2018

QTW unit 6 Case Study

PREDICTING LOCATION VIA INDOOR POSITIONING SYSTEMS

JOSHUA HERRERA, MATTHEW PAYNE, REMY LAGROIS

Abstract

Real time location systems (RTLS) are an integral part of our lives. Technologies like smart locks on doors which automatically unlock based on proximity and vacuum cleaners that can map a room for cleaning rely on RTLS. In this study, signal data for an RTLS is examined to implement a model that can accurately predict location. 166 points have been mapped to the floor of a building with 6 wireless access points (routers). A researcher walked around systematically and recorded their location in the building. Based on signal strength to the 6 routers we estimated the researcher's location. Because there are more than 6 MAC addresses in our data, our first task was to determine which MAC was more accurate in estimating location, and it was found that using both MAC addresses was the most accurate method. The predictions were done using KNN, and our second task was to implement a weighted KNN strategy which was compared to normal KNN.

Introduction

In this case study, we examine how to predict the location of electronic devices using wireless access points (routers) with known locations and known signal strength between the devices and access points. Wireless networks and hand-held devices that connect to those networks are ubiquitous in today’s society. To be able to predict the location of a hand-held device based on the wireless signal strength, it is necessary to model a dataset of known locations and then apply a model to the test set. For this case study, we used a reference dataset of over one million measurements of signal strength recorded by researchers at the University of Mannheim.

This reference dataset was created by having a handheld device connect to six access points in the test environment. Test measurements were made throughout the environment by measuring the signal strength of each access point from a fixed position in the room. There are 166 locations and at each, 110 readings were taken from each of 7 access point for each of 8 45-degree orientations; resulting in over a million lines. Using the signal strength and angle of orientation data, we will develop a model to predict the location of an electronic device in the test environment.

Methods

Data Processing:

The data needed to be cleaned and formatted before it could be analyzed. The data was initially in a ragged array that did not lend itself to data analysis. We followed the steps outlined in the text [1] to clean the data and reformat it into data frames for analysis. One obstacle to using the data for analysis was that all the variables were in one long string that needed to be tokenized and organized into individual variables of the correct type. The variables that resulted from the cleaning were: time, scanMAC, posX, posY, posZ, orientation, MAC, signal, channel, type. All the values for posZ were 0 because the same floor was used for all the measurements. ScanMAC only contains one value. ScanMAC and posZ were dropped from our further analysis. Channel was not useful for our analysis, and we only want type "3", which are access points, so after subsetting for access points, these variables were dropped. The data was given in milliseconds since 1970, wand was converted to seconds and formatted with the POSIXt and POSIXct formats. Next, our roundOrientation function was run which takes the orientation variable and rounds the value to the nearest 45-degree orientation. Lastly, a new variable is created, which is a concatenation of the X and Y positions, separated by a hyphen. This posXY variable is essential for our analysis. All this processing is wrapped into one function called readData.

Data Exploration:

Now that the data is read into a data frame and well formatted, we begin exploration by examining the shape and distribution of the data. An image of the floor-plan can be found in Figure 1, below. The 166 grey dots are the training data points, the black dots are the random locations at which the experimenter stood for readings, which we will be estimating later on, and the black boxes are the access points.

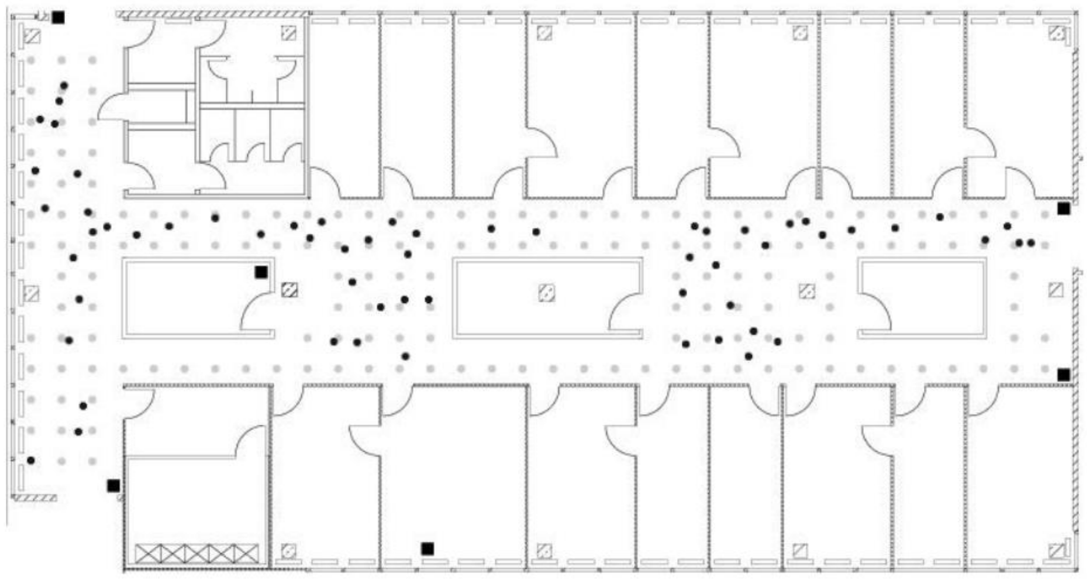


Figure 1

A boxplot function is run using lattice, which can be seen in Figure 2, below. This boxplot shows signal strength for one location, organized by MAC address, for every 45-degree orientation.

