Developing a Real Time Location Systems (RTLS)

1. Business Understanding

Overview: Real Time Location Systems (RTLS) have become increasingly popular with the need to track asset location, especially in places where the Global Positioning System (GPS) does not work so well such as indoor locations. An RTLS has a variety of applications such as tracking the location of medical equipment in hospitals, or that or robots in a factory warehouse. Knowing the exact location of assets in a building can lead to cost savings by enabling automation that can be built around the tracked location data. In other cases, cost savings can come from reduced labor costs in tracking equipment, especially when this needs to be repeated on a regular basis as part of day to day activities.

Triangulation: In this case study, we look at developing one such RTLS which will be used to track the location of assets in a section of a single floor in a building. One popular technique to do this is called Received Signal Strength Indicator (RSSI). In this technique, we have a few fixed readers placed at strategic locations within the area where the assets need to be tracked. These assets whose locations are to be evaluated are fitted with transmitters. These assets transmit a signal at regular intervals of time and the signal strength of these transmitted signals is recorded at the receivers (readers). According to Friis Free-Space equation, the signal strength recorded is inversely proportional to the square of the distance between the transmitter and the receiver (Fig. 1). In an ideal laboratory scenario, we could in theory use just 3 receivers to triangulate the location of an asset using the Friis Free-Space equation. However, in the real world, due to interference from other equipment in the vicinity and due to signal reflections from walls, this method is not feasible, and an alternative method must be used.

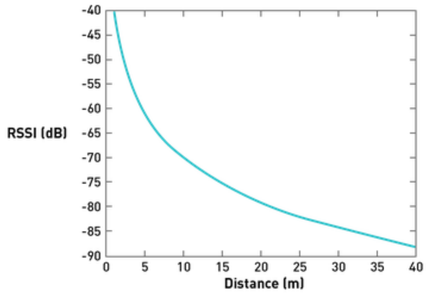

Fig. 1 Friis Free-Space Equation showing signal strength vs. distance [1]

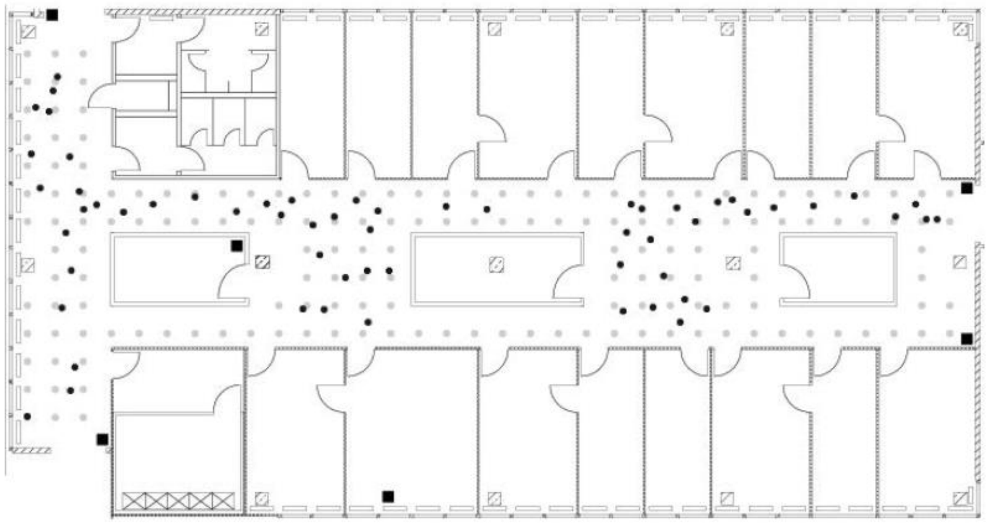

Fig. 2 Layout of the “space” used for predicting indoor location [2].

Nearest Neighbors: One such alternative method is to characterize the signal strength at various known locations (called offline points in this study and denoted by the grey dots in Fig. 2) in the space where the assets are going to be located. By measuring the signal strength from these known locations to each of the readers (black squares in Fig. 2), we can predefine the signal strength map of the locations in the space. Once this information has been built, we can then use this information as a lookup table to find the location of unknown assets in this space (referred to as online data points in this study and denoted by the black dots in Fig. 2). In practice, the grey dots are a small discrete subset of all possible locations in the space, whereas the black dots can be a continuous spectrum of locations. Since the location of the online points (black dots) is a superset of the location of the fixed offline positions (grey dots), we cannot directly use the lookup table values. Instead, we can use a modified approach called "nearest neighbors". In this approach, we find the "k" nearest known (offline) points to the online point under investigation. Points close to each other in the space are expected to have a similar received signal strength characteristic. Hence by comparing the received signal strength of the online point to the pre-characterized signal strength of the offline points at the same receivers, we can determine the vicinity (neighbors) of the online point. Then, to get a point estimate of its location, we can simply take an "average" of the location of the neighbors. This is an approximate method but is quite effective when accuracy is not of utmost importance and when the primary objective is to get an approximate location of an asset (accuracy is within a few feet of the actual location).

Assumption 1: The first assumption being made here is that locations close to each other will have a similar signal strength signature at the receivers. While this is expected to be the case in most scenarios, there are certain scenarios where the "nearest" neighbor in terms of actual distance is not the "nearest" neighbor" in terms of signal strength. This can happen when there are sharp changes in the environment around these two points, such as around wall edges. In such cases, signal from one point may have to travel through more walls to reach the receiver compared to the other point hence their signal strength signatures may not be similar.

Assumption 2: The second assumption being made is that the pre-characterized signal strength (from the offline points) does not change over time. However, this may not be the case in real life. New equipment added to the floor outside the space under consideration, or even to floors above or below the space under consideration, may interfere with or alter the signal strength signature that was pre-characterized earlier. We will provide methods to mitigate these issues during this case study.

2. Data Evaluation / Engineering

Measurement Setup: For this study, the predefined space was characterized using 166 offline locations placed on a 1x1 meter grid (9.1). At each location, the signal strength measurement was taken at 8 different orientations (reference angles) of the asset. These reference angles were multiples of 45 degrees (Fig. 3 x-axis) with slight variations in actual measurements (Fig. 3 y-axis) .

