Indoor WiFi Localization with a Dense Fingerprint Model

Plamen Levchev, Chaoran Yu, Michael Krishnan, Avideh Zakhor
Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA, United States
{levchev, chaoran_yu, mkrishna, avz}@eecs.berkeley.edu

***Abstract*—WiFi-based localization is a popular approach for positioning a WiFi-enabled device in an indoor environment. Most implementations rely on querying fingerprint databases, created by stop and go sampling of WiFi signals at discrete locations used as reference points. In this paper, we propose an approach for rapid creation of a dense WiFi fingerprint database using a human operated ambulatory backpack and a single walkthrough and data collect in an indoor environment. In addition, we present and compare the performance of 4 algorithms for localizing mobile devices based on the collected fingerprints that take advantage of the dense database. We show that it is possible to achieve mean error of 2.8 meters with 90th percentile of 5.0 meters using one of our algorithms.**

Keywords—indoor localization; WiFi localization; WiFi fingerprints;

I. Introduction And Related Work

Indoor localization is an important problem in several applications including locating people or important equipment in a large indoor environment, indoor navigation, and augmented reality. One of the most common approaches to indoor localization is through WiFi. There are two major types of WiFi localization techniques: triangulation-based and fingerprint-based. Triangulation techniques use time of flight or signal strength to determine distances from known anchors in order to determine the user's location via triangulation. These techniques require knowledge of the locations of the anchors, as well as consistent signal strengths or tight time synchronization. Fingerprint-based techniques are based on an initial "sounding" phase where a human operator records information about WiFi signals in many locations throughout the building, creating a set of fingerprints. A user can then compare observed WiFi signals to the fingerprints in the database to estimate his/her location. In many applications, fingerprinting is the preferred method, because it does not have as stringent requirements on the hardware or infrastructure. However, the process of collecting the readings is an extremely arduous process and the locations in the database are typically limited to room or other landmark-level granularity [1], [2], [5].

The grid-based database approach presented in [6] combines an arbitrary number of WiFi signatures into grid cells, by saving the mean RSSI value for every AP within each grid cell. This allows for a reasonable accuracy provided that the grid spacing is small enough, and that there are enough RSSI readings per AP to represent its signal strength. Its main disadvantage is that the data collection has to be in a known environment with a predefined path. In this paper we use the database grid-based structure in conjunction with our database acquisition system as a baseline for comparison of our own localization algorithms.

One of the most successful algorithms for WiFi localization is Redpin [1] which uses a weighted sum of proximity in signal levels, presence of common access points, and difference between the mobile device's readings and the recorded readings for non-common accesspoints in the fingerprints. The comparison in [2] shows that Redpin-based approaches achieve a better accuracy than other methods. One disadvantage of the Redpin framework is that it relies on users' input and intention to participate in order to create and maintain the fingerprint database. That approach might not be suitable for all environments, and it does not allow for a database with a precise coordinate system. The process we present in this paper creates a fingerprint database in a local coordinate system and uses a modified Redpin algorithm to perform localization. Our proposed process consists of the following steps:

1. *Acquisition* - data is collected using the human operated ambulatory backpack [2]. In particular, we collect data from the laser scanners and the IMU scanner, and we capture WiFi access points beacons through 3 USB AirPcap NX WiFi cards, while the human operator walks through the building.

2. *Database preparation* - the data from the laser scanners and the IMU scanner is used for creating a precise 3D model of the indoor environment in a local coordinate system as described in [3], and the WiFi signal readings are structured into fingerprints identified by their coordinates in the same

local coordinate system. In addition we generate a floor-plan describing the walls/obstacles as described in [4].

3. *Online positioning* - A WiFi-enabled device, typically a cell phone, submits a query to the database by sending the list of access points it can detect with their respective signal strengths. Our system runs a modified Redpin algorithm to determine a location in our local coordinate system and sends the coordinates back to the device.

The remainder of the paper is organized as follows: In Section II we describe the pipeline of our system in more detail; in Section III we show experimental results of localization attempts in different environments; in Section IV we present our conclusions and describe potential future work in this area.

II. System Overview and Proposed Method

Our proposed method consists of a training phase and a testing phase. The training phase is a pipeline of two stages. The first stage is the data acquisition, in which a human operator equipped with an ambulatory backpack walks across the indoor environment. The second stage is processing the acquired data from the first stage, and building the interior model and the WiFi fingerprint database. In the testing phase, the database is tested by submitting queries from a mobile, WiFi-enabled device and attempting to position it on our coordinate system.

