Localization with Bayesian and Particle Filter

Group 9 - Xiaomi Redmi 5A (Android 7.1.2, API 25)

Sonnya Dellarosa
4783344

Pranjal Rajput
4788087

I. Bayesian Localization

A. Data collection

We collected the RSSI information of the available access points in each of the 16 cells mapped in the provided floor plan of the 5th floor of the EWI building. We collected 3 samples by performing a Wi-Fi scan with a time interval of 300 ms between each scan.

Date	Sample size per cell	Condition
21-05-2019	100	Empty
29-05-2019	100	Crowded
14-06-2019	200	A few people

TABLE I: Collected samples

The three set of samples were collected in three different scenarios: (1) empty floor, (2) crowded floor, and (3) a few people walking around the floor.

B. Data processing

We performed an offline analysis of the collected data to filter out all the access points besides *eduroam* or *tudelft-dastud*, as these two can be considered permanent access points. We, then, split the collected samples for training (75%) and testing purposes (25%). The testing samples are then put into groups of 5 such that there are at least 5 tests of 5 iterations. We trimmed the outliers in the training data by calculating their z-score values and removing those with a value above the threshold, which we set as 2.

C. Convergence

To evaluate the Bayes probability of each cell, we performed the calculation both in a serial and a parallel manner. One press of the *Locate Me* button is one iteration in the calculation. Repeatedly pressing the button will result in a serial calculation of the probability. Inside one iteration, we go through all the access points (AP) in the scan results, and calculate the probability of each AP and normalize the values at the end of the iteration.

D. Evaluation

We tested our application online for all the cells. Each cell is tested 10 times for 10 iterations. The confusion matrix is shown in the appendix.

E. Novelty

In order to improve the accuracy of the Bayesian localization, we implemented a **Bayesian motion model**. For this, we implemented a step and direction which is detailed in Section II.A.

*1) **Update frequency**:* The probability distribution of all the cells is updated every time a distance of 4 m is covered.

*2) **Motion model**:* The following is the conditional probability that we used to update the probability:

$P(X_i \mid X_i)$	0.1
$P(X_{i+1} \mid X_i)$	0.8
$P(X_{i+2} \mid X_i)$	0.1

TABLE II: Probability Distribution for cells X_i, X_{i+1}, X_{i+2}

F. Discussions

- **Similar PMFs**: Where the cells are small and next to each other, the PMF of the cells might be very similar and indistinguishable such that the application is not able to detect the correct cell. An example of this case can be seen in Figure 1 (a).
- **Fading effect**: When the floor is crowded, the probability of detecting the correct cell is observed to be a lot less accurate than when the floor is empty. Our data from scenario 1 gives an accuracy of 0.99 during testing while our data from the scenario 2 produces a much lower accuracy of 0.79. This may be due to multi-path fading, i.e. absorption effect, which causes the RSSI values to fluctuate.
- **Zero probability**: At first, we implemented our application such that whenever the PMF does not exist for the scanned AP, then the perception model probability used is 0. However, this means that if the first scan wrongly puts a probability of 0 to the true cell, this value will be in the prior probability for the next iteration and will stay at 0 in consequent iterations. The application would never be able to converge to the true cell. For this reason, we changed our implementation where we instead of 0 we use a small value ϵ. After trying different values, $\epsilon = 0.01$ seems to provide the best result.

II. Particle Filters Localization

A. Method

The implementation of the step and direction detectors are adapted from a code that we found online [1]. The following is a description of how they work.

1) ***Step detector****:* The number of steps is calculated using the x, y, and z values of the accelerometer. A set of 50 values are recorded for each axis, of which the average x_{avg}, y_{avg}, and z_{avg} are calculated. These average values are then normalized to x_n, y_n, and z_n using a normalization factor $k = RMS(x_{avg}, y_{avg}, z_{avg})$. These values are used to then estimate the user's velocity. Every 0.25 second, we obtain the accelerometer values x_t, y_t, and z_t. Multiplying each of these values to x_n, y_n, and z_n respectively, then summing them and subtracting it by k result in the estimated velocity $\widetilde{v}$, where $\widetilde{v} > 10$ means a step was taken.