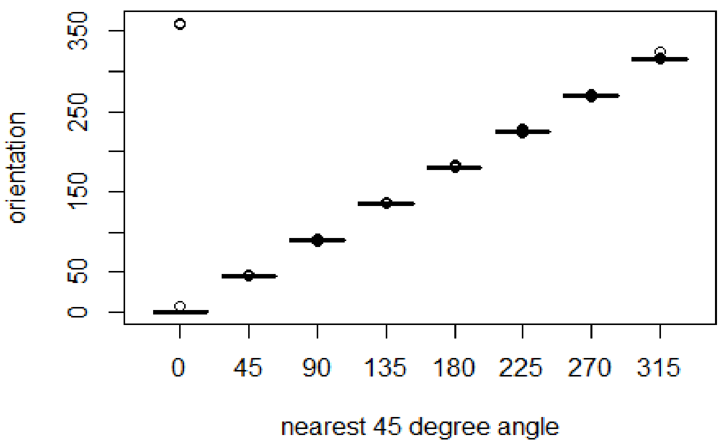

Fig. 3 Orientation vs. Nearest Reference Angle (multiple of 45 degrees)

Also, there can be fluctuations in the received signal strength from the same location and orientation (due to local effects such as another person walking by during the time of the signal measurement). In other cases, the signal may not be registered at the receiver. In order to overcome these challenges, 110 replications of the signal were taken from each location and orientation combination. Hence, a total of 166 x 8 x 110 = 146080 observations were made (9.2).

Orientation: Although the measurements were taken at 8 distinct orientations, due to manual error in placing the assets, the actual recorded orientation may differ slightly from the expected value (Fig. 3). This is important to note, and we will address this in the subsequent sections.

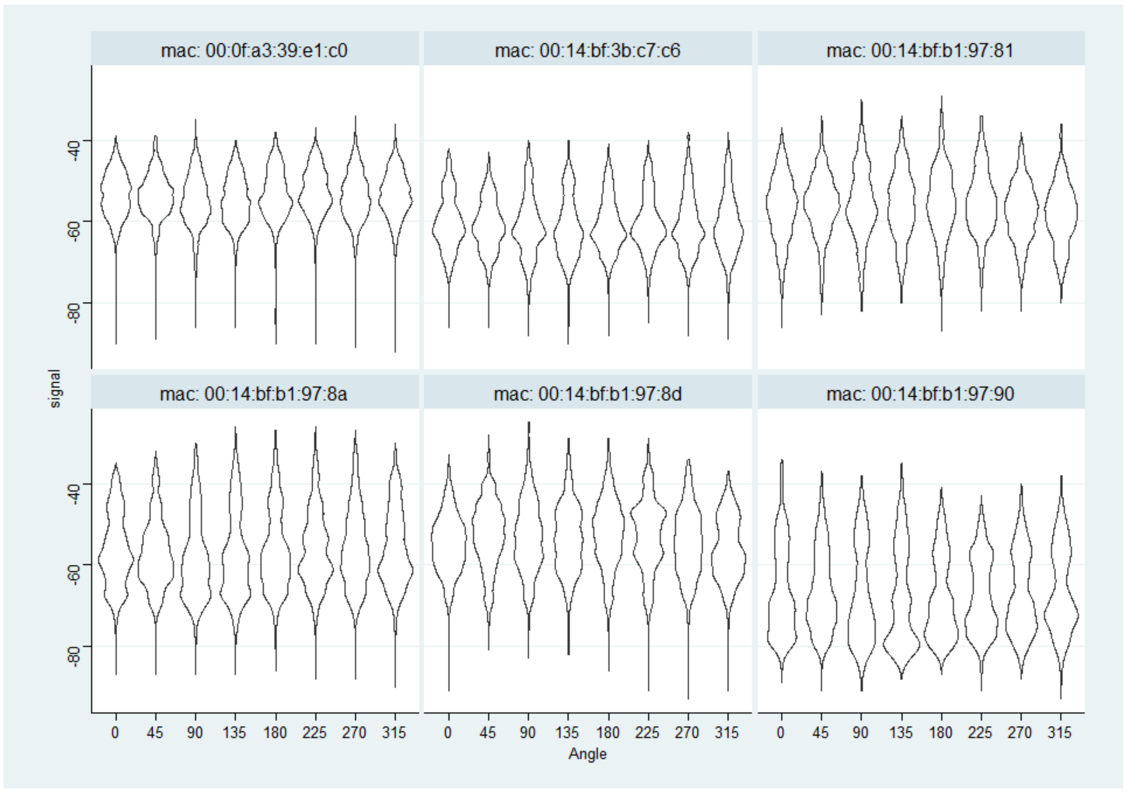

Fig. 4 Variation in signal strength at various readers for the different orientations/angles.

Also, the reason why measurement was taken at 8 different orientations at each location was because the received signal strength can change drastically depending on whether the asset is facing towards or away from the reader as can be seen in Fig. 4 and 9.3. This is an important consideration and will impact how we retrieve the dataset used to train our model.

Missing Values: As discussed above, some measurements may not be registered, and this shows up as missing values. The missing values at each reader is summarized in Table 1(Note that each reader is denoted by a unique MAC address). We will address handling of missing values in the subsequent sections.

Readers: Although there were only six fixed readers, one of the readers had two MAC addresses and the measurement was taken at both these MAC addresses. For our analysis, we will evaluate various combinations of including and excluding these MAC addresses to determine the best approach for our RTLS.

Table 1: Expected and missing records per each MAC address

MAC	Expected Records	Missing Records	% Missing Records
00:0f:a3:39:e1:c0	146,074	296	0.2%
00:0f:a3:39:dd:cd	146,074	1,823	1.2%
00:14:bf:b1:97:8a	146,074	24,412	16.7%
00:14:bf:3b:c7:c6	146,074	25,304	17.3%
00:14:bf:b1:97:90	146,074	26,691	18.3%
00:14:bf:b1:97:8d	146,074	28,463	19.5%
00:14:bf:b1:97:81	146,074	28,572	19.6%

3. Modeling Preparations

Rounding Angles: The "nearest neighbors" approach does not scale very well with the number of observations (since it is basically a lookup table approach). We saw above that the recorded orientation differed from the reference angles (multiple of 45 degrees) by a small amount in some cases. Since this fluctuation is not drastic, in order to reduce the complexity and the model evaluation time, we will round the observed orientation to the nearest reference angle (multiples of 45 degrees). This is implemented by the *roundOrientation* function shown in 9.4 and the results are shown in Fig. 3.