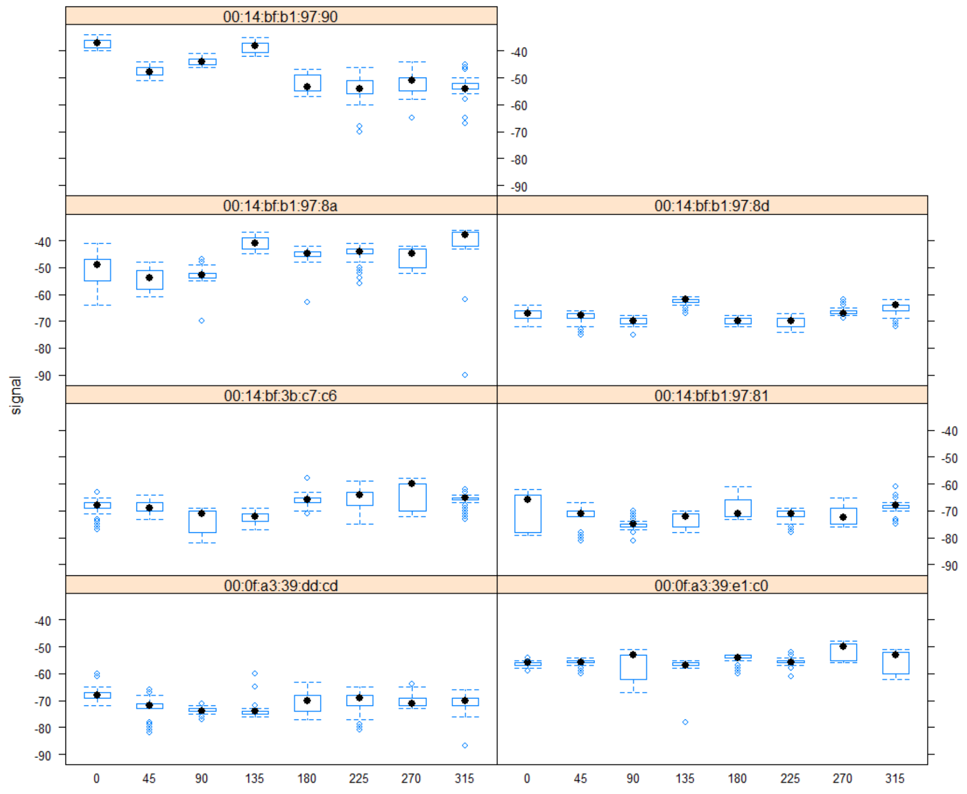


Figure 2

As shown in Figure 2, the orientation of the experimenter affects the signal strength, we will keep this in mind for future reference. The signal strengths are negative numbers, in which numbers closer to 0 are stronger. Based on this fact, we can assume that the location at which these readings were taken are very close to access points with MAC addresses 00:14:bf:b1:97:8a and 00:14:bf:b1:97:90. The location used in the boxplot is the top-left most location. If we refer to the floor-map shown in Figure 1, it would be wise to assume that the access point with MAC address ending in 90 is the northern-most router. We can infer this because at orientation 0 (north), this point has the strongest signal, and at orientation 180 (south), the reader shows the weakest signal. It would also make sense that the MAC address ending in 9a referred to the south-western-most access point. From our first location in the top left, MAC 8a shows strong signal when pointing south and weak signal when pointing north.

Instead of mapping all locations with a multitude of boxplots by hand, we will use the fields package to create a heatmap. Most importantly, we want to determine which access point has the two MAC addresses within it, as there are 7 MAC addresses and 6 access points. The offline data which contains the 1 million rows is shrunk into offlineSummary, in which the 110 readings for each orientation are summarized and placed into new variables, such as avgSignal, and medSignal. Now that there is one entry for every orientation, 8 entries for one location, 66 locations, and 7 MAC addresses which are being compared, we have 9,296 lines in our data frame. This summarized data can finally be turned into heatmaps using our surfaceSS function. Figure 3, below, shows a very interesting set of heat-maps.

