, 1],  y = estXY[, 2],  pch = 8,  cex = 0.8  )  segments(  x0 = estXY[, 1],  y0 = estXY[, 2],  x1 = actualXY[, 1],  y1 = actualXY[, 2],  lwd = 2,  col = "red"  )  }    trainPoints = offlineSummaryC0[offlineSummaryC0$angle == 0 &  offlineSummaryC0$mac == "00:0f:a3:39:e1:c0" ,  c("posX", "posY")]    #pdf(file="GEO\_FloorPlanK3Errors.pdf", width = 10, height = 7)  oldPar = par(mar = c(1, 1, 1, 1))  floorErrorMap(estXYk3,  onlineSummary[, c("posX", "posY")],  trainPoints = trainPoints,  AP = AP)  par(oldPar)  dev.off()    #pdf(file="GEO\_FloorPlanK1Errors.pdf", width = 10, height = 7)  oldPar = par(mar = c(1, 1, 1, 1))  floorErrorMap(estXYk1,  onlineSummary[, c("posX", "posY")],  trainPoints = trainPoints,  AP = AP)  par(oldPar)  dev.off()    calcError =  function(estXY, actualXY)  sum(rowSums((estXY - actualXY) ^ 2))    actualXY = onlineSummary[, c("posX", "posY")]  sapply(list(estXYk1, estXYk3), calcError, actualXY)    calcError =  function(estXY, actualXY)  sum(rowSums((estXY - actualXY) ^ 2))    actualXY = onlineSummary[, c("posX", "posY")]  sapply(list(estXYk1, estXYk3), calcError, actualXY)    v = 11  permuteLocs = sample(unique(offlineSummaryC0$posXY))  permuteLocs = matrix(permuteLocs, ncol = v,  nrow = floor(length(permuteLocs) / v))    onlineFold = subset(offlineSummaryC0, posXY %in% permuteLocs[, 1])    reshapeSS = function(data,  varSignal = "signal",  keepVars = c("posXY", "posX", "posY"),  sampleAngle = FALSE,  refs = seq(0, 315, by = 45)) {  byLocation =  with(data, by(data, list(posXY),  function(x) {  if (sampleAngle) {  x = x[x$angle == sample(refs, size = 1),]  }  ans = x[1, keepVars]  avgSS = tapply(x[, varSignal], x$mac, mean)  y = matrix(  avgSS,  nrow = 1,  ncol = 6,  #ncol was 6  dimnames = list(ans$posXY,  names(avgSS))  )  cbind(ans, y)  }))    newDataSS = do.call("rbind", byLocation)  return(newDataSS)  }    offlineC0 = offline[offline$mac != "00:0f:a3:39:dd:cd",]    keepVars = c("posXY", "posX", "posY", "orientation", "angle")    onlineCVSummary = reshapeSS(offlineC0, keepVars = keepVars, #this has offline data, shouldn't if be online?  sampleAngle = TRUE)    onlineFold = subset(onlineCVSummary,  posXY %in% permuteLocs[, 1])    offlineFold = subset(offlineSummaryC0,  posXY %in% permuteLocs[,-1])    estFold = predXY(  newSignals = onlineFold[, 6:11],  newAngles = onlineFold[, 4],  offlineFold,  numAngles = 3,  k = 3  )    onlineCVSummary    permuteLocs    actualFold = onlineFold[, c("posX", "posY")]  calcError(estFold, actualFold)    K = 20  err = rep(0, K)    for (j in 1:v) {  onlineFold = subset(onlineCVSummary,  posXY %in% permuteLocs[, j])  offlineFold = subset(offlineSummaryC0,  posXY %in% permuteLocs[,-j])  actualFold = onlineFold[, c("posX", "posY")]    for (k in 1:K) {  estFold = predXY(  newSignals = onlineFold[, 6:11],  newAngles = onlineFold[, 4],  offlineFold,  numAngles = 3,  k = k  )  err[k] = err[k] + calcError(estFold, actualFold)  }  }    err    #pdf(file = "Geo\_CVChoiceOfK.pdf", width = 10, height = 6)  #oldPar = par(mar = c(4, 3, 1, 1))  plot(  y = err,  x = (1:K),  type = "l",  lwd = 2,  ylim = c(100, 2100),  xlab = "Number of Neighbors",  ylab = "Sum of Square Errors"  )    rmseMin = min(err)  