Collapsing replications to point measurements: We observed that we have 110 signal strength observations for every position and orientation combination. However, for the calculation of the "nearest neighbors", we need only one measurement at for each location and orientation combination. Then to get a point estimate of the signal strength at each offline location, an "average" statistic was used. We used the mean value of these 110 observations (since drastic outliers were not observed). If we suspect drastic outliers in the 110 replications, we could choose to use the median value of the signal strength since that is robust to outliers unlike the mean. Taking an average point estimate will also help us negate the effect of the few NA values that were observed in these 110 replications as well.

Distance Measure: Other considerations include the choice of a distance metric when computing the nearest neighbors. There are various distance metrics such as Euclidean (measures the diagonal distance between points) and Manhattan (measures the distance along a grid). These can be visualized in Fig. 5. Since the layout of our space is in a "grid pattern" (mostly horizontal and vertical walkways), we decided to choose the Manhattan distance for all our analysis. However, a Euclidean distance measure could also be used in this case. The code to calculate the distance based on the distance metric of choice is defined using the *findNN* function in 9.6

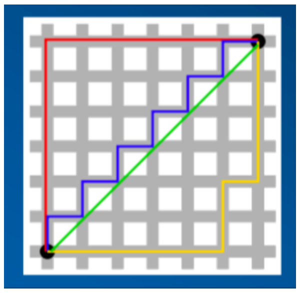

Fig. 5 Various Distance Measures: Green = Euclidean Distance, Rest: Manhattan Distance [1]

Metric: The metric used to measure the accuracy of the system depends on the distance metric chosen. For Euclidean distance, a Root Mean Squared Error or RMSE (Equation 1) would be an appropriate choice since it aligns well with the way the "diagonal" distance is calculated. For Manhattan distance, Mean Absolute Error or MAE (Equation 2) would be a more appropriate choice since it calculates the distance along the grid. Since we have 2 coordinates (X and Y), we will calculate this error metric across each coordinate and sum them up to get the resultant error at each location. Finally, this metric is averaged over the number of observations (N) so that we can compare 2 different scenarios that may have a different number of observations. A lower metric means a more accurate RTLS. The code to calculate the error metric is defined in *calcError* shown in 9.5.

Equation 1: $$RMSE = \sqrt{\frac{\sum (x_i - x_i')^2 + \sum (y_i - y_i')^2}{N}}$$

Equation 2: $$MAE = \frac{\sum |(x_i - x_i')| + |(y_i - y_i')|}{N}$$

Number of Neighbors: Other important considerations when performing the nearest neighbors is the choice of the number of neighbors. We saw in the section above, that choosing only one neighbor could lead to issues especially around wall edges. This also causes the results to not generalize well (overfitting). This is because even minor "local" fluctuations in the received signal strength can impact which neighbor is chosen and hence the ultimately impact the calculation of the position of the online point. On the other end of the spectrum, choosing a high number of neighbors will lead to a biased system. In this case, since the neighbors may be far away from the online point under investigation, we may not get a very accurate measurement. Hence, we need to find the "right" number of neighbors in order to have the most accurate measurements. The right number of neighbors can be obtained using a grid search approach where we vary the number of neighbors (k), record the chosen metric (from above) for each "k" value and finally pick the value of "k" that gives the best (lower) metric value.

Number of Orientations: From the data evaluation, we saw that the orientation can have a drastic impact on the signal strength. One approach to decide how many orientations to include in the nearest neighbor calculation would be to use just the offline orientations that are closest to the orientation of the online observation in order. Alternately, we could pick the two closest "reference" orientations on either side of the orientation of the online observation. Similarly, we could use 3, 4, 5, … , 8 closest orientations. It is again difficult to pick the "right" number of orientations to use in the final model and a grid search approach can be used here as well.

Leakage: Another very important consideration when building the model using grid search is the choice of dataset used to evaluate the results. We do not want to include the online data points in the choice of optimal "k" and number of orientations. This will result is what is called "leakage" and lead to a biased estimate since we technically will not have access to the online data when building the model. What we can do instead is use a technique called "v-fold cross validation". In this technique, the offline data (training) is divided into "v" segments. In each iteration, only "v-1" folds are used for training and the left out fold is used as a "validation" set to compute the metric. This process is repeated "v" times and in the end, each fold will be treated as the "validation" set once. The final metric can be obtained by averaging the metric across the "v" folds. The code to run "v" fold cross validation is defined by function *run_kfold* in 9.7.

Parallelization: To run the grid search, we picked number of neighbors from 1 to 20 and number of orientations from 1 to 8. In addition, we have a total of 11 folds for each hyperparameter combination. This results in building 20 x 8 x 11 (or 1760) models. If done in a brute force manner, this can be quite costly. Hence, we made some code optimizations to speed up this process as described below.

(1) The way the nearest neighbor code is organized, we can get the nearest neighbors back from 1 to N (where N is the number of training observations). Since we have access to the ordered distance to each training point, we only need to run this once and we can get the neighbors for all values of "k" from 1 to 20. As opposed to the brute force method where we run the nearest neighbor calculation for each value of k, this method is exponentially faster. This is implemented using the *all_K_means_single_obs* function shown in 9.7.

(2) Since each combination of number of neighbor and number of orientations are independent of each other, they can be run in parallel to speed up the hyperparameter search. For a computer with eight cores, a speed up of at least 6x can be observed taking overhead into account (*get_CV* function shown in 9.8).

Simplified Models: One of the challenges with the nearest neighbor algorithms is that usually as the number of neighbors increase, the error reduces, but the gains are only incremental. After a certain number of neighbors, the decrease in error does not justify increasing the number of neighbors since this can lead to a model that is underfit or biased and one that may not generalize well in the long run. One approach to pick a more parsimonious model that still performs well was suggested by Breiman et al. in 1984 [3] which recommends using the simplest model that is within one standard error of the empirically optimal model from grid search. In this case study, we use this approach to simplify the selected model in order to avoid underfitting (i.e. avoid selecting a greater number of neighbors than is absolutely needed). This method is an automated way to pick the "best" number of neighbors (and number of orientations) in lieu of using a more manual method that is traditionally used (knee or elbow method). The implementation of this method is shown in 9.9.