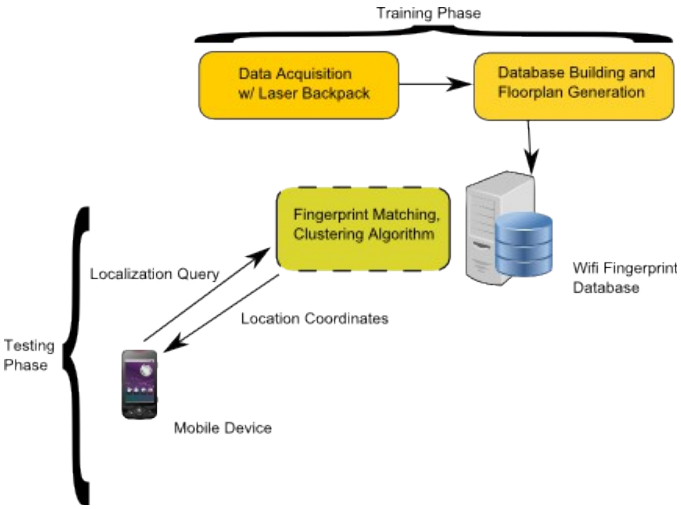


Figure 1. High-level overview of our localization method.

A. Data Acquisition

The most essential component in this process is the ambulatory backpack [2]. We use 3 USB AirPcap NX cards from Riverbed Technology connected to the backpack's laptop to capture 802.11 beacons. The AirPcap driver supports the use of up to 3 cards simultaneously in a Multi-Channel Aggregator. This allows capturing data from 3 different channels simultaneously, and switching the channel on each of the cards separately.

As the human operator walks with the backpack, the Multi-Channel Aggregator captures WiFi beacons on all supported channels in the 2.4GHz and 5GHz bands, recording the timestamp and received signal strength indicator (RSSI) of each beacon. Each card is assigned to specific channels i.e. ⅓ of the available 32 channels in U.S. and it switches only between these assigned channels. The operator walks slowly, at about 0.7 m/s, to ensure that every set of captured signals from all channels is close to one particular physical location.

The primary advantage of our system compared to the current state of the art is that the acquisition can be done during a continuous walkthrough rather than a stop and go fashion whereby the operator stops to take measurements at a variety of discrete locations. This significantly reduces the time required to generate a WiFi fingerprint database.

The minimum dwell time on each channel is 102ms, which is the standard beacon period for 802.11. In our experiments, we use dwell times of 102ms, 204ms, and 306ms in order to investigate the tradeoff between the number of received beacons and the scan duration, which affects spatial resolution of the signature database. The longer dwell times allow for multiple beacons to be received from each AP, which allows us to use the median RSSI value rather than a single sample.

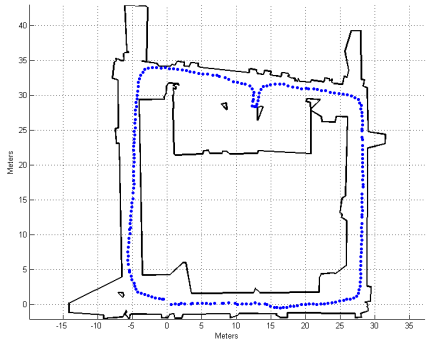
B. Database Building

The database consists of points in our local coordinate system. Each point has an associated fingerprint. A fingerprint consists of a variable-sized set of pairs of the form (AP MAC address, RSSI). This fingerprint is associated with a single location or point in the coordinate system. The MAC addresses and corresponding RSSI values are collected by the WiFi cards over time as they sweep through all the channels. As a complete sweep of all channels may take several seconds, depending on channel-switching delay, the set of MAC addresses and corresponding RSSI values does not precisely correspond to a single location.