kMin = which(err == rmseMin)[1]  segments(  x0 = 0,  x1 = kMin,  y0 = rmseMin,  col = gray(0.4),  lty = 2,  lwd = 2  )  segments(  x0 = kMin,  x1 = kMin,  y0 = 1100,  y1 = rmseMin,  col = grey(0.4),  lty = 2,  lwd = 2  )  text(  x = kMin - 2,  y = rmseMin + 40,  label = as.character(round(rmseMin)),  col = grey(0.4)  )  # par(oldPar)  # dev.off()    offlineSummaryCD = offlineSummary[offlineSummary$mac != '00:0f:a3:39:e1:c0', ]  dim(offlineSummaryCD)  # unique(offlineSummary$mac)    submac[1]    # onlinePath1 = 'C:/Users/casey/Dropbox/SMU\_DataScience/MSDS\_7333\_QuantifyingTheWorld/Homework/CaseStudy1/online.final.trace.txt'  # onlinePath2 = '[/home/kyle/Documents/thomaskh522@gmail.com/SMU/Quant](mailto:/home/kyle/Documents/thomaskh522@gmail.com/SMU/Quant) the World/RTLS\_CaseStudy/Data/online\_data.txt'    online = readData(onlinePath, subMacs = submac[-1])    length(unique(online$posXY))    tabonlineXYA = table(online$posXY, online$angle)  tabonlineXYA[1:6,]    keepVars = c("posXY", "posX", "posY", "orientation", "angle")  byLoc = with(online,  by(online, list(posXY),  function(x) {  ans = x[1, keepVars]  avgSS = tapply(x$signal, x$mac, mean)  y = matrix(  avgSS,  nrow = 1,  ncol = 6,  #ncol was 6  dimnames = list(ans$posXY, names(avgSS))  )  cbind(ans, y)  }))    onlineSummary = do.call("rbind", byLoc)        dim(onlineSummary)    names(onlineSummary)  m = 3  angleNewObs = 230  refs = seq(0, by = 45, length = 8)  nearestAngle = roundOrientation(angleNewObs)    if (m %% 2 == 1) {  angles = seq(-45 \* (m - 1) / 2, 45 \* (m - 1) / 2, length = m)  } else {  m = m + 1  angles = seq(-45 \* (m - 1) / 2, 45 \* (m - 1) / 2, length = m)  if (sign(angleNewObs - nearestAngle) > -1)  angles = angles[-1]  else  angles = angles[-m]  }  angles = angles + nearestAngle  angles[angles < 0] = angles[angles < 0] + 360  angles[angles > 360] = angles[angles > 360] - 360    offlineSubsetCD =  offlineSummaryCD[offlineSummaryCD$angle %in% angles,]    reshapeSS = function(data,  varSignal = "signal",  keepVars = c("posXY", "posX", "posY")) {  byLocation =  with(data, by(data, list(posXY),  function(x) {  ans = x[1, keepVars]  avgSS = tapply(x[, varSignal], x$mac, mean)  y = matrix(  avgSS,  nrow = 1,  ncol = 6,  #ncol was 6  dimnames = list(ans$posXY,  names(avgSS))  )  cbind(ans, y)  }))    newDataSS = do.call("rbind", byLocation)  return(newDataSS)  }    trainSS = reshapeSS(offlineSubsetCD, varSignal = "avgSignal")    selectTrain = function(angleNewObs,  signals = NULL,  m = 1) {  # m is the number of angles to keep between 1 and 5  refs = seq(0, by = 45, length = 8)  nearestAngle = roundOrientation(angleNewObs)    if (m %% 2 == 1)  angles = seq(-45 \* (m - 1) / 2, 45 \* (m - 1) / 2, length = m)  else {  m = m + 1  angles = seq(-45 \* (m - 1) / 2, 45 \* (m - 1) / 2, length = m)  if (sign(angleNewObs - nearestAngle) > -1)  angles = angles[-1]  else  angles = angles[-m]  }  angles = angles + nearestAngle  angles[angles < 0] = angles[angles < 0] + 360  angles[angles > 360] = angles[angles > 360] - 360  angles = sort(angles)    offlineSubsetCD = signals[signals$angle %in% angles,]  reshapeSS(offlineSubsetCD, varSignal = "avgSignal")  }    train130 = selectTrain(130, offlineSummaryCD, m = 3)    head(train130)    length(train130[[1]])    findNN = function(newSignal, trainSubset) {  diffs = apply(trainSubset[, 4:9], 1,  function(x)  x - newSignal)  dists = apply(diffs, 2, function(x)  sqrt(sum(x ^ 2)))  closest = order(dists)  return(trainSubset[closest, 1:3])  }        predXY = function(newSignals,  newAngles,  trainData,  numAngles = 1,  k = 3) {  closeXY = list(length = nrow(newSignals))    for (i in 1:nrow(newSignals)) {  trainSS = selectTrain(newAngles[i], trainData, m = numAngles)  closeXY[[i]] =  findNN(newSignal = as.numeric(newSignals[i,]), trainSS)  }    estXY = lapply(closeXY,  function(x)  sapply(x[, 2:3],  function(x)  mean(x[1:k])))  estXY = do.call("rbind", estXY)  return(estXY)  }  estXYk3 = predXY(  newSignals = onlineSummary[, 6:11],  newAngles = onlineSummary[, 4],  offlineSummaryCD,  numAngles = 3,  k = 3  )    estXYk1 = predXY(  newSignals = onlineSummary[, 6:11],  newAngles = onlineSummary[, 4],  offlineSummaryCD,  numAngles = 3,  k = 1  )        floorErrorMap = function(estXY,  actualXY,  trainPoints = NULL,  AP = NULL) {  plot(  0,  0,  xlim = c(0, 35),  ylim = c(-3, 15),  type = "n",  xlab = "",  ylab = "",  axes = FALSE  )  box()  if (!is.null(AP))  points(AP, pch = 15)  if (!is.null(trainPoints))  points(trainPoints,  pch = 19,  col = "grey",  cex = 0.6)    points(  x = actualXY[, 1],  y = actualXY[, 2],  pch = 19,  cex = 0.8  )  points(  x = estXY[, 1],  y = estXY[, 2],  pch = 8,  cex = 0.8  )  segments(  x0 = estXY[, 1],  y0 = estXY[, 2],  x1 = actualXY[, 1],  y1 = actualXY[, 2],  lwd = 2,  col = "red"  )  }    trainPoints = offlineSummaryCD[offlineSummaryCD$angle == 0 &  offlineSummaryCD$mac == "00:0f:a3:39:e1:c0" ,  c("posX", "posY")]    #pdf(file="GEO\_FloorPlanK3Errors.pdf", width = 10, height = 7)  oldPar = par(mar = c(1, 1, 1, 1))  floorErrorMap(estXYk3,  onlineSummary[, c("posX", "posY")],  trainPoints = trainPoints,  AP = AP)  par(oldPar)  dev.off()    #pdf(file="GEO\_FloorPlanK1Errors.pdf", width = 10, height = 7)  oldPar = par(mar = c(1, 1, 1, 1))  floorErrorMap(estXYk1,  onlineSummary[, c("posX", "posY")],  trainPoints = trainPoints,  AP = AP)  par(oldPar)  dev.off()    calcError =  function(estXY, actualXY)  sum(rowSums((estXY - actualXY) ^ 2))    actualXY = onlineSummary[, c("posX", "posY")]  sapply(list(estXYk1, estXYk3), calcError, actualXY)    calcError =  function(estXY, actualXY)  sum(rowSums((estXY - actualXY) ^ 2))    actualXY = onlineSummary[, c("posX", "posY")]  sapply(list(estXYk1, estXYk3), calcError, actualXY)    v = 11  permuteLocs = sample(unique(offlineSummaryCD$posXY))  permuteLocs = matrix(permuteLocs, ncol = v,  nrow = floor(length(permuteLocs) / v))    onlineFold = subset(offlineSummaryCD, posXY %in% permuteLocs[, 1])    reshapeSS = function(data,  varSignal = "signal",  keepVars = c("posXY", "posX", "posY"),  sampleAngle = FALSE,  refs = seq(0, 315, by = 45)) {  byLocation =  with(data, by(data, list(posXY),  function(x) {  if (sampleAngle) {  x = x[x$angle == sample(refs, size = 1),]  }  ans = x[1, keepVars]  avgSS = tapply(x[, varSignal], x$mac, mean)  y = matrix(  avgSS,  nrow = 1,  ncol = 6,  #ncol was 6  dimnames = list(ans$posXY,  names(avgSS))  )  cbind(ans, y)  }))    newDataSS = do.call("rbind", byLocation)  return(newDataSS)  }    offlineCD = offline[offline$mac != "00:0f:a3:39:dd:cd",]    keepVars = c("posXY", "posX", "posY", "orientation", "angle")    onlineCVSummary = reshapeSS(offlineCD, keepVars = keepVars, #this has offline data, shouldn't if be online?  