4. Modeling Scenarios (Original Case)

Though there were only six fixed readers used to measure signal strength, one of the readers had two MAC addresses (`00:0f:a3:39:e1:c0`, `00:0f:a3:39:dd:cd`) and the measurements were taken from both these MAC addresses. To understand how the signal measurements from these two MAC addresses would affect the model, we analyzed the following approaches.

1. Swapping the MAC address (`00:0f:a3:39:e1:c0`, `00:0f:a3:39:dd:cd`) and run the model with six MAC addresses only.
2. Including both the MAC addresses and run the model with seven MAC addresses.

The average value of the Mean Absolute Error across the "v" folds for these three different approaches were then compared to determine the best model. The results are shown in the form or a heatmap where the x-axis shows the number of neighbors and the y-axis shows the number of angles used in calculating the neighbors. The color of the heatmap shows the MAE. Greener colors denote a smaller MAE (better model) while redder shades represent a larger MAE (worse model). As explained above, OneSE approach was used to select the best parameters rather than the hyperparameters corresponding to the lowest MAE value.

4.1 Including MAC Address - `00:0f:a3:39:e1:c0`

Fig. 6 shows the cross validation results when MAC `00:0f:a3:39:e1:c0` is included. The best (lowest) MAE value from the grid search was ***2.703*** for number of orientations = 6 and number of neighbors "k" = 2.

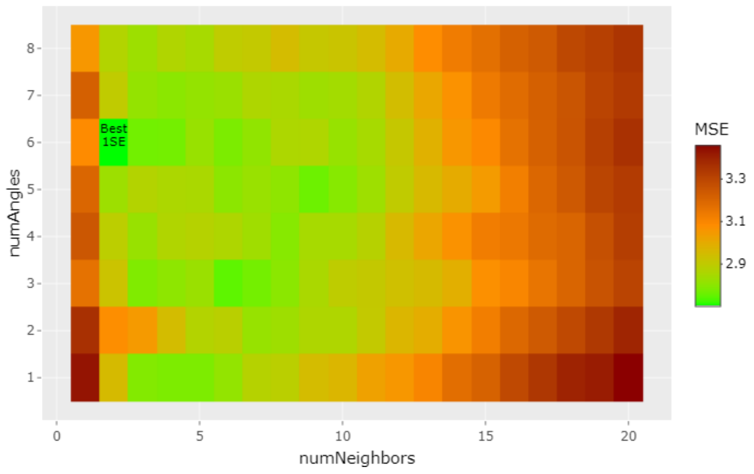

Fig. 6 Cross Validation Results including MAC: 00:0f:a3:39:e1:c0

4.2 Including MAC Address - `00:0f:a3:39:dd:cd`

Fig. 7 shows the cross validation results when MAC `00:0f:a3:39:dd:cd` is included. The lowest MAE value was ***2.473*** with number of orientations = 6 and number of neighbors "k" = 4.

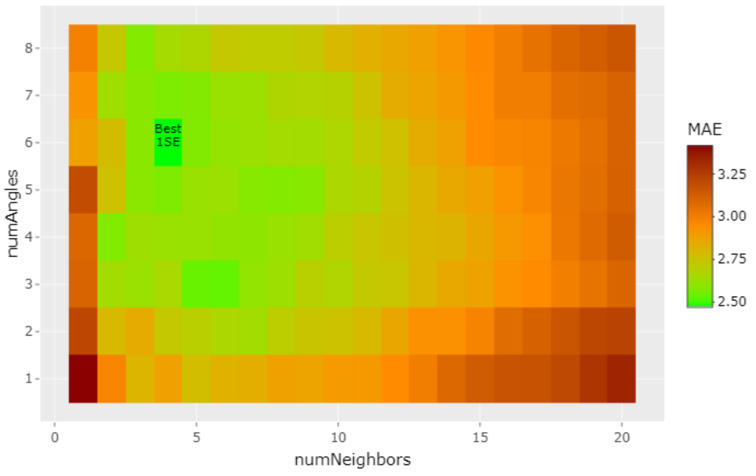

Fig. 7 Cross Validation Results including MAC: `00:0f:a3:39:dd:cd`

4.3 Including both MAC Addresses

Fig. 8 shows the cross validation results when both MAC `00:0f:a3:39:e1:c0` and `00:0f:a3:39:dd:cd` are included. The lowest MAE value was ***2.526*** with number of orientations = 5 and number of neighbors "k" = 5.

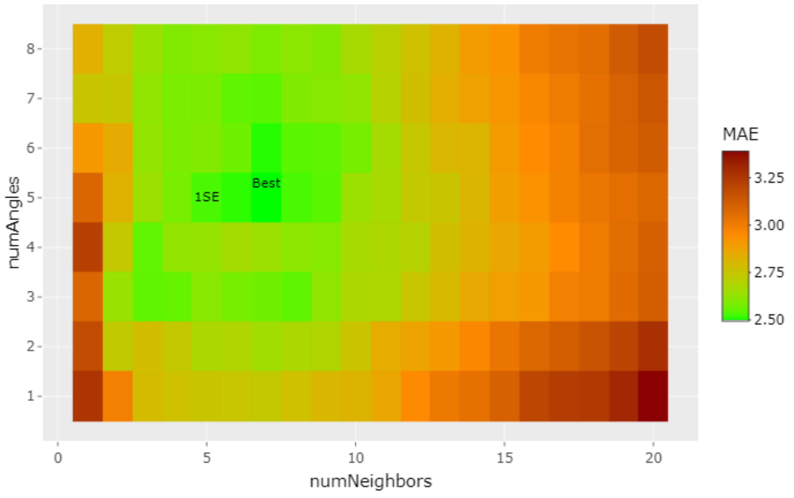

Fig. 8 Cross Validation Results including MAC: `00:0f:a3:39:e1:c0` and `00:0f:a3:39:dd:cd`

4.4 Summary

Table II Comparison of Nearest Neighbors model for the 3 scenarios

	`00:0f:a3:39:e1:c0`	`00:0f:a3:39:dd:cd`	**Both MAC Addresses**
Number of Neighbors	2	4	5
Number of Angles	6	6	5
MAE	2.703	2.473	2.526

Based on the above MAE values in Table II, it is evident that using the measurements from MAC address 00:0f:a3:39:dd:cd with 6 orientations and 4 nearest neighbors would result in a better model when compared to other options. We picked this model for further analysis and fine tuning.