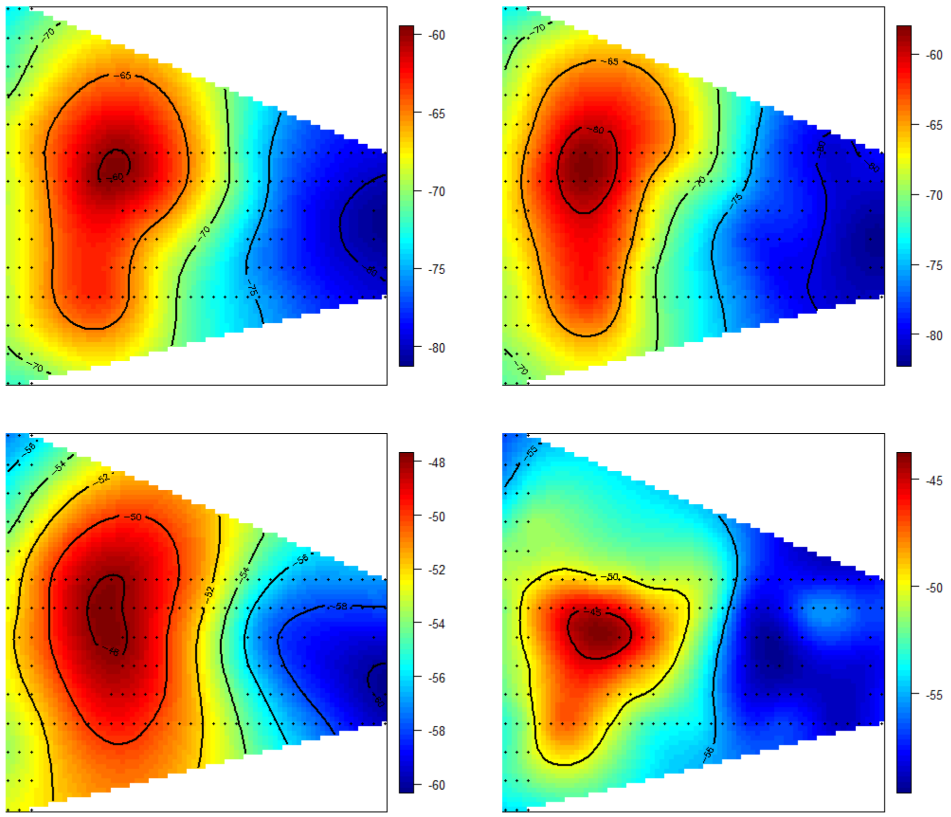


Figure 3

The top two heat-maps are access point two, 00:0f:a3:39:dd:cd, and the bottom two heat-maps are access point one, 00:0f:a3:39:e1:c0. The left two plots are readings at angle 0, and the right are angle 180. We fix the angles due to the information we learned from the boxplots, where angles affect signal strength. These two MAC addresses have overlapping heat-maps so we determine that the mysterious access point is this middle one. By creating heat-maps on each MAC address and counting the x,y centers of heat, we can determine the location of each access point, which is stored in a matrix called AP.

Implementation:

We used K-Nearest Neighbors (KNN) clustering to predict the online (test) data based on signal strength relations to each access point. However, what do we do with two MAC addresses in one access point? Do we use the MAC ending with c0, cD, or both? In order to determine what we should do, KNN is run for each of the three options listed, and their sum of square errors (SSEs) are compared.

The implementation of KNN began with creating a data frame of all training points on the floor, which will be used for visualization later. The online data was then read in using the readData function mentioned in the data processing section. Similar to how we created an offlineSummary data frame with summary statistics, an onlineSummary data frame was made and the x,y positions were extracted to actualXY. This actualXY data frame will be our test data-set to which we will compare our predictions. The FindNN function which finds the distance from a new point to all observations in the training data set. This is integrated into our larger predXY function, which estimates locations of new signals in terms of x,y coordinates based on the K nearest neighbors which are defined in the function call. This predXY is our KNN function. However, how do we know which number of nearest neighbors to use?

To solve this question, we implemented a function which graphs the sum of square error (SSE) vs the number of nearest neighbors, from 1 to 15 neighbors. The number of neighbors with the lowest SSE is chosen as the number of neighbors to be used in this KNN function. However, it is not that simple, as we have implemented V-Fold (normally called K-Fold, but V is used as it is different from the number of neighbors) cross validation. Cross validation is implemented to lower overfitting, due to training and testing on the same data set. The cross validation data set is chosen as one random angle for every one of the 166 locations, with the signal strengths being variables by MAC address. 11 folds are chosen because 11 divides into 166 with a remainder of 1, meaning that only one location will not be used in cross validating the data. This matrix is titled permuteLocs and is used to determine the folds for cross validation, where one fold is taken from onlineCVSummary, and the other 10 are taken from offlineSummary. This is repeated 11 times, once for each fold, so that the entire offlineSummary and onlineCVSummary is used. The predXY function, which does the KNN, is integrated within this for-loop of cross validation. The loop runs from 1 to 15 neighbors, for each fold, and the SSE is cumulatively calculated for each fold that is run.

The number of neighbors with the lowest SSE is chosen as the ideal number of neighbors. KNN clustering is run, using this ideal neighbor value, and we receive a data frame of x,y value pairs which correspond to the estimated locations from which the function thinks the experimenter was standing. Using the floorErrorMap function, the estimations were plotted on an image similar to Figure 1. These error plans can be found in the next section. After plotting the error plans, the SSE for this KNN was calculated using the calcError function and stored to be compared to the SSEs from the other MAC address permutations.

Results and Conclusion:

The implementation of this KNN clustering method with cross validation was run once with the cd MAC address removed, once with the c0 MAC address removed, and again with no MAC addresses removed. The SSE vs Neighbors graph is output to the working directory for each run, and so are the floor error maps. However, they appear below as Figures 4 – 6 for the SSE vs Neighbor plots, and 7 – 9 for the floor error maps. For the trials run at 15:20 EST, here are the results:

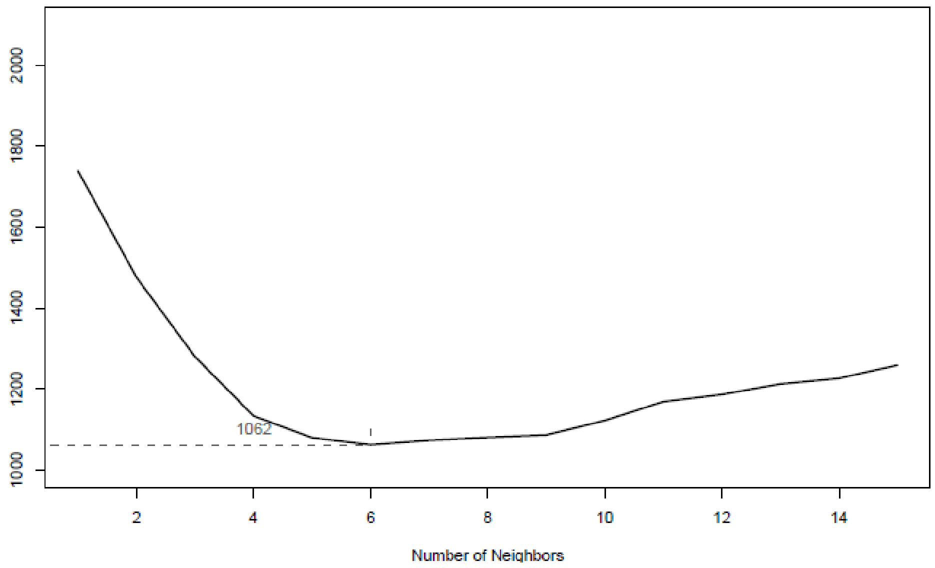


Figure 4: Cross Validation with MAC 00:0f:a3:39:e1:c0 Included

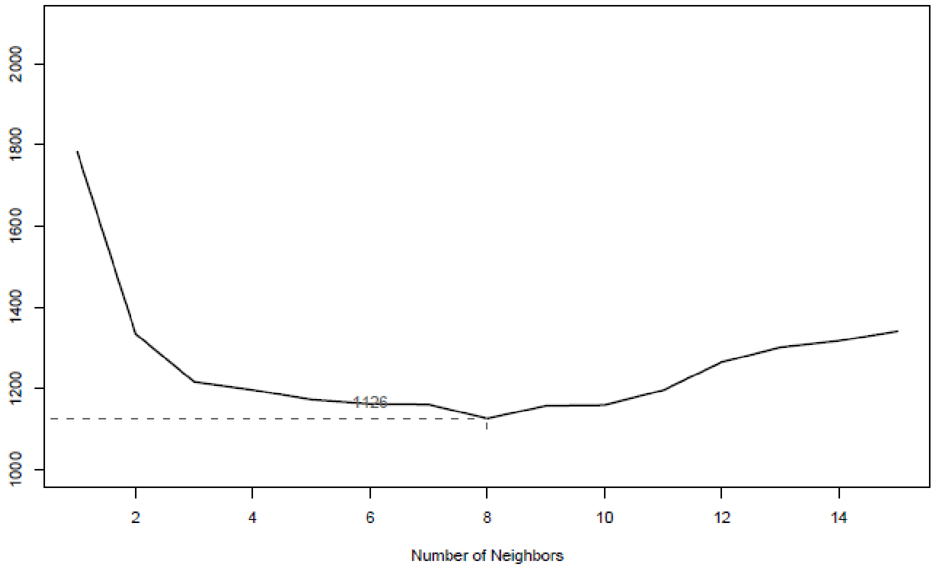


Figure 5: Cross Validation with MAC 00:0f:a3:39:dd:cd Included

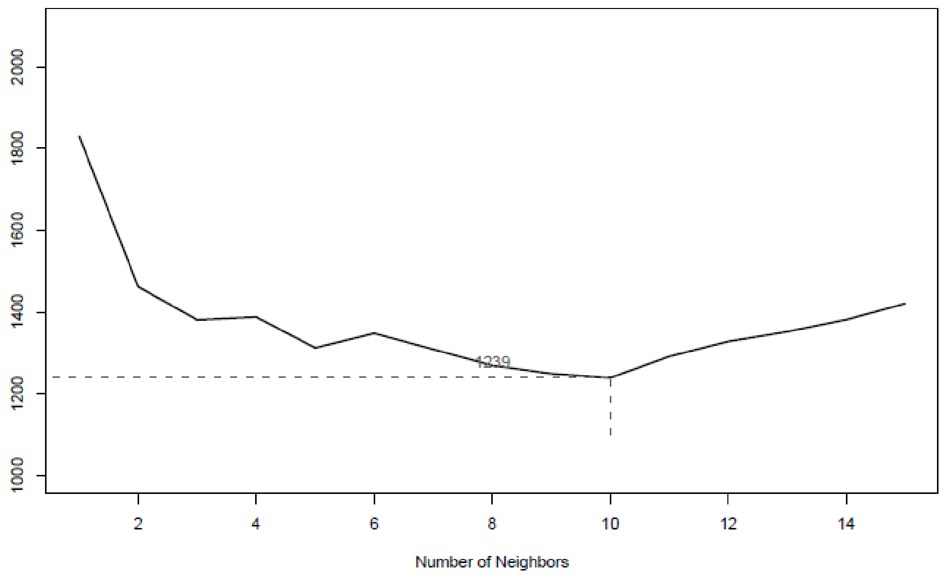


Figure 6: Cross Validation with Both MACs Included

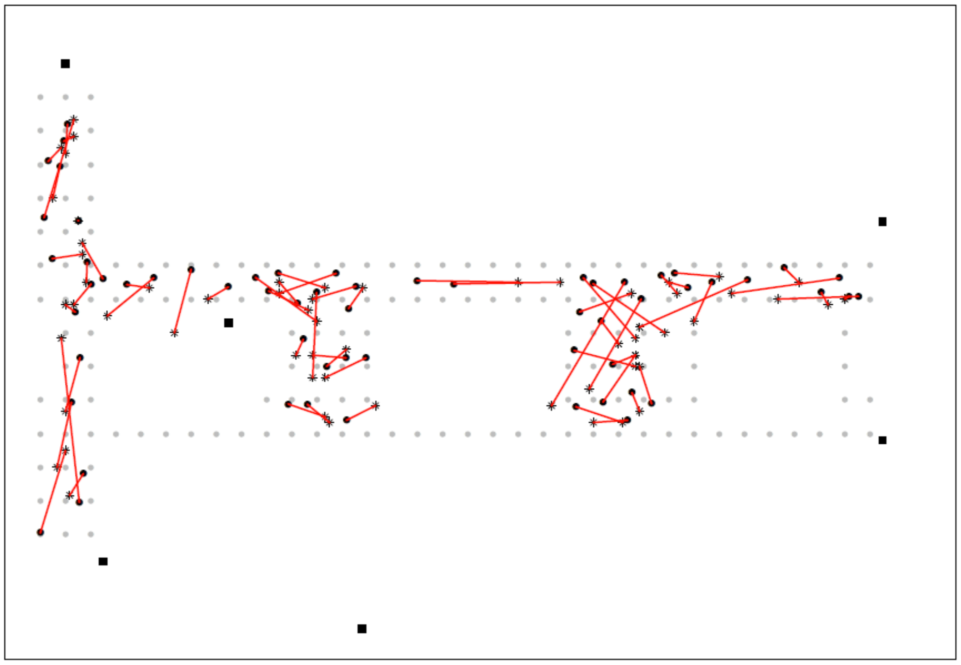


Figure 7: Floor Error Map with MAC 00:0f:a3:39:e1:c0 Included

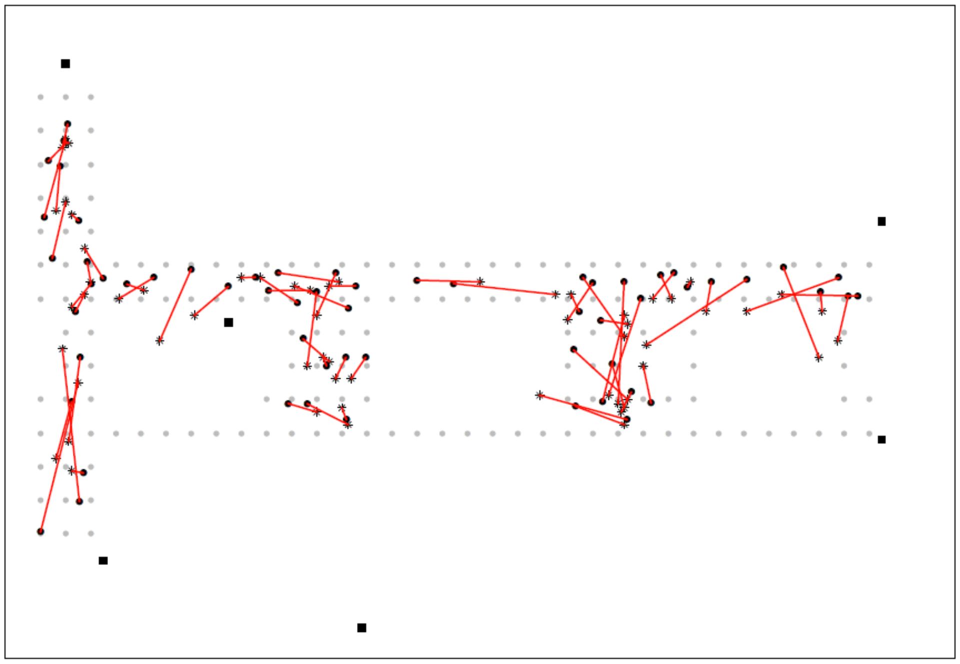


Figure 8: Floor Error Map with MAC 00:0f:a3:39:dd:cd Included

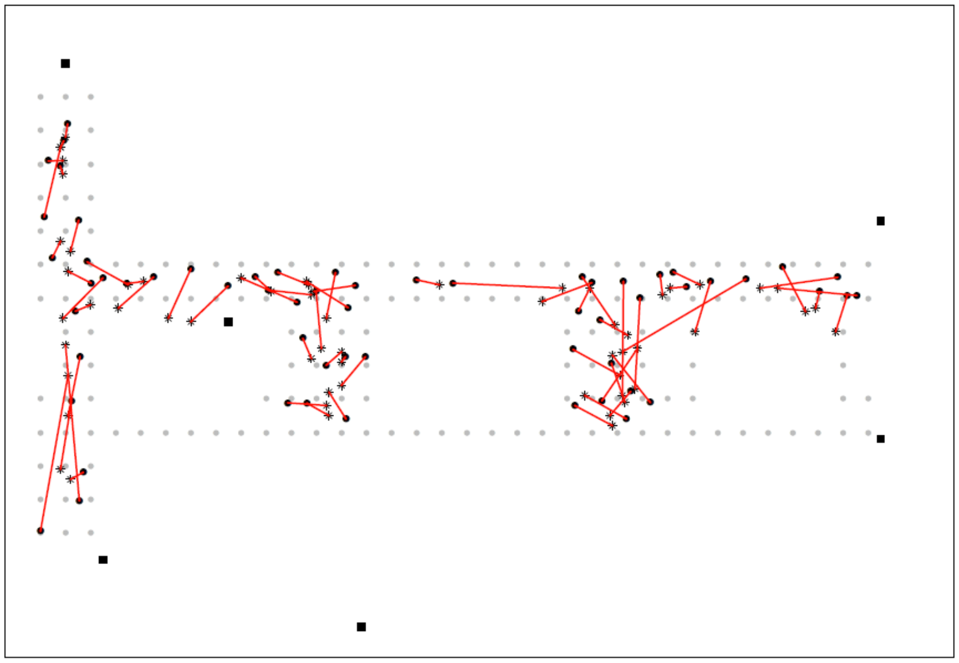
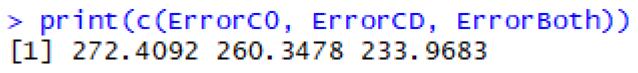


Figure 9: Floor Error Map with Both MACs Included



```
> print(c(ErrorC0, ErrorCD, ErrorBoth))
[1] 272.4092 260.3478 233.9683
```

Figure 10: Resulting SSEs for Each Run

As we can see from Figure 10, the run that uses both of the MAC addresses, had the lowest SSE. This is not unique to this run, as the code was run several times, with the orders of SSE being the same for all. Using the C0 MAC address has the largest SSE, the CD MAC address lands in the middle, and using both results in the lowest SSE, meaning that we should use both MAC addresses in our RTLS system.