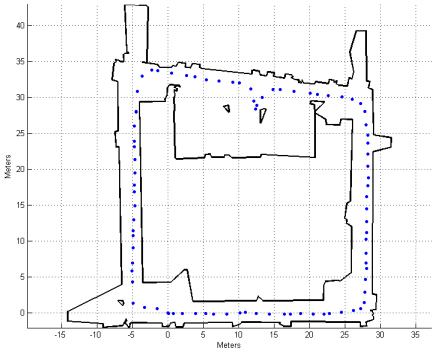
We employ two different methods for selecting database point locations and fingerprints. The first is a grid-based approach similar to that used in [6]. The locations are represented by fixed-size grid spaces of length 1-8 meters along the path walked by the operator with the backpack. At the completion of each scan, we compute the mean timestamp of all beacons, estimate which grid space the backpack system was in at that time, and associate the collected list of APs and RSSIs with that grid space. We then generate the fingerprint for each grid space by taking the union of all APs detected and associating with each MAC address the mean RSSI for that AP for all scans which included that AP.

The second method uses continuous-valued fingerprint locations. In this method, we compute the mean timestamp of all beacons used to form that fingerprint, and use the coordinates provided by the backpack system for that time instant. Consequently the density of the fingerprint map depends on the scan period length, as a shorter scan allows us to place points closer to each other.

In our experiments, we generated 12 different databases, corresponding to 1m, 2m grid, 4m grid, 8m grid, and continuous locations for dwell times of 102, 204, and 306ms. One data collection is done for each dwell time, five databases are generated for each. Figure 2 shows the set of database points for the continuous location method for the three different dwell times. It can be seen that longer dwell times result in sparser databases.



(a)



(b)

(c)

Figure 2: Floorplan and data points (fingerprint locations) represented by blue dots in our first testing environment with (a) 102 ms; (b) 204 ms, and (c) 306 ms channel dwell time.

C. *Online Positioning*

Once the database is created, a mobile device can make localization queries by scanning all channels and sending query fingerprint which consists of a set (AP MAC, RSSI) pairs to the localization server, which performs a lookup on the database and returns an estimated location. We have implemented two location estimation methods for the grid-based database, and another two for the continuous database.

The first method, called "RSSI matching" is the one used in [6]. It computes the distance to each database point as the mean squared difference in RSSI for each AP. It then returns the centroid of the three closest matches.

The second method, called "Redpin matching" uses the Redpin algorithm, which assigns a score to each point in the database that reflects the quality of the match between the query fingerprint and each database fingerprint. This score is a combination of the number of common APs, the number of non-common APs, and the differences in RSSI values for the common APs. Again the algorithm computes the three database points with the minimum distance and returns the centroid.

The first method we use for the continuous database is a clustering algorithm using Redpin scores, similar to that in [2]. The authors of [2] show that a k-nearest neighbors search with k=5 on the Redpin results with the location chosen through a majority vote produces more accurate results than standard Redpin. In our proposed method, we compute the Redpin scores for all database points, and sort them in descending order. We then build clusters starting with the highest score, adding a point to a cluster if it is within a certain fixed distance from any point in that cluster. If it is within this distance of multiple clusters, it is assigned to the one with the element with the highest score. Once a cluster of size 3 is created, we return the centroid of the cluster. The motivation behind this approach is the observation that although the closest point to the ground truth might not be among the top 5 results, there is usually a set of points close to ground truth, and consequently very close to each other, among the top results. Pseudo-code for this algorithm, which we refer to as "clustering algorithm" is shown as Algorithm 1.

The second method for the continuous database is referred to as the "median algorithm". This method also uses Redpin scores, but takes a different approach to eliminating high scores at outlier locations. As with the previous method, the Redpin score is computed between the query fingerprint and all database fingerprints. Then for each location represented in the database, we compute the median Redpin score of all fingerprints within a radius of D meters from that location and set that as the score for that location. We then take the centroid of all locations with score within a factor of (1-γ) from the maximum score and return that as the result. The values of D and γ differ depending on parameters of the database, as discussed in Section III. Pseudo-code for the median algorithm is shown in Algorithm 2.

```
parameters: thresh
scores <- new list
for each point in database
  scores.add(redpin_score(query, point))
endfor
scores.sort_descending
cluster_list <- empty list
for each point in scores:
  for each cluster in clusters:
    for each c_point in cluster:
      if dist(point, c_point) < thresh
        cluster.add(point)
        if cluster.size >=3:
          return cluster.centroid
        endif
      endif
```

```
      endfor
    endfor
    clusters.add(new cluster(point))
  endfor
```

