Project Report

Title: Indoor Navigation using WiFi Localisation in Android

Author: Divisha Gupta
Roll No.: 2017A7PS0959G
Email id: f20170959@goa.bits-pilani.ac.in

Prepared in partial fulfillment of the requirements of the course
CS F266: STUDY PROJECT
Under the guidance of Prof. Vinayak Naik

AT

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE
K.K. BIRLA GOA CAMPUS

Table of Contents

Acknowledgments

I would like to express my special thanks of gratitude to Prof. Vinayak Naik, who gave me the opportunity to work on this project. His constant guidance, comments, and suggestions inspired me to progress through this project.

I would also like to thank Ph.D. student Yogesh Dasgaonkar for his valuable inputs and mentorship. He provided me with all the relevant study materials.

Lastly, I would like to thank my project members, Chinmay Gupta, Rajat Gupta, and Deepak Divya Tejaswi Rao for their cooperation and support.

Problem Statement

Outdoor navigation using GPS is a very common phenomenon nowadays that finds its application in our daily lives. Apps like Google Maps, Waze, Apple Maps, etc. have helped in creating this a reality. So much so is the advancement that we can have a real-time satellite view of the route we are trying to navigate through.

But when it comes to Indoor localization and navigation, this technology is still in its nascent stages, because like satellites there is no infrastructure that provides an indoor view of a building.

Therefore, there is a need for user-friendly applications that let people navigate through unknown buildings using alternate technologies. It can be used to locate people/objects in malls, hospitals, airports, and multi-storey parking lots.

Related Work

We conducted a literatue survey in the area of Indoor Localization using WiFi. The thesis (Zhang 2018) describes a non-intrusive system to find parking space using the CSI based WiFi technology. It argues that the features of CSI data of received WiFi signals have a strong correlation with number of empty slots. It therefore deploys Machine Learning technique to perform multiclass classification and is able to achieve an accuracy of 93.8% proving its feasibility.

The survey (Ma et al. 2019) is a comprehensive literature survey that gives a review of the signal processing techniques, algorithms, applications, performance results, challenges, and future trends of WiFi sensing with CSI. It discusses the significance of multiple state information (MIMO) technology of WiFi, which uses multiple antenna to generate a channel state information (CSI) matrix to get the subcarrier frequency amplitudes

These works discuss the use of CSI in localization. CSI provides information about both amplitude and phase angle of the incoming WiFi signals. But due to certain limitations, our project uses RSSI values of which report only the amplitude.

The paper (Lemic et al 2014) discusses Indoor Localisation using RSSI values of WiFi signals. It introduces the fingerprint-based pattern matching technique and discusses multiple algorithms to create these *fingerprints*. There are separate parameters to measure the similarity between these fingerprints for each algorithm.

Approach

We try to tackle the problem of Indoor Localisation using WiFi RSSI (*Received signal strength Indication*) signals. Every position in a 3D space is uniquely characterized by the different RSSI signals from the WiFi routers installed in the vicinity.

Suppose there are 2 WiFi routers A and B. If we scan for WiFi signals near router A, we'll get stronger signals from router A and weaker from router B. Vice versa when scanned near router B. The signal strength will remain more or less the same around these points. Since every router has a unique MAC address (*BSSID*), we can map a composition of scanned WiFi RSSI signals to the corresponding location without really knowing the location.

This approach can be extended to multiple locations in a building that has a uniformly spread WiFi network. Every location will have a fixed pattern for RSSI signals according to the spatial arrangement of the routers around the location and after studying these *"patterns"* we can predict locations directly from RSSI values.

Data Collection

The new academic building (NAB) for CS Dept. of BITS Pilani, Goa has a uniformly spread network of WiFi routers and was used to collect data for this project. It is a double-storey building with both floors completely identical. 24 locations were selected, 12 on each floor.