5. Modeling Scenario (Extended Case)

The original nearest neighbor algorithm weighted all neighbors equally, even though they may be at different distances from the online point under consideration. One alternate would be to weight each neighbor based on the distance to the online point. So offline neighbors closer to online point under consideration would contribute more to the final coordinate prediction compared to the offline neighbors that were further out.

If we decide to use this approach, the next decision to make is how these neighbors are weighted. One approach would be to use a simple weight which is inversely proportional to the distance (Equation 3). Then the predicted coordinates is calculated as the weighted average of the individual coordinated of these 'k' neighbors (Equation 4).

Equation 3: $$w_i = \frac{\frac{1}{d_i}}{\sum_{i=1}^{k} \frac{1}{d_i}}$$

Equation 4: $$Predicted\ x = \frac{\sum_{i=1}^{k} w_i * x_i}{k} \quad Predicted\ y = \frac{\sum_{i=1}^{k} w_i * y_i}{k}$$

An alternate approach would be to use a leaf out of the Friis Free-Space equation. Since we know that the signal strength is inversely proportional to the square of the distance, we can adjust the weight accordingly instead (Equation 5). This has been implemented in the *findNN* function shown in 9.6. The equation to calculate the predicted x and y values remains the same as before.

Equation 5: $$w_i = \frac{\frac{1}{d_i^2}}{\sum_{i=1}^{k} \frac{1}{d_i^2}}$$

We reran the analysis with the weighted distance measure using the best model from before (one with MAC 00:0f:a3:39:dd:cd). Our analysis indicates that in this case, the choice of weighting the distances results in a MAE of 2.477.

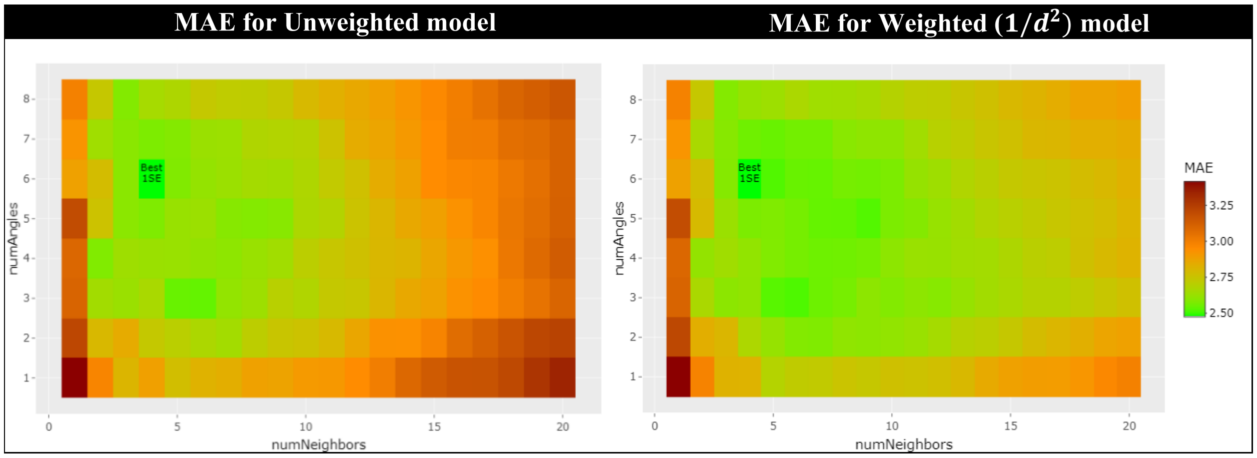

Fig. 9: Cross Validation Results between unweighted and weighted method for MAC `00:0f:a3:39:dd:cd`

Table 3: Comparing unweighted and weighted models

00:0f:a3:39:dd:cd	Unweighted	Weighted
Number of Neighbors	4	4
Number of Angles	6	6
MAE	2.473	2.477

Fig. 9 shows the MAE error across the different combination of neighbors and angles, and between weighted and unweighted model. The error improves significantly using the weighted model across the entire spectrum of hyperparameter values (more greener shades), although the minimum (and OneSE) value remains largely unchanged. In both cases, the OneSE method gives the best hyperparameters as 4 neighbors and 6 orientations. Although the best model did not show much improvement, the fact that MAE improves across the spectrum of hyperparameters values in general gives us a valid alternative to use as our final model. Hence, we will use this weighted model as the final model for online predictions.

6. Online Predictions

Once the final model has been determined, we use it to make predictions on the online data. The online data represent a selection of measurements not used during the model training and hyperparameter selection. This final step gives a measure of how well the model can be generalize to future real-life data. Fig. 10 shows the real position of the online data and the error in predicting its actual location. As with the heatmap above, greener shades indicate a lower error while redder shades indicate a larger error.