Algorithm 1. Clustering algorithm

```
parameters: D, gamma
score <- new list
adj_scores <- new list
for each point in database
  scores = redpin_score(query, point)
  adj_scores.add(point)
endfor
for each a_point in adj_scores
  pt_list = new list
  for each point in scores
    if dist(point, a_point) < D:
      pt_list.add(point)
    endif
  endfor
  a_point.score <- pt_list.avg_score
endfor
max_score = a_point.max_score
pt_list = new list
for each point in adj_scores
  if point.score/max_score >= 1-gamma
    pt_list.add(point)
  endif
endfor
return pt_list.centroid
```

Algorithm 2. Median algorithm

III. Experimental Results

Our initial experimental environment is the doughnut-shaped hallway shown in Fig. 2 (a) through 2 (c) spanning over a 35x35m area. An operator walks with the backpack performing one full loop through the hallway. We repeat this process for dwell times of 102, 204, and 306ms, and generate 4 databases for each, corresponding to grid sizes of 1, 2, 4, and 8m, and to continuous locations.

For the testing phase we used a Samsung Galaxy S4 smartphone with Android 4.2.2 operating system. We created an Android application that retrieves the available access points and their signal strengths, and submits the set to a server. The server runs our localization algorithm on the received data and returns the resulting coordinates to the mobile device.

This process is repeated at certain known locations in the coordinate system as determined by landmarks. The locations are shown as blue dots in Figure 3. We collected 20 consecutive measurements at every location. We measured the deviation from ground truth for every set of coordinates received from the server. The Android application also saves the set of available access points and their signal strengths along with the timestamp for when the scanning is complete. This data is later used for testing the localization algorithm offline.

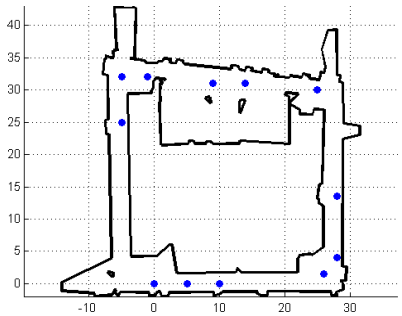
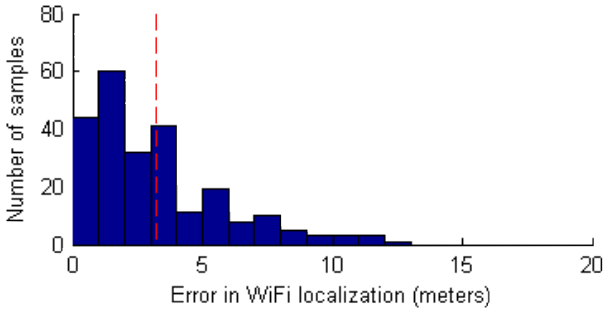
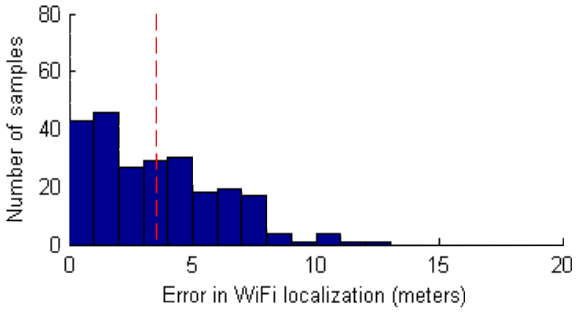


Figure 3: Ground truth locations used for measuring error in localization.

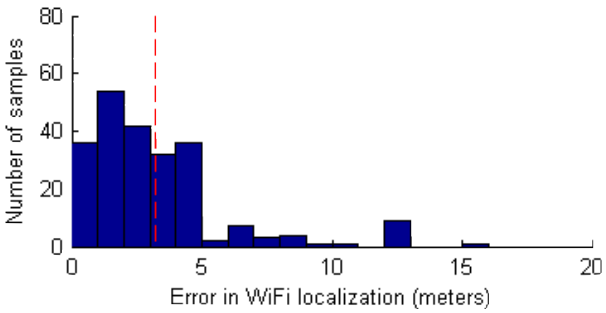
Fig. 4 (a) shows the distribution of error deviation in meters from ground truth for all localization queries for the 102ms continuous database using the clustering algorithm. The red dashed line in all the following histogram figures shows the mean error. It can be seen that the 306 ms database has a lower mean error due to a greater number of sub-2m errors.