2) ***Walking distance****:* The stride length is first calculated using the height of the person and adding a noise constant to account for uncertainties: $height * 0.413 - noise$, where the constant value of 0.413 was taken from one of the lecture's slides. To calculate for the noise constant, we performed a small training by taking 10 samples and calculate the error in the distance. Multiplying the number of steps with the stride length gives us the walking distance.

3) ***Direction detector****:* To detect the direction the phone is pointing at, we calculated a rotation matrix using Android's API that takes the accelerometer and magnetometer data as input and outputs an azimuth value. Azimuth is the value of rotation around an axis perpendicular to the mobile screen. We convert this value to degrees, and use that to find the direction.

B. Implementation of Map

The map was implemented using Android's `Canvas` and `Drawable` classes. The `Canvas` is held on a separate work thread to allow for a constant update on the screen. The mapping from distance to pixel is calculated programmatically, according to the dimension of a phone where the application is installed, thus the mapping is 1 to $W/40$, where W is the screen width divided by the width of the map (40 m). The map is drawn in accurate scale through multiple `Drawable` objects.

C. Particle Filter Implementation

1) ***Particle movement****:* When a step is detected, every particle will make very small movements of p_m pixels for n times, where p_m is a small constant and $n = stepsize/p_m$. We have implemented it in such a way after realizing that moving the particles directly according to the step size will result in grid-like distribution of particles, which is undesirable. We used $p_m = 2$.

2) ***Movement constraints****:* A particle violates the movement constraint if it fulfills any of the following conditions: (a) the pixel boundary of the particle intersects with the boundary of any wall; or (b) the pixel is located outside of the map.

3) ***Resampling****:* When the particle violates the constraints, it is immediately resampled as it gets placed on top of a random surviving particle.

4) ***Determining location****:* The location of the user is determined by calculating the particle density of every single cell. The cell with the highest density is the location of the user.

D. Results

Following a unique path results in a convergence of the particle filter as the particles end up forming a cloud, as long as only proper 90° turns are taken while walking. After convergence, we went to every cell and the application was able to detect the correct cell location.

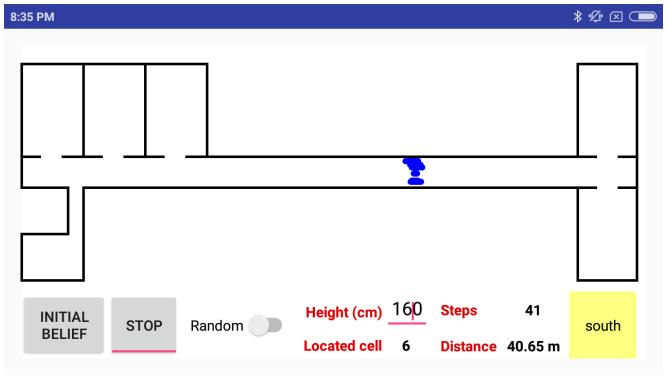


Fig. 1: Screenshot of the Particle Filter application

III. Individual workload

	Sonnya	Pranjal
Bayes GUI		x
Bayes training	x	
Bayes calculation	x	
Bayes motion model		x
PF map	x	
PF implementation	x	
PF motion model		x

TABLE III: Workload distribution

IV. Attendance and Challenges

We attended all lectures except for the last one. We did not go to any of the labs except for the last one, in order to get some feedback on our applications. The following is a list of the most challenging parts of the process.

- **Offline analysis vs online testing**: Even though we obtained very high accuracy in our offline analysis (at least 90% on all sample sets), this was not reflected in our online testing. It may be due to the RSSI values that vary a lot over a period of time, depending on the state of the surroundings and environmental conditions.
- **Imperfect sensor values**: The direction calculated is relatively accurate in normal conditions. However, due to higher magnetic field, sometimes the detected direction is inaccurate and thereby affects the motion model. Additionally, the accelerometer values are quite noisy, causing some false positives in the step detection.