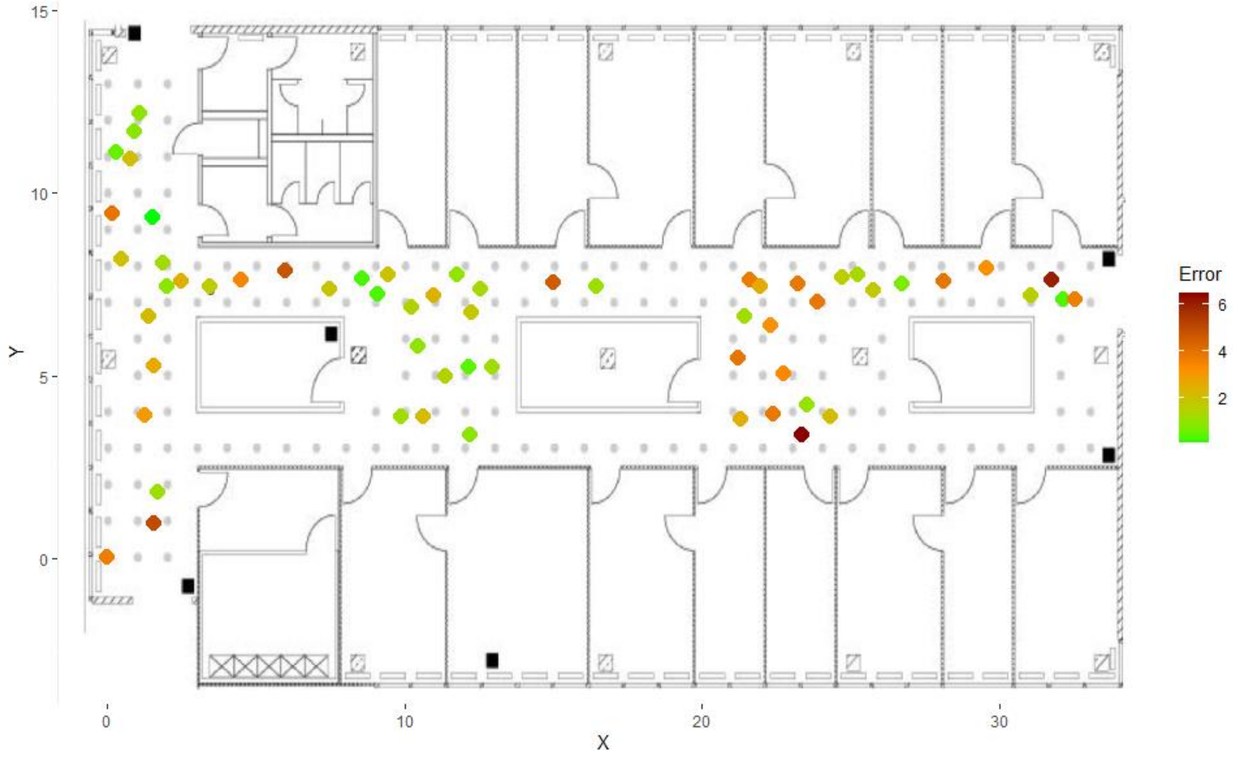

Fig. 10 Position of online measures and their error to the estimated position

Spatial Dependence: We can notice there are some areas with a concentration of low errors indicated by the greener shades (left side of the space) and some with a concentration of high errors indicated by redder shades (right side of the room). The errors can be caused by the presence of walls and beans or a lack of readers in the vicinity that can create a non-uniform distribution of signal's strength between offline points.

Temporal Dependence: The errors may also be caused by change in the environment between the offline and online measures, like people moving in the building. Analyzing the hour in the day when the measures where take (Fig. 11), we can see the offline measures were taken from midnight to 2pm, while online measure where taken between 11am to 3pm. The environment in the building may be significantly different between these hours, as more people may be present during business hours and they may cause more interference in the measurements compared to the night hours.

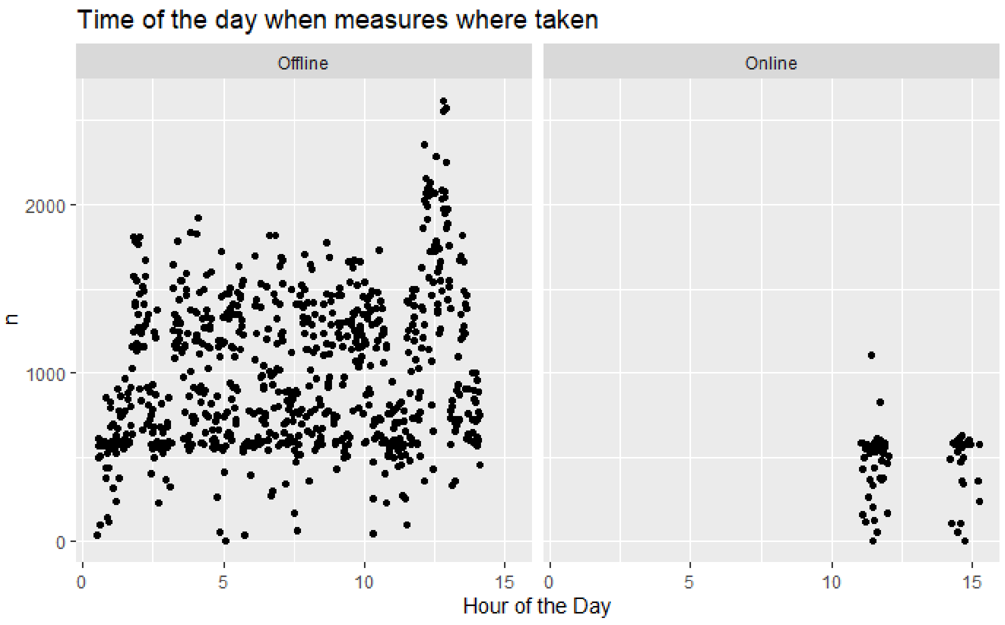

Fig. 11 Distribution of measures taken during the time of the day

7. Conclusion

Model Selection: From the above analysis, we can conclude that including MAC address 00:0f:a3:39:dd:cd with 6 orientations and 4 nearest neighbors resulted in the best model. This model was further refined by using a weighted nearest neighbors algorithm which provided a lower error across the board (though the absolute lowest error value remained the same). Given the above analysis, the model that should be deployed in production should be the one that includes the MAC address 00:0f:a3:39:dd:cd with 6 orientations and 4 nearest neighbors using a weighted distance metric.

Accuracy of the System: Our RTLS has an accuracy of 2.477 (MAE). This means that on “average”, we can accurately pinpoint the asset within 2.477 meters of its actual location (along a grid, i.e. Manhattan distance). If this is not good enough, we recommend adding more readers to strategic locations in order to improve the accuracy. For example, Fig. 10 shows that the right half of the space has a larger error in general than the left half of the space. Hence as a first step, we can reposition some of the existing readers from the left to the right to see if it improves the error (this would not incur any addition hardware cost). If this does end up improving the accuracy, then we can also consider adding additional readers on the right half of the space keeping those on the left half intact. This would incur additional hardware cost but may be worth the expense if the desired accuracy can be achieved.

Data Collection Improvement: We saw from Fig. 11 that the data collection times for the offline and online data did not match up completely. This process is called “covariate shift” and can lead to inaccuracies in the predictions after deployment. It might be more representative if the offline measurements were taken during the same time slots as what would be used when the actual real time (online) predictions would need to be made in the future. This would result in a model that can generalize much better in the future when presented with unseen data.

Other Methods and Metrics: We picked Manhattan distance in our algorithm, but Euclidean could also be used. An important consideration here is that if we decide to make comparisons using both Euclidean and Manhattan distance in the future, we need to use a common metric (we cannot use RMSE in one case and MAE in the other case). Also, as discussed earlier, data at each location-orientation could be collapsed using median measures if outliers are suspected.