The mobile app *WiFi Analyzer* (available on PlayStore) was used for scanning. The source code of App was tweaked to scan WiFi signals every 1 second for 1 minute (App performed 60 scans in every run). 4 different devices were used at the same time and the devices were kept at random separation from one another to introduce some variance in the data. Data was extracted in the form of .csv files with fields namely- *Time Stamp, SSID, BSSID, Strength, Primary Frequency, Distance, 802.11mc* and *Security.*

So finally, there were a total of 96 .csv files (24 from each of the 4 devices).

It was noticed that the data from one of the devices was skewed with lots of repeating entries, therefore the files from that device were altogether discarded.

Solution

We have tried to implement the paper Lemic et al. (2014). It introduces a fingerprint-based pattern matching approach. The dataset is first split into training and testing sets. Data from one particular device is taken for testing (24 data frames) and the others are kept for training (48 dataframes).

For the training set, a unique fingerprint is created for every data frame. This "*fingerprint"* can be created using three algorithms:

1. Averaging the RSSI vectors
2. Quantile distribution
3. Multi-variate Gaussian Distribution

The same algorithm should be used for creating fingerprints of locations in the testing set.

Now to calculate the distance between these two fingerprints, a different metric is used for each of the three algorithms:

1. Euclidean Distance
2. Pompeiu-Hausdorff distance
3. Kullback-Leibler Divergence

Respectively.

For each fingerprint in the testing set, the fingerprint in the training set that gives the least distance is chosen and the corresponding location_label is predicted, as it *"matched"* the most.

Preprocessing

The .csv files had a lot of unrequired columns. We only needed the *BSSID*, *Strength,* and corresponding *Time Stamp.* Also, the app scanned some nearby temporary hotspots, but we wanted to keep our project limited to the routers installed in the CS-NAB only. So the dataset was cleaned and files stored in an array of dataframes. Location labels of these data frames were stored in a separate array maintaining the original order. This was done using standard python libraries.

It was found that the entire dataset had 32 unique BSSIDs. These BSSIDs were sorted so that they had a fixed order and were stored in an array. But all files did not contain all the BSSIDs. These missing values were handled differently in each of the three algorithms.

1. ***Averaging the RSSI vectors***

For this algorithm, only *BSSID* and *Strength* columns were needed. The data frame was grouped by *BSSID* and corresponding *Strength* values were averaged out. So now we had a single *Avg. Strength* value for each BSSID. But there were still some BSSIDs missing from the dataframe. *Average Strength* for these BSSIDs was initialized to a fixed practically infinite value (-10000, since RSSI values are negative), which basically indicated that this BSSID was not seen.

The fingerprint of each data frame was converted to a (1x32) size vector where 32 denotes the number of BSSIDs. The Averaged RSSI values were stored in the order of sorted BSSIDs, so as to make comparison of two different fingerprints easier.

Matching: Euclidean Distance

```
for i in range(len(test)):
  min_index = 0
  distance = np.inf
  for j in range(len(train)):
    ans = euclidean_distance(test[i][1], train[j][1])
    if(ans < distance):
      distance = ans
      min_index = j
  print(min_index)
  test_result.loc[i]['Predicted'] = train[min_index][0]
```

For each fingerprint in the test dataset, Euclidean distance with every fingerprint of the training dataset was calculated and the location_label of fingerprint that gave the least distance(most similar) was predicted.

The same algorithm was run after normalizing the dataset. There was an apprehension that since the data was collected from 4 different devices, there might be calibration issues and we may not get accurate results. So for each data frame, the RSSI values were normalized using standardization.

2. *Quantile Distribution*

The concept used here is more or less the same in the previous section, the only difference being instead of single averaged out values for each BSSID, a Quantile Distribution of those values was taken. Since the no. of observations per BSSID varied across data frames, values were sorted according to their frequency in descending order and the 30 most frequent (repeating) values were chosen. So we get a matrix of shape 30X32 for each data frame (Order of BSSIDs remains the same as they are sorted).

The columns of these matrices were then transformed into a Quantile Distribution. This method transforms the features (32 BSSIDs here) to follow a uniform or a normal distribution. Therefore, for a given feature, this transformation tends to spread out the most frequent values.
Therefore, fingerprints were in the form of (30x32) matrices following Quantile Distribution.
The algorithm was run with different numbers of quantiles - 10 , 30 and 60.