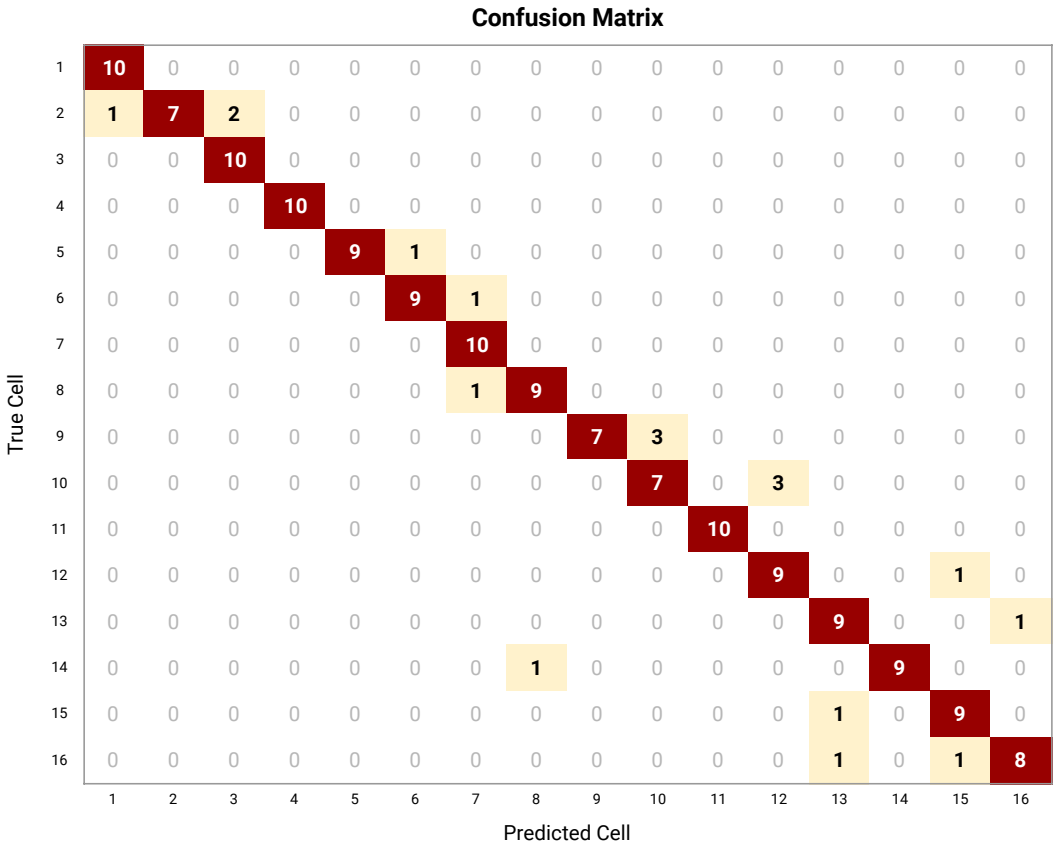
V. Possible future directions

- **More accurate directions**: in our current implementation, we only detect the 4 main compass directions. If we had more time, we would have implemented all 8 directions.

References

[1] ANU S PILLAI. *Create pedometer and step counter for android*. http://www.gadgetsaint.com/android/create-pedometer-step-counter-android/#.XQvdKOgzZPY. Accessed on 2019-05-30. Mar. 2017.

APPENDIX


Confusion Matrix
True Cell
Predicted Cell
1 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 1 7 2 0 0 0 0 0 0 0 0 0 0 0 0 0
3 0 0 10 0 0 0 0 0 0 0 0 0 0 0 0 0
4 0 0 0 10 0 0 0 0 0 0 0 0 0 0 0 0
5 0 0 0 0 9 1 0 0 0 0 0 0 0 0 0 0
6 0 0 0 0 0 9 1 0 0 0 0 0 0 0 0 0
7 0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0
8 0 0 0 0 0 0 1 9 0 0 0 0 0 0 0 0
9 0 0 0 0 0 0 0 0 7 3 0 0 0 0 0 0
10 0 0 0 0 0 0 0 0 0 7 0 3 0 0 0 0
11 0 0 0 0 0 0 0 0 0 0 10 0 0 0 0 0
12 0 0 0 0 0 0 0 0 0 0 0 9 0 0 1 0
13 0 0 0 0 0 0 0 0 0 0 0 0 9 0 0 1
14 0 0 0 0 0 0 0 1 0 0 0 0 0 9 0 0
15 0 0 0 0 0 0 0 0 0 0 0 0 1 0 9 0
16 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 8
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16