Our implementation of weighted KNN was unsuccessful. We were unable to produce a function that generated predictions based on KNN with weights inversely proportional to the distance. These weights would have given preference to the closest neighbors to possibly produce better predictions. Although we were not able to get the code for weighted KNN to run, we did research various methods for weighting KNN.

Future Work:

While our implementation of KNN for predicting errors was successful, we would like to explore other methods for weighting KNN to see if we can improve on the results. For future experiments, use of access points at varying heights could prove interesting, such as access points on other floors to make use of the unused z axis variable which was dropped at the beginning of the data processing. Weighting by Manhattan distance and median distance are two alternative methods for implementing that we would like to explore in future work.

Aside from methodology that could be improved, there could also be improvements in the code itself. We would have liked to use an apply function to perform the cross validation KNN, but used a much slower and CPU intensive for-loop in the interest of time. Streamlining the three sections where the MAC addresses are used would be the biggest priority, either through a for-loop or another method, repeating 200 lines of code is not ideal, but it worked as a more heuristic approach. Going back to methodology, we would like to noise-proof the system somehow. This experiment was conducted in a controlled environment, however, in a real-life situation, there are tons of other objects and signals which may interfere with our RTLS system. At a later stage in the development of this system, we would like to implement counter-measures for this. Lastly, we would like to implement a weighted KNN strategy for future work. As unfortunate as it is, we were unable to this time around.

References:

[1] Nolan, D, “Data Science in R: A Case Studies Approach to Computational Reasoning and Problem Solving”

