Smart Phone Sensing - Report 3

Group: 7, SmartPhone: OnePlus 6T, Android Ver: 10, Git Branch: main

Minas Melas
NetID: mmelas
Student No.: 5397324
Email: M.Melas@student.tudelft.nl

Nishad Mandlik
NetID: nmandlik
Student No.: 5242541
Email: N.G.Mandlik@student.tudelft.nl

I. Bayes Localization

A. Improvements

We fixed some bugs which resulted in the total probability being less than 1, under certain edge conditions. Secondly, we implemented the serial approach for the Bayes, but unfortunately it performed worse than the parallel approach which we originally had. We consider the most probable cell valid only if it is reported with a confidence greater than 15%. Otherwise, the position is considered to be unknown. In case of the parallel approach, occurrences of unknown positions were found to be quite rare. However, in the serial approach, such events occurred frequently. In the serial approach, an incorrect low posterior value affects the posterior values for the corresponding cell in all subsequent stages. This reduces the accuracy of inference.

The training was conducted on 07 June 2021, and testing was performed on 15 June 2021. It can be seen from Table II and Table I, that the accuracy is 67.5% and 55% for parallel and serial respectively. This also shows that the accuracy decreases as the time between training and testing increases. Previously, an accuracy of 78.75% was achieved, when testing was conducted immediately after training.

TABLE I: Confusion Matrix for the parallel approach. (U: Unknown)

Actual \ Predicted	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**U**
1	7	0	2	0	0	1	0	0	0
2	1	8	0	1	0	0	0	0	0
3	0	1	6	2	0	1	0	0	0
4	2	0	1	4	0	3	0	0	0
5	0	0	0	0	4	5	0	0	1
6	1	0	1	0	0	8	0	0	0
7	0	0	0	0	0	0	7	3	0
8	0	0	0	0	0	0	0	10	0

TABLE II: Confusion Matrix for the serial approach. (U: Unknown)

Actual \ Predicted	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**U**
1	4	4	0	0	0	0	0	0	2
2	1	9	0	0	0	0	0	0	0
3	0	2	6	0	2	0	0	0	0
4	0	3	0	7	0	0	0	0	0
5	0	0	0	0	6	0	0	0	2
6	0	0	0	0	1	3	0	0	6
7	0	0	0	0	0	0	6	2	2
8	0	0	0	0	0	0	2	1	7

B. Novelties

- Inclusion of signal strength of mobile network for computing the posteriors.
- Filtering APs based on vendors, strength and number of occurrences.
- Interpolation using averaging values in the neighbourhood instead of Gaussian distribution (for ease of implementation).
- Calculating the best range for the neighbourhood, via cross validation.