Matching: Pompeiu-Hausdorff distance (P.H. Distance)

For each fingerprint in the test dataset, P.H. distance with every fingerprint of the training dataset was calculated and the location_label of fingerprint that gave the least distance(most similar) was predicted.

```
def Loc_estimation():
  estimatedPositionArray = []
  for i in range(len(Q_test_trans)):  #for each quantile in testing set
    Q1 = Q_test_trans[i]
    estimatedPosition = 0
    reference = np.inf
    for j in range(len(Q_train_trans)):
      Q2 = Q_train_trans[j]
      d = directed_hausdorff(Q1, Q2)[0] #calculates and returns PH
distance
      if d<reference:
        reference = d
        estimatedPosition = y_train[j]
    estimatedPositionArray.append(estimatedPosition)
  return estimatedPositionArray
```

The same algorithm was run after normalizing the data using standardization and a change in results was noted.

3. Multivariate Gaussian Distribution

In this approach, every individual feature is treated as a variable and made to follow a Gaussian Distribution. Therefore the entire dataset becomes a Multi-Variate Gaussian Distribution. The combination of mean and covariance matrix form the fingerprint in this case.

Initial Approach

Data frame was grouped by *Time Stamp* and for every timestamp corresponding RSSI values for each BSSID was filled in. The data frame had a total of 60

timestamps (60 scans in runtime), therefore a matrix of shape (60x32) is formed. The matrix had a lot of missing values too, as not all the APs were detected during a scan. So, these missing values were filled with a fixed large value (-100). The mean vector is calculated after averaging out the columns and the covariance matrix is a 32x32 matrix.

But doing so made some of our covariance matrices singular, as columns in the original 60x32 matrix might be linearly dependent and therefore may have given a singular covariance matrix. The reason for them being singular was the initialization we had done for missing values. There's a high probability that more than one BSSID had all the values missing as they were not detected in the scan or the scan reported the same value for a particular BSSID. This would make the columns linearly dependent.

Final Approach

Therefore instead of initializing missing values to a fixed number, they were sampled randomly from a Gaussian Distribution with mean = -94 and std. Deviation = 3.
Mean was recalculated for all the columns and the entire dataset was sampled from a Multi-variate Gaussian Distribution with the same mean and a std. Deviation of 0.5. This was done to introduce randomness in the data set without much deviation from the original points.

The combination of mean and the newly calculated non-singular covariance matrix formed the fingerprint in this case.

Matching: Kullback-Leibler Divergence (K.L. Distance)

$$D_{\mathrm{KL}}(\mathcal{N}_0 \parallel \mathcal{N}_1) = \frac{1}{2}\left(\mathrm{tr}\left(\Sigma_1^{-1}\Sigma_0\right) + (\mu_1 - \mu_0)^{\mathsf{T}}\Sigma_1^{-1}(\mu_1 - \mu_0) - k + \ln\left(\frac{\det \Sigma_1}{\det \Sigma_0}\right)\right)$$

K.L. distance was calculated for with all the fingerprints in the training set for a fingerprint in the testing test, and the location label of fingerprint that gave the least distance was predicted.

```
for i in range(len(test)):
  position = 0
  distance = np.inf
  for j in range(len(train)):
    ans = KL_dist(test[i],train[j])
    if(ans < distance):
      distance = ans
      position = names[train[j]][:-2]
  test_result.loc[i]['Predicted'] = position
```

Dataset was not explicitly normalized as it was already following a Gaussian Distribution.

Evaluation

For evaluation, we have used 2 accuracy metrics.

1. ***Exact Accuracy***: When iterating over the testing samples after prediction, only an exact match between the original label and predicted label is considered.

   Weightage

   1: Exact match

   0: Otherwise

```
def exact_accuracy_score(predicted):
  sum_ans = 0
  for i in range(len(predicted)):
    if predicted['Original'][i] == predicted['Predicted'][i]:
#Original == Predicted exactly
      sum_ans = sum_ans + 1
    else:
      continue
  return sum_ans/(len(predicted))
```

2. ***Relaxed Accuracy:*** Here the matching rules are relaxed and if the predicted label lies adjacent to the original label then it is awarded a weightage of 0.5.