Appendix:

```
setwd("G:/JoshuaData/Classes/MSDS7333 Quantifying the World/Week 6/")

# Create function to format input into a matrix
processLine = function(x)
{
  tokens = strsplit(x, "[;=,]")[[1]]
  if (length(tokens) == 10)
    return(NULL)
  tmp = matrix(tokens[ - (1:10)],, 4, byrow = TRUE)
  cbind(matrix(tokens[c(2, 4, 6:8, 10)], nrow(tmp), 6,
               byrow = TRUE), tmp)
}

# Create function to return measurements to the proper 45 degree values
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]
}

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")

# Create function to read and format data
readData =
  function(filename = "G:/JoshuaData/Classes/MSDS7333 Quantifying the
World/Week 6/offline.final.trace.txt",
           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", ]

    # 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)

    # 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)

    # Create variable for location
    offline$posXY = paste(offline$posX, offline$posY, sep = "-")

    return(offline)
  }

offline = readData()

if(!require(fields)){
  install.packages("fields")
  library(fields)
}

if(!require(lattice)){
  install.packages("lattice")
  library(lattice)
}

# Create function which is used for plotting
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)
}

# Create a list of data frames for every combination of location (x,y),
angle, and mac address
byLocAngleAP = with(offline,
                    by(offline, list(posXY, angle, mac),
                       function(x) x))

# Create list of data frames with new summary variables
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
         })

# Combine list of data frames into one summary dataframe
offlineSummary = do.call("rbind", signalSummary)

# Create boxplot, showing signal strength varies by orientation and mac
address
bwplot(signal ~ factor(angle) | mac, data = offline,
       subset = posX == 2 & posY == 12,
       layout = c(2,4))

# Create multiple comparison heatmaps macs seperated with top being first
entry, angles being seperated by left being first entry
parCur = par(mfrow = c(2,2), mar = rep(1, 4))
mapply(surfaceSS, mac = subMacs[ rep(c(2, 1), each = 2)],
       angle = rep(c(0, 180), 2),
       data = list(data = offlineSummary))
par(parCur)

# Create singular heatmap
surfaceSS(mac=subMacs[1], angle = 0, data=offlineSummary)

####
##### Remove 00:0f:a3:39:dd:cd
####
{
```

```
# Remove 00:0f:a3:39:dd:cd mac from Offline Summary
offlineSummary = subset(offlineSummary, mac != subMacs[2])

AP = 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(subMacs[ -2 ], c("x", "y") ))

# Read online data
macs = unique(offlineSummary$mac)
online = readData("online.final.trace.txt", subMacs = macs)

#Create onlineSummary
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,
                             dimnames = list(ans$posXY, names(avgSS)))
                  cbind(ans, y)
                }))

onlineSummary = do.call("rbind", byLoc)

actualXY = onlineSummary[ , c("posX", "posY")]

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,
                               dimnames = list(ans$posXY,
                                               names(avgSS)))
                    cbind(ans, y)
                  }))

  newDataSS = do.call("rbind", byLocation)
  return(newDataSS)
}
```

```
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")
}

train130 = selectTrain(130, offlineSummary, m = 3)

### Using Training Data Set for Prediction with KNN

# Create a function which finds the distance from a new point to all
observations in the training data set
# "The parameters to this function are a numeric vector of 6 new signal
strengths and the return value from selectTrain()."

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 ])
}

# Create a function called predxy which estimates the location of new
signals, taking as input new signals, the angles at which thise signals
are taken,
#the train data, the number of angles around the initial, and the number
of nearest neighbors
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)
}


# Create a floor error map function
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")
}

# Create data frame of all training points on the floor
trainPoints = offlineSummary[ offlineSummary$angle == 0 &
                                offlineSummary$mac ==
"00:0f:a3:39:e1:c0" ,
                              c("posX", "posY")]

# Create a function which calculates the error of the KNN based on the
cumulative distance between estimated and actual points (red lines)
calcError =
  function(estXY, actualXY)
    sum( rowSums( (estXY - actualXY)^2) )

### Cross validating with K(V)-Fold CV
```

```
# Modify reshapeSS function to only return one randomized angle
(orientation) to be used in cross validation
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,
                               dimnames = list(ans$posXY,
                                               names(avgSS)))
                    cbind(ans, y)
                  }))

  newDataSS = do.call("rbind", byLocation)
  return(newDataSS)
}

# Omit the 7th mac address from the data set
offline = offline[ offline$mac != "00:0f:a3:39:dd:cd", ]


keepVars = c("posXY", "posX","posY", "orientation", "angle")
# Create a training CV data set
onlineCVSummary = reshapeSS(offline, keepVars = keepVars,
                            sampleAngle = TRUE)


# Establish variables for upcoming for loop: K=Maximum number of nearest
neighbors used V=Number of folds in CV (iterations of for loop)
# 11 is chosen as the largest number which divides closely to the total
number of samples (166) where 11 x 15 = 165
K = 15
err = rep(0, K)
v = 11

# Create a matrix of randomized positions
permuteLocs = sample(unique(offlineSummary$posXY))
permuteLocs = matrix(permuteLocs, ncol = v,
                     nrow = floor(length(permuteLocs)/v))

# Create a for loop in which CV is run
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[ , 6:11],
                    newAngles = onlineFold[ , 4],
                    offlineFold, numAngles = 3, k = k)
    err[k] = err[k] + calcError(estFold, actualFold)
  }
  print(err)
}

# Create a plot of SSE vs # of Neighbors
pdf(file = "SSEvsNeighborsC0.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(1000, 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()

minc0 = which.min(err)

# Run the KNN estimation baed on the least SSE from on the prior plot
estXYkc0 = predXY(newSignals = onlineSummary[ , 6:11],
                 newAngles = onlineSummary[ , 4],
                 offlineSummary, numAngles = 3, k = minc0)

pdf(file="FloorErrorMapC0.pdf", width = 10, height = 7)
oldPar = par(mar = c(1, 1, 1, 1))
floorErrorMap(estXYkc0, onlineSummary[ , c("posX","posY")],
              trainPoints = trainPoints, AP = AP)
par(oldPar)
dev.off()

ErrorC0 = calcError(estXYkc0, actualXY)

#Add Weighted KNN here
```

```
}

####
##### Remove 00:0f:a3:39:e1:c0
####
{
offline = readData()

offlineSummary = do.call("rbind", signalSummary)

# Remove 00:0f:a3:39:dd:c0 mac from Offline Summary
offlineSummary = subset(offlineSummary, mac != subMacs[1])

AP = 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(subMacs[ -1 ], c("x", "y") ))

# Create data frame of all training points on the floor
trainPoints = offlineSummary[ offlineSummary$angle == 0 &
                                offlineSummary$mac ==
"00:0f:a3:39:dd:cd" ,
                              c("posX", "posY")]

# Read online data
macs = unique(offlineSummary$mac)
online = readData("online.final.trace.txt", subMacs = macs)

#Create onlineSummary
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,
                             dimnames = list(ans$posXY, names(avgSS)))
                  cbind(ans, y)
                }))

onlineSummary = do.call("rbind", byLoc)

actualXY = onlineSummary[ , c("posX", "posY")]

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,
                               dimnames = list(ans$posXY,
                                               names(avgSS)))