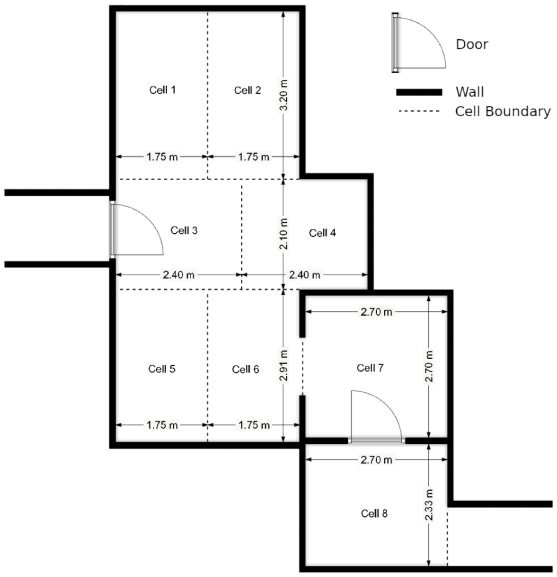

Fig. 1: Home layout

II. Particle Filter Localization

A. Motion model

For calculating the motion of the user, we initially tried Android's built-in step detector. However, this composite sensor is designed for calculating steps for long distances, and in order to cope with noise, it only starts calculating steps after it detects movement for several seconds. Thus, we implemented our own step counter based on paper [1]. The method suggested by the paper performs auto-correlation based on the resultant acceleration values. We divide the sample window into smaller 'sub-windows', and compute the correlation coefficient for successive sub-windows. The process is repeated for a range of sub-window sizes, in order to find the maximum correlation value (mac). The corresponding size (τ_{opt}) is chosen as the number of samples per step. The state of the user (walking or idle) is determined by analysing the maximum correlation value as well as the standard deviation of the acceleration values. Nevertheless, the constants suggested in the paper were not accurate for our application, so after experimenting we found that the most suited conditions were the ones shown in Figure 2.

```
if std < 0.02 * mean then
    state ← stationary
else if mac > 0.7 and not walking then
    state ← stationary
else
    state ← walking
end if
```

Fig. 2: Find motion state of user

We calculate the step count according to paper [1] and the stride length according to the empirical model depicted in paper [2]. The equations of these are as follows:

$$Step\ Count = \frac{window\ sample\ size}{\tau_{opt}}$$

$$Stride\ Length = K \times \sqrt[4]{Acc_{vmax} - Acc_{vmin}}$$

Acc_{vmax} and Acc_{vmin} are the maximum and minimum acceleration forces along the y axis throughout the window and K is personalized parameter.

Finally, we used the magnetic field and accelerometer sensors, and Android's `getRotationMatrix` function in order to find the azimuthal angle.

B. Implementation of map

We used Android API's `Canvas` class in order to draw the particles on screen. After the "Start" button is pressed, we get the *top*, *left*, *right* and *bottom* parameters of the 'inner' layout which holds the `TextViews` of the cells. We also create an instance of Android API's `Bitmap` class, containing the drawing of the whole screen, which we call the 'outer' layout. All drawings will take place on top of this bitmap. Afterwards, we compute the horizontal and vertical scale factors that will be used for scaling the particles from meters to pixels. For that, we divide the width and height of the 'inner' layout (in pixels) with the corresponding physical dimensions (in meters).

Every time we want to make a new drawing, we create a new bitmap similar to the first one. Then, we create a new canvas instance and draw the points by first scaling (multiplying horizontal and vertical values with horizontal and vertical factors respectively) and then transforming (adding left for horizontal value and top for vertical value) in order to get the real position in pixels in the layout.

C. Particle filter implementation

Having the step count and the step length, we can calculate the total distance traveled throughout a window. After testing, we found that the best window size is 2 seconds with a sampling period of 20 milliseconds. According to the Shannon Sampling Theorem, the sampling frequency should be more than two times the maximum walking frequency. We assume that the maximum walking frequency will be 2Hz (2 steps per second), so our sampling frequency $\frac{1}{0.02sec} = 50$Hz satisfies this condition. Therefore, each window will contain 100 values which are the resultant magnitudes of the components reported by the accelerometer sensor.

Upon completion of each window, the algorithm proceeds in the following manner:

1) Detect if the state is walking and then calculate the distance covered during the window with the method described in II-A. Afterwards, update the positions of the particles by adding a Gaussian noise to both the angle and direction. This noise is shown in Figure 4. We specified a standard deviation of 22.5° for the angle and $0.1m$ for the distance and a mean value equal to the new calculated angle and distance. Thus, the Gaussian

```
for i ∈ dead particles do
    prob ← random(0, 1)
    ptotal ← 0
    for j ∈ alive particles do
        ptotal = ptotal + w_j
        if prob <= ptotal then
            i ← j
            break
        end if
    end for
end for
```

Fig. 3: Re-assign dead particles

distribution ensures that there is a very low probability of being outside a 45° angle and a very high probability of having an angle and a distance close to the measured values.

2) After calculating the new positions that include the Gaussian noises, 'kill' all the particles that are not inside any cell and the particles that have intersected at least one wall, by setting their weight equal to 0. The former condition is trivial. The second condition is derived from our assumption that all the values used for the new distance will be derived from a window where the user was moving forward and the user cannot go through any objects. In order to find the particles that intersect a wall, we consider the wall and the particle's motion as infinite lines and calculate the intersection point (if any). Then, we check if this point lies within the segments that define the wall and the user's motion.