sampleAngle = TRUE)    onlineFold = subset(onlineCVSummary,  posXY %in% permuteLocs[, 1])    offlineFold = subset(offlineSummaryCD,  posXY %in% permuteLocs[,-1])    estFold = predXY(  newSignals = onlineFold[, 6:11],  newAngles = onlineFold[, 4],  offlineFold,  numAngles = 3,  k = 3  )    onlineCVSummary    permuteLocs    actualFold = onlineFold[, c("posX", "posY")]  calcError(estFold, actualFold)    K = 20  err = rep(0, K)    for (j in 1:v) {  onlineFold = subset(onlineCVSummary,  posXY %in% permuteLocs[, j])  offlineFold = subset(offlineSummaryCD,  posXY %in% permuteLocs[,-j])  actualFold = onlineFold[, c("posX", "posY")]    for (k in 1:K) {  estFold = predXY(  newSignals = onlineFold[, 6:11],  newAngles = onlineFold[, 4],  offlineFold,  numAngles = 3,  k = k  )  err[k] = err[k] + calcError(estFold, actualFold)  }  }    err    # pdf(file = "Geo\_CVChoiceOfK.pdf", width = 10, height = 6)  # oldPar = par(mar = c(4, 3, 1, 1))  plot(  y = err,  x = (1:K),  type = "l",  lwd = 2,  # ylim = c(100, 2100),  xlab = "Number of Neighbors",  ylab = "Sum of Square Errors"  )    rmseMin = min(err)  kMin = which(err == rmseMin)[1]  segments(  x0 = 0,  x1 = kMin,  y0 = rmseMin,  col = gray(0.4),  lty = 2,  lwd = 2  )  segments(  x0 = kMin,  x1 = kMin,  y0 = 1100,  y1 = rmseMin,  col = grey(0.4),  lty = 2,  lwd = 2  )  text(  x = kMin - 2,  y = rmseMin + 40,  label = as.character(round(rmseMin)),  col = grey(0.4)  )  # par(oldPar)  # dev.off()  ## Weighted mean in predXY  Ok. I dont know why the code works on its own but not in the cross validation. I will give up now. I will paste this code in the appendix and finish the write up.  '''predXY2 = function(newSignals, newAngles, trainData,  numAngles = 1, k = 3){  closeXY = list(length = nrow(newSignals))  for (i in 1:nrow(newSignals)) {  trainSS = selectTrain(newAngles[i], trainData, m = numAngles)  #from findNN  diffs = apply(trainSS[ , 4:9], 1, #subtract the new signal from all mac signals  function(x) abs(x - as.numeric(newSignals[i, ])))  dists = apply(diffs, 2, function(x) sqrt(sum(x^2)) ) #add up all differences  closest = order(dists)  weights = (closest/length(closest))  closeXY[[i]] = cbind(trainSS[closest, 1:3 ],weights)  # closeXY[[i]] = trainSS[closest, 1:3 ]  }  estXY = lapply(closeXY,  function(x) sapply(x[ , 2:3],  function(x) weighted.mean(x[1:k],closeXY[[1]][1:k,4])))  estXY = do.call("rbind", estXY)  results = list('diffs'=diffs,  'dists'=dists,  'closeXY'=closeXY,  'estXY'=estXY  )  return(results)  }  ################  estXYk3 = predXY2(newSignals = onlineSummary[ , 6:11],  newAngles = onlineSummary[ , 4],  offlineSummary, numAngles = 3, k = 3)  estXYk1 = predXY2(newSignals = onlineSummary[ , 6:11],  newAngles = onlineSummary[ , 4],  offlineSummary, numAngles = 3, k = 1)  estXYk3 = estXYk3['estXY'][[1]]  # estXYk3  estXYk1 = estXYk1['estXY'][[1]]''' |