                    cbind(ans, y)
                  }))

  newDataSS = do.call("rbind", byLocation)
  return(newDataSS)
}

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")
}

train130 = selectTrain(130, offlineSummary, m = 3)

### Using Training Data Set for Prediction with KNN

# Create a function which finds the distance from a new point to all
observations in the training data set
# "The parameters to this function are a numeric vector of 6 new signal
strengths and the return value from selectTrain()."
```

```
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 ])
}

# Create a function called predxy which estimates the location of new
signals, taking as input new signals, the angles at which thise signals
are taken,
#the train data, the number of angles around the initial, and the number
of nearest neighbors
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)
}


# Create a floor error map function
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")
}


# Create a function which calculates the error of the KNN based on the
cumulative distance between estimated and actual points (red lines)
calcError =
  function(estXY, actualXY)
    sum( rowSums( (estXY - actualXY)^2) )

### Cross validating with K(V)-Fold CV

# Modify reshapeSS function to only return one randomized angle
(orientation) to be used in cross validation
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,
                               dimnames = list(ans$posXY,
                                               names(avgSS)))
                    cbind(ans, y)
                  }))

  newDataSS = do.call("rbind", byLocation)
  return(newDataSS)
}

# Omit the 7th mac address from the data set
offline = offline[ offline$mac != "00:0f:a3:39:e1:c0", ]


keepVars = c("posXY", "posX","posY", "orientation", "angle")
# Create a training CV data set
onlineCVSummary = reshapeSS(offline, keepVars = keepVars,
                            sampleAngle = TRUE)


# Establish variables for upcoming for loop: K=Maximum number of nearest
neighbors used V=Number of folds in CV (iterations of for loop)
# 11 is chosen as the largest number which divides closely to the total
number of samples (166) where 11 x 15 = 165
K = 15
```

```
err = rep(0, K)
v = 11

# Create a matrix of randomized positions
permuteLocs = sample(unique(offlineSummary$posXY))
permuteLocs = matrix(permuteLocs, ncol = v,
                     nrow = floor(length(permuteLocs)/v))

# Create a for loop in which CV is run
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[ , 6:11],
                     newAngles = onlineFold[ , 4],
                     offlineFold, numAngles = 3, k = k)
    err[k] = err[k] + calcError(estFold, actualFold)
  }
  print(err)
}

# Create a plot of SSE vs # of Neighbors
pdf(file = "SSEvsNeighborsCD.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(1000, 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()

minCD = which.min(err)

# Run the KNN estimation baed on the least SSE from on the prior plot
estXYkCD = predXY(newSignals = onlineSummary[ , 6:11],
                  newAngles = onlineSummary[ , 4],
                  offlineSummary, numAngles = 3, k = minCD)
```

```
pdf(file="FloorErrorMapCD.pdf", width = 10, height = 7)
oldPar = par(mar = c(1, 1, 1, 1))
floorErrorMap(estXYkCD, onlineSummary[ , c("posX","posY")],
              trainPoints = trainPoints, AP = AP)
par(oldPar)
dev.off()

ErrorCD = calcError(estXYkCD, actualXY)

# Add weighted KNN Here
}

####
##### Use both mac addresses
####

{
offline = readData()

offlineSummary = do.call("rbind", signalSummary)


AP = 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(subMacs[ ], c("x", "y") ))

# Create data frame of all training points on the floor
trainPoints = offlineSummary[ offlineSummary$angle == 0 &
                                offlineSummary$mac ==
"00:0f:a3:39:dd:cd" ,
                              c("posX", "posY")]

# Read online data
macs = unique(offlineSummary$mac)
online = readData("online.final.trace.txt", subMacs = macs)

#Create onlineSummary
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,
                             dimnames = list(ans$posXY, names(avgSS)))
                  cbind(ans, y)
                }))
```

```
onlineSummary = do.call("rbind", byLoc)

actualXY = onlineSummary[ , c("posX", "posY")]

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$mac, mean)
                    y = matrix(avgSS, nrow = 1, ncol = 7,
                               dimnames = list(ans$posXY,
                                               names(avgSS)))
                    cbind(ans, y)
                  }))

  newDataSS = do.call("rbind", byLocation)
  return(newDataSS)
}

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")
}

train130 = selectTrain(130, offlineSummary, m = 3)

### Using Training Data Set for Prediction with KNN
```

```
# Create a function which finds the distance from a new point to all
observations in the training data set
# "The parameters to this function are a numeric vector of 6 new signal
strengths and the return value from selectTrain()."

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 ])
}

# Create a function called predxy which estimates the location of new
signals, taking as input new signals, the angles at which thise signals
are taken,
#the train data, the number of angles around the initial, and the number
of nearest neighbors
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)
}


# Create a floor error map function
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")
}


# Create a function which calculates the error of the KNN based on the
cumulative distance between estimated and actual points (red lines)
calcError =
  function(estXY, actualXY)
    sum( rowSums( (estXY - actualXY)^2) )

### Cross validating with K(V)-Fold CV

# Modify reshapeSS function to only return one randomized angle
(orientation) to be used in cross validation
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,
                               dimnames = list(ans$posXY,
                                               names(avgSS)))
                    cbind(ans, y)
                  }))

  newDataSS = do.call("rbind", byLocation)
  return(newDataSS)
}


keepVars = c("posXY", "posX","posY", "orientation", "angle")
# Create a training CV data set
onlineCVSummary = reshapeSS(offline, keepVars = keepVars,
                            sampleAngle = TRUE)
```

```
# Establish variables for upcoming for loop: K=Maximum number of nearest
neighbors used V=Number of folds in CV (iterations of for loop)
# 11 is chosen as the largest number which divides closely to the total
number of samples (166) where 11 x 15 = 165
K = 15
err = rep(0, K)
v = 11

# Create a matrix of randomized positions
permuteLocs = sample(unique(offlineSummary$posXY))
permuteLocs = matrix(permuteLocs, ncol = v,
                     nrow = floor(length(permuteLocs)/v))

# Create a for loop in which CV is run
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[ , 6:11],
                     newAngles = onlineFold[ , 4],
                     offlineFold, numAngles = 3, k = k)
    err[k] = err[k] + calcError(estFold, actualFold)
  }
  print(err)
}

# Create a plot of SSE vs # of Neighbors
pdf(file = "SSEvsNeighborsBoth.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(1000, 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()

minBoth = which.min(err)
```

```
# Run the KNN estimation baed on the least SSE from on the prior plot
estXYkBoth = predXY(newSignals = onlineSummary[ , 6:11],
                  newAngles = onlineSummary[ , 4],
                  offlineSummary, numAngles = 3, k = minBoth)

pdf(file="FloorErrorMapBoth.pdf", width = 10, height = 7)
oldPar = par(mar = c(1, 1, 1, 1))
floorErrorMap(estXYkBoth, onlineSummary[ , c("posX","posY")],
              trainPoints = trainPoints, AP = AP)
par(oldPar)
dev.off()

ErrorBoth = calcError(estXYkBoth, actualXY)

# Add weighted KNN Here
}

print(c(ErrorC0, ErrorCD, ErrorBoth))
```