Recalibration: Assumption 2 violation can be avoided by recalibrating the system from time to time. Of course, this adds additional cost of taking new offline measurements, but may be worth it if we suspect material change in the environment in and around the space under consideration.

Conclusion: We were able to get accurate estimates of the location using just 6 readers and this might be accurate enough for our needs. We also presented ways in which accuracy could be improved in the future. Some of these methods don't need to incur additional capital (hardware) cost while others may need to incur some additional capital cost. We will leave this as an exercise after performing post model deployment analysis and assessment.

8. References

[1] D. Engels and R. Slater, *Quantifying the World,* Southern Methodist University.

[2] D. Nolan and D. T. Lang, Data Science in R: A Case Studies Approach to Computational Reasoning and Problem Solving.

[3] L. Breiman, J. Friedman, C. Stone and R. A. Olshen, Classification and Regression Trees (Wadsworth Statistics/Probability), 1984.

9. Appendix

9.1 Unique X and Y coordinates of the offline data points

```
unique(offline$posX)

##  [1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
## [26] 25 26 27 28 29 30 31 32 33

unique(offline$posY)

##  [1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13
```

9.2 Number of observations: After removing the comments (151392 – 5312), we have 146,080 observations.

```
# Number of comments in offline file
sum(substr(txt, 1, 1) == "#")

## [1] 5312

# Number of total lines in the offline file
length(txt)
```

```
## [1] 151392
```

9.3 Plot showing the variation of signal strength vs. angle for various readers

```
p=ggplot(data=filter(offline, mac != "00:0f:a3:39:dd:cd")
         ,aes(y=signal,x=factor(angle))) +
  ggplot2::geom_violin()+
  labs(x='Angle')+
  ggplot2::facet_wrap(vars(paste0('mac: ',mac))) +
  ggthemes::theme_stata()
```

9.4 Rounding of orientation to the nearest reference angle

```
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$angle = roundOrientation(offline$orientation)

with(offline, boxplot(
  orientation ~ angle,
  xlab = "nearest 45 degree angle",
  ylab="orientation")
)
```

9.5 Error Calculation

```
calcError = function(estXY, actualXY, method = DIST_METHOD){
  if('numeric' %in% class(estXY)) rows = 1 else rows = nrow(estXY)
  if(method == 'Euclidean')  er = sqrt(sum(rowSums((estXY - actualXY)^2)))/rows
  if(method == 'Manhattan')  er = sum(rowSums(abs(estXY - actualXY)))/rows
  return(er)
}
```

9.6 Find the nearest Neighbors

```
DIST_METHOD = 'Manhattan'
weightFormula = function(x){ 1/(x ^ 2)} #inverse of sq distance

#' @description Computes the distance of the new signal (single observation) to each
observation in the training dataset
#' @param newSignals The Signal Values for the validation data for each observation
#' @param trainSubset The training data to be used
#' @param weighted Whether the mean value should be weighted based on distancde or no
t.
#' @return A dataframe containing same number of rows as that in the training data.
#'   The observations are ordered by the distance to the new signal. Each row contain
s 5 columns.
```

```
#'   1st column is the XY location of the training observation (string)
#'   2nd column is the X location of the training observation (float)
#'   3rd column is the Y location of the training observation (float)
#'   4th column is the distance to the point under consideration to the training obse
rvation (float)
#'   5th column is the inverse distance or weight (float). Weight is hard coded to 1
for all observations if weighted = FALSE
findNN = function(newSignal, trainSubset, weighted=FALSE, method = DIST_METHOD) {
  diffs = apply(trainSubset[ , 4:(4+numMacs-1)], 1, function(x) x - newSignal)
  if(method=='Euclidean')  dists = apply(diffs, 2, function(x) sqrt(sum(x^2)) )
  if(method=='Manhattan')  dists = apply(diffs, 2, function(x) sum(abs(x)) )
  closest = order(dists)

  ordered_dist = dists[closest]
  if(weighted == TRUE){
    weight = weightFormula(ordered_dist)
  }
  if(weighted == FALSE){
    weight = rep(1, length(dists))
  }
  return(cbind(trainSubset[closest, 1:3], ordered_dist, weight))
}
```