(a)



(b)



(c)

Figure 4: Error in localization in testing against the (a) 102ms; (b) 204ms; (c) 306ms; continuous location databases

Table 1 compares the localization accuracy in terms of mean and 90 percentile error for all matching methods and dwell times. It can be seen that for both matching methods, the grid-based algorithms have the best mean error performance for 1m grid size, which is consistent with the results of [6]. It can also be seen that Redpin matching outperforms RSSI matching for 1-2m grid sizes, but performs worse for 8m grid size. Nearly all continuous matching results outperform all grid-based database results for all dwell times. The only 2 exceptions are that 102ms 1m grid with RSSI matching has a slightly lower mean error than 102 ms continuous clustering, and 204ms 4m grid has slightly lower 90 percentile error than 204ms continuous clustering. The best performance in terms of both mean and 90 percentile error for each dwell time is for the median matching algorithm. The best overall performance in terms of mean error is 2.8m with median for 102 ms, and the best in term of 90 percentile error is median for 204ms i.e. 5.3m.

One further step to increase the density of the fingerprints, and to acquire more RSSI samples per access point, is to perform 2 or more walking passes with the backpack through the environment. We have empirically found that accuracy does not improve for 3 or 4 passes.

Matching method	Grid size	102ms	204ms	306ms
RSSI Matching	1m	3.3m/8.4m	3.1m/7.0m	3.6m/7.7m
	2m	3.4m/8.4m	3.3m/6.5m	3.7m/7.7m
	4m	4.0m/8.1m	4.1m/6.5m	3.7m/8.5m
	8m	5.3m/9.3m	5.1m/7.7m	5.3m/8.5m
Redpin Matching	1m	3.8m/7.9m	3.7m/7.0m	3.9m/8.7m
	2m	3.8m/7.9m	3.7m/7.0m	4.1m/9.3m
	4m	4.7m/8.5m	4.3m/6.3m	4.7m/8.9m
	8m	5.4m/9.5m	5.5m/9.4m	6.0m/12.1m
Clustering Algorithm	continuous	3.5m/7.3m	3.2m/6.4m	3.2m/7.0m
Median Algorithm		2.8m/5.4m	2.9m/5.3m	3.2m/6.0m

Table 1: Comparison of fingerprint databases and localization algorithms (avg error/ 90 percentile error).

Table 2 compares the localization accuracy of all methods for the 2-pass databases. Again, the continuous databases outperform the grid-based ones. The best overall performance in terms of both mean and 90 percentile error is 204ms continuous with median matching, which is slightly improved compared to 1-pass. In this case, the mean error is 2.8m and 90th percentile is 5.0m. The grid based algorithms do not noticeably improve as compared to 1-pass.

Matching method	Grid size	102ms	204ms	306ms
RSSI Matching	1m	4.0m/7.8m	4.0m/8.5m	4.1m/8.6m
	2m	3.9m/9.4m	3.8m/7.4m	4.4m/8.6m
	4m	4.7m/8.9m	4.5m/8.5m	4.9m/9.7m
	8m	5.5m/10.0m	5.9m/13.0m	6.0m/10.7m
Redpin Matching	1m	4.4m/8.6m	4.1m/7.4m	4.4m/8.8m
	2m	4.2m/7.6m	4.0m/7.5m	4.8m/8.3m
	4m	4.7m/8.7m	4.3m/7.9m	5.3m/11.3m
	8m	5.7m/12.8m	8.4m/12.7m	6.1m/8.7m
Clustering Algorithm	continuous	3.6m/7.3m	3.0m/6.3m	3.4m/7.0m
Median Algorithm		3.0m/5.9m	2.8/5.0m	3.1m/6.8m

Table 2: Comparison of fingerprint databases and localization algorithms for data collected from 2 consecutive passes.

	γ=0.06	γ=0.06	γ=0.1	γ=0.12
D=2	3.4/6.3	3.2/6.3	3.0/6.3	2.9/6.1
D=3	3.2/6.4	3.0.6.3	2.8/5.6	2.8/5.6
D=4	3.2/6.2	3.0/6.2	2.9/5.6	2.9/5.4
D=6	3.0/6.2	2.9/5.6	2.8/5.4	2.9/5.4
D=8	3.0/5.6	3.0/6.0	3.0/5.6	3.1/5.6

Table 3: Comparison mean and 90 percentile error in meters for various parameter settings for the median algorithm on the 1-pass 102ms continuous database.