3) Increase the weight of each alive particle according to how many rounds it is alive. This condition makes sure that the dead particles will have a higher probability of being re-assigned near particles that have had made more correct predictions in the past. After calculating the new weights, normalize them in order to have weights that represent probabilities.

4) Update the positions of the dead particles in one pass as shown in Figure 3. Each dead particle picks a random variable ($prob$) and repeatedly tries to find the best alive particle candidate with weight w_j.

5) Assign a dead particle to an alive particle, by also adding a small noise equal to a random value in the range $[0, std_{distance})$ where $std_{distance}$ is the standard deviation that we also used for the Gaussian noise distance. We included this noise, because we observed that the particles tend to stack in one place, as when they died they were all allocated on the same positions of other alive particles. This would be beneficial if we had a perfect motion model, but since we expect inaccuracies from it, this could lead to higher chances of all particles dying in the same round.

6) Normalize the weights again so that they all have values in the range of (0, 1). By doing so, the rounds that particle has been alive will have a bigger impact on the new particle's weight.

7) Calculate the most probable cell that the user currently stands with respect to the density of the particles. To do so, calculate the particles that are inside each cell and choose the cell that contains the majority of particles.

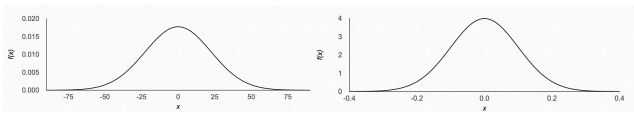

Fig. 4: Gaussian noise distributions for angle (left) and distance (right)

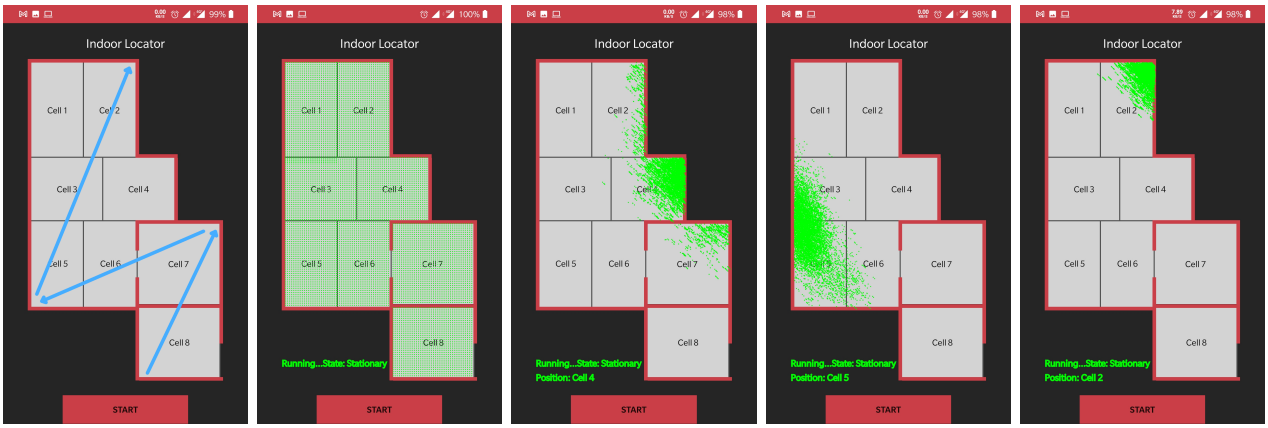

Fig. 5: Screenshots of Particle Convergence (left: Convergence Path, right: Stages of Convergence)

D. Results

The convergence path and the convergence stages are shown in Figure 5. We observed that after convergence, the accuracy depends on the path followed. We tested two paths:

- Cell 2 → Cell 3 → Cell 6 → Cell 1 → Cell 4 → Cell 5 → Cell 7 → Cell 8
- Cell 2 → Cell 6 → Cell 4 → Cell 1 → Cell 3 → Cell 5 → Cell 7 → Cell 8

In the first one the particles followed the user correctly during the entire motion. In the second one, the