9.7 Faster Cross Validation (computing distance only once)

```
#' @description Modified XY Prediction to help with faster CV for all K values at onc
e (from 1 to K)
#' @param newSignals The Signal Values for the validation data for each observation
#' @param newAngles The Orientation of the validation data for each observation
#' @param trainData The training data to be used
#' @param numAngles Number of closest reference angles to include in the data
#' @param K Perform the prediction for num neighbors from 1 to K
#' @param weighted Whether the mean value should be weighted based on distancde or no
t.
#' @return A nested dataframe with num rows = number of (validation) observations and
num columns = number of folds
#'   Each entry in this dataframe is a vector of 2 values indicating the prediction o
f the mean X and Y values for that obs and num neighbors
predXYallK = function(newSignals, newAngles, trainData, numAngles = 1, K = 10, weight
ed=FALSE){
  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, ]),
      trainSubset = trainSS,
      weighted = weighted
    )
  }

  #' @description Returns the (un)weighted mean X and Y locations for a single observ
ation and multiple neighbor values
  #' @param x Dataframe containing 5 columns
  #' 1st column is the XY location (string)
```

```r
  #' 2nd column is the X location (float)
  #' 3rd column is the Y location (float)
  #' 4th column is the distance to the point under consideration (float)
  #' 5th column is the inverse distance or weight (float)
  #' @param K Number of nearest neighbors to use
  #' @return A list of K pairs (each pair is a XY mean value for a single k)
  all_K_means_single_obs = function(x, K){
    # Row will contain the K mean values for k = 1 to K
    rows = list()
    for(k in seq(1, K)){
      rows[[k]] = k_means_single_obs(x, k)
    }
    return(rows)
  }

  #' @description Returns the (un)weighted mean X and Y locations for a single observ
ation and single value of neighbors
  #' @param x Dataframe containing 5 columns
  #' 1st column is the XY location (string)
  #' 2nd column is the X location (float)
  #' 3rd column is the Y location (float)
  #' 4th column is the distance to the point under consideration (float)
  #' 5th column is the inverse distance or weight (float)
  #' @param k Number of nearest neighbors to use
  #' @return A pair of XY mean values for k number of neighbors
  k_means_single_obs = function(x, k){
    weights = x[1:k, 5]
    weighted_x = sum(x[1:k, 2] * weights) / sum(weights)
    weighted_y = sum(x[1:k, 3] * weights) / sum(weights)
    return(c(weighted_x, weighted_y))
  }

  estXY = lapply(closeXY, all_K_means_single_obs, K)
  estXY = do.call("rbind", estXY)
  return(estXY)
}

#' @description Returns the (un)weighted mean X and Y locations for a single observat
ion and multiple neighor values
#' @param K Number of nearest neighbors to use (Will run Grid Search over all values
from k = 1 to K)
#' @param v Number of folds to use
#' @param offline Use "as is" from script for now
#' @param onlineCVSummary Use "as is" from script for now
#' @param folds A matrix with rows = number of observations in each fold and columns
= number of folds.
#'              The values are the XY IDs to be included in that fold
#' @param numAngles Number of closest reference angles to include in the data
#' @param weighted Whether the mean value should be weighted based on distancde or no
t.
#' @return A vector of K values indicating the Error for each value of k from 1 to K
run_kfold = function(K, v, offline, onlineCVSummary, folds, numAngles, weighted=FALSE
){
  err= rep(0, K)
  errCV = rep(0, K)
  allErr = data.frame()
  for (j in 1:v) {
    print(paste("Running Fold: ", j))
    onlineFold = subset(onlineCVSummary, posXY %in% folds[ , j])
    offlineFold = subset(offline, posXY %in% folds[ , -j])
    actualFold = onlineFold[ , c("posX", "posY")]

    estFold = predXYallK(
        newSignals = onlineFold[ , 6:(6+numMacs-1)],
        newAngles = onlineFold[ , 4],
        trainData = offlineFold,
        numAngles = numAngles,
        K = K,
        weighted=weighted
      )
    # Reformat into correct format for each 'k' value
    for(k in 1:K){
      estSingleK = data.frame()
      for(i in seq(1, length(estFold)/K)){  # i = NUmber of the observtion
        estSingleK = rbind(estSingleK, t(as.data.frame(estFold[i,k])))
      }
      err[k] = err[k] + calcError(estSingleK, actualFold)
      errCV[k] =  calcError(estSingleK, actualFold) #returning all folds
    }
    allErr=rbind(allErr,data.frame('fold'=j, 'numNeighbors' = 1:K,'errValue' = errCV)
)
  }

  return(list(err=err,allErr=allErr))
}
```

9.8 Parallel Cross Validation

```r
get_CV = function(K,v,offline,onlineCVSummary,permuteLocs,numAngles,weighted = TRUE){
  library(foreach)
  library(progress)
  library(doParallel)
  library(doSNOW)
  cl <- makeCluster(detectCores())
  doSNOW::registerDoSNOW(cl)

  allErrors = data.frame()

  start = proc.time()

  pb <- progress::progress_bar$new(total = length(allAngles),format='[:bar] :percent
:eta')
  progress <- function(n) pb$tick()
  allErrorsCV = foreach(numAngles = allAngles
                        ,.combine = rbind
                        ,.options.snow = list(progress=progress)
                        ,.export =
c('run_kfold','predXYallK','reshapeSS','findNN','calcError','numMacs','selectTrain','
roundOrientation','DIST_METHOD','weightFormula')
```

```
  ) %dopar% {
    print(paste("Running ", v, "-fold cross validation with 1 to ", K, " neighbors,
for number of Angles = ", numAngles))
    err = run_kfold(
      K = K,
      v = v,
      offline = offline,
      onlineCVSummary = onlineCVSummary,
      folds = permuteLocs,
      numAngles = numAngles,
      weighted = weighted
    )
    err$allErr$numAngles = numAngles

    return(err$allErr)
    #return(data.frame(t(err)))
  }
  stopCluster(cl)
  stop = proc.time()
  diff = stop-start
  print(diff)

  return(allErrorsCV)
}
```

9.9 Finding the best hyperparameters

```
find_best = function(allErrorsCV){
  library('caret')
  library(tidyverse)
  allErrors = allErrorsCV %>%
    group_by(numAngles,numNeighbors) %>%
    dplyr::summarise(errValue = mean(errValue)) %>%
    ungroup() %>%
    mutate(errValueSD=sd(errValue), best=FALSE, oneSE=FALSE)

  allErrors[best(as.data.frame(allErrors), "errValue", maximize=FALSE),]$best=TRUE
  allErrors[oneSE(as.data.frame(allErrors),"errValue", maximize=FALSE,
num=30),]$oneSE=TRUE

  return(allErrors)
}
```

9.10 Online Data Error Plot

```
floorErrorMap = function(estXY, actualXY,method = DIST_METHOD){

  if(method == 'Euclidean')  er = sqrt(rowSums((estXY - actualXY)^2))
  if(method == 'Manhattan')  er = rowSums(abs(estXY - actualXY))
  data = cbind(actualXY,er)
  library(png)
  library(ggpubr)
  img <- png::readPNG('building.png')
  p = ggplot(data=data,aes(x=posX,y=posY,color=er)) +
    background_image(img) +
    labs(x='X',y='Y',color='Error')+
    ggplot2::scale_y_continuous(limits=c(-3.1,14.4))+
    ggplot2::scale_x_continuous(limits=c(0,33.4))+
    ggplot2::geom_point(shape=16,size=4) +
    scale_color_gradient2(low = "green",mid='darkorange', high = "darkred", na.value
= NA
                               ,midpoint=mean(c(max(data$er),min(data$er)))
    )
  return(p)

}
```

9.11 Time of the day plot

```
library(lubridate)
```

```
offlineTime = offline$time /1000
class(offlineTime) = c("POSIXt", "POSIXct")
dfOffT= data.frame('hour'=hour(offlineTime) + lubridate::minute(offlineTime)/60) %>%
count(hour)
dfOffT$type = 'Offline'

onlineTime = online$time /1000
class(onlineTime) = c("POSIXt", "POSIXct")
dfOnT= data.frame('hour'=hour(onlineTime)+ lubridate::minute(onlineTime)/60) %>%
count(hour)
dfOnT$type = 'Online'
dfTime = rbind(dfOffT, dfOnT)
ggplot(dfTime,aes(x=hour,y=n)) + geom_point() + ggtitle('Time of the day when
measures where taken') +
  facet_wrap(vars(type)) + labs(x='Hour of the Day')
```