	γ=0.02	γ=0.03	γ=0.05	γ=0.08
D=2	3.6/8.2	3.5/8.1	3.4/7.1	3.4/7.1
D=3	3.4/7.3	3.3/6.0	3.2/6.0	3.2/6.0
D=4	3.5/7.2	3.5/7.4	3.3/7.5	3.3/7.1
D=6	3.2/7.1	3.2/7.2	3.2/7.5	3.3/7.5
D=8	3.3/7.2	3.4/7.9	3.3/7.2	3.5/7.2

Table 4: Comparison mean and maximum error in meters for various parameter settings for the median algorithm on the 1-pass 306ms continuous database.

Tables 3 and 4 show the performance of the median algorithm on the 1-pass 102ms and 306ms databases, respectively. It can be seen that the best performance for 102ms occurs for D=6, γ=0.1, and for 306ms, it occurs for D=3, γ=0.1. For 204 1-pass, the optimal is (2, 0.08), and for the 2-pass databases, optimal settings are (4, 0.1), (2, 0.1), and (4, 0.08), for the 102, 204, and 306ms databases, respectively. Determining the optimal tradeoff remains as future work.

In order to test our method in a larger environment we repeated the training phase in a different building, using only 306ms dwell time. The indoor floor plan of the environment, a hallway spanning through an approximately 100x60m area is shown in Figure 5. The dots correspond to the database locations for the continuous database with dwell time of 306ms.

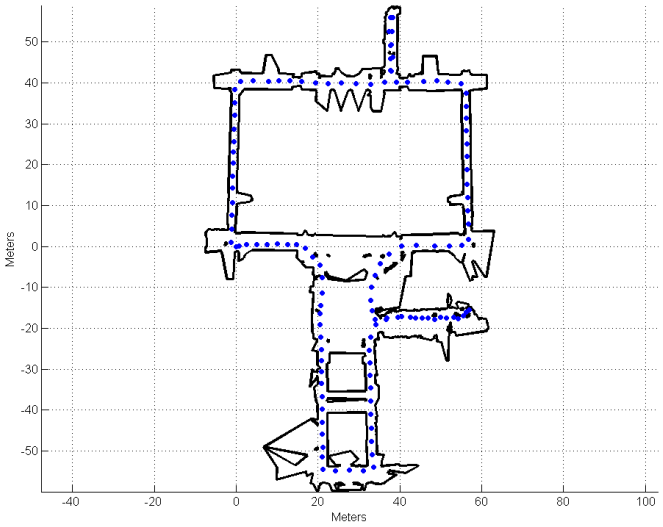


Figure 5. Floorplan of a 100x60m testing area; the blue dots show fingerprint locations.

The test data was collected from 12 different locations, identified by landmarks in the hallways. We collected 10 consecutive measurements from each location. Figure 6 shows the ground-truth locations as blue dots on the floorplan.

The error histogram from testing in the above environment for the 306ms continuous location database using our clustering algorithm is shown in Fig. 7 (a). The mean error is 6.2m, and the 90th percentile error is 14.3m.

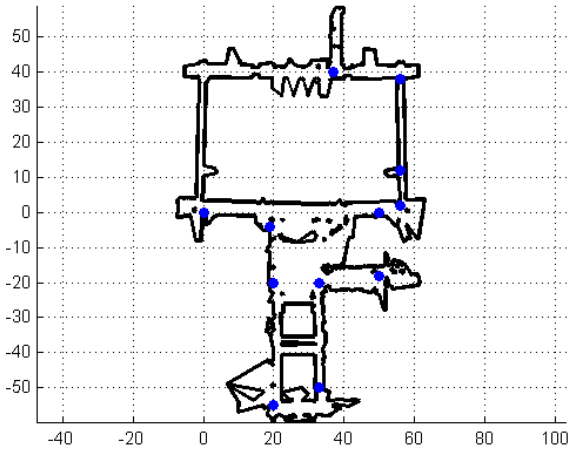


Figure 6: Ground truth locations for measuring error in localization.

In order to test our hypothesis that a dense database produces better localization results, we created sparser databases by subsampling the original one progressively and running the tests again, using the same test data saved by the Android application and running the test offline with the clustering algorithm.

The results of the experiment are shown in Table 3. There is a clear degradation of performance in terms of both mean and 90 percentile error as the density of points in the database are reduced. In Figure 7, we show the detailed error histogram for all points, 1/4, and 1/16, respectively.