   Weightage

   1: Exact match

   0.5: Predicted Label adjacent to Original Label

   0: Otherwise

```
def relaxed_accuracy_score(df):
        sum_ans = 0
        for i in range(len(df)):
          if df['Original'][i] == df['Predicted'][i]:
      #exact match
            sum_ans = sum_ans + 1
          elif
      match_adjacent(df['Original'][i],(df['Predicted'][i])):
      #adjacent positions with lesser weights
            sum_ans = sum_ans + 0.5
          else:
            Continue
return sum_ans/(len(df))
```

Conclusion

The results obtained are as follows:

1. Averaging Vectors:

	Without Normalization	*With Normalization*
Exact Accuracy	*75%*	*83.33%*
Relaxed Accuracy	*83.33%*	*91.67%*

For obvious reasons, Relaxed Accuracy would always be greater than Exact Accuracy and in this case, Normalization further improved the accuracies.

2. Quantile Distribution:

	Without Normalisation		With Normalisation	
No. of Quantiles	Exact Accuracy	Relaxed Accuracy	Exact Accuracy	Relaxed Accuracy
10	33.33%	52.08%	33.33%	52.08%
30	41.67%	58.33%	41.67%	58.33%
60	54.16%	66.67%	54.16%	66.67%

The accuracy therefore increased with number of quantiles.

3. Multivariate Gaussian Distribution:

Exact Accuracy	*70.83%*
Relaxed Accuracy	*83.33%*

Multi-Variate Gaussian Distribution was found to have the best score for both Exact and Relaxed Accuracies. Also for the previous 2 algorithms, a common prediction error was floor mismatch, i.e. predicting a location symmetrically above or below the current floor. But for MvG Dist, these floor mismatches reduced significantly, almost disappeared.

So, we conclude that out of all the three algorithms, averaging out the vectors performed the best. This was the least complex algorithm out of all the three and yet outperformed all of them. Therefore a complex algorithm need not be the best.

Quantile Dist. < Multivariate Gaussian Dist < Averaging Vectors

These accuracies depend heavily on the choice of training and testing sets. The figures may vary if different training and testing sets are used, but the order remains the same.

Future Work

The algorithms used worked on the basic fingerprint matching approach and gave us decent results. I believe applying standard Machine Learning algorithms for Clustering and Classification, might yield better results. This project had a very limited dataset, and a larger and more holistic dataset would be needed to run these ML algorithms. Even Neural networks can be used for the same purpose, but again Neural Networks demand a very large dataset for them to perform well.

In this project, RSSI values have been used that give only a scalar value. We can use CSI received from MIMO antennas, that provide angle and the magnitude of incoming WiFi signals. Angle information would be more concise and improve results further.

The integration of the prediction model and the Android app has not been done yet and needs to be done. Once it is integrated into an app, results can be made better through crowd-sourcing as more and more data is collected from users.

References

- *Yongsen Ma, Gang Zhou, and Shuangquan Wang. 2019. WiFi Sensing with Channel State Information: A Survey. ACM Comput. Surv. 1, 1, Article 1 (January 2019), 35 pages.*
- *Filip Lemic. 2014. Benchmarking of Quantile-based Indoor Fingerprinting Algorithm. (July 2014), 43 pages.*
- *F. Lemic, A. Behboodi, V. Handziski and A. Wolisz, "Experimental decomposition of the performance of fingerprinting-based localization algorithms," 2014 International Conference on Indoor Positioning and Indoor Navigation (IPIN), Busan, 2014, pp. 355-364.*
- *Zhang, Yunfan, "An Approach to Finding Parking Space Using the CSI-based WiFi Technology" (2018). Electronic Theses and Dissertations. 2440.*
- *Application by VREM Software Development, source code here*
- *https://www.microsoft.com/en-us/research/project/radar/*