	Mean Error (meters)	90th Percentile Error (meters)
All points	6.2	14.3
½ points	7.2	15.0
¼ points	9.2	17.0
1/8 points	12.5	17.8
1/16 points	26.9	43.0
1/32 points	32.2	50.3

Table 5. The effect to mean error and 90 percentile error in localization from reducing the density of the fingerprint database.

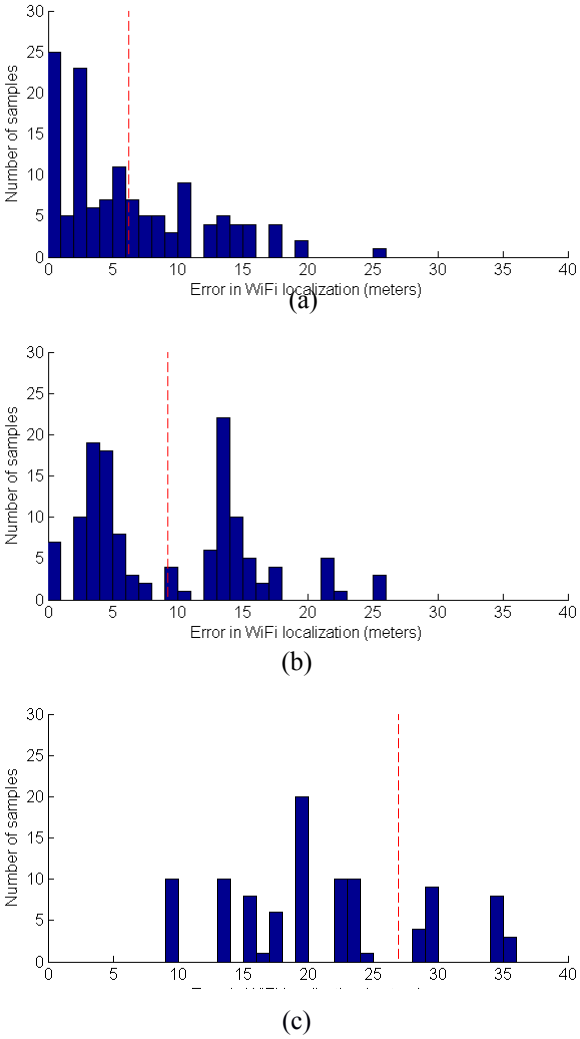


Figure 7: Error histogram for testing in the 100x60m environment with (a) all of the fingerprint location; (b) ¼ of the original fingerprint locations; (c) 1/16 of the original fingerprint locations.

IV. Conclusions And Future Work

We have presented a fast method for building a WiFi fingerprint database of an indoor environment and compared several localization algorithms that take advantage of the density of such a database. We have shown that we can achieve localization with a mean error of 2.8m and 90 percentile error of 5.0m with a database collected from a single walk through the environment with no stops, which significantly reduces fingerprint data acquisition time. We have also shown experimentally that a higher density of the database produces more accurate localization results. Future work involves developing a reliable confidence estimator for the calculated results, and fusing the WiFi localization results with other methods such as image-based localization.

V. References

[1] Bolliger, P., “Redpin - Adaptive, zero-configuration indoor localization through user collaboration.” In: ACM International Workshop, March 2008, pp. 55–60 (2008)

[2] Lin H., Zhang Y, Griss M, and Landa I “WASP: An Enhanced Indoor Locationing Algorithm for a Congested WiFi Environment”.

[3] G. Chen, J. Kua, S. Shum, N. Naikal, M. Carlberg, and A. Zakhor. "Indoor Localization Algorithms for a Human-Operated Backpack System," 3D Data Processing, Visualization, and Transmission 2010, Paris, France, May 2010.

[4] E. Turner and A. Zakhor, "Floor Plan Generation and Room Labeling of Indoor Environments from Laser Range Data," GRAPP 2014, Lisbon, Portugal, January 2014

[5] F. Evennou and F. Marx, "Advanced integration of WIFI and inertial navigation systems for indoor mobile positioning", EURASIP Journal on Applied Signal Processing archive Volume pp 164, New York, NY, United States, 01 Jan 2006

[6] Kim, J., Ji, M., Cho, Y., Lee, Y., & Park, S. (2013, October). Performance evaluation of fingerprint based location system using dynamic collection. In ICT Convergence (ICTC), 2013 International Conference on (pp. 950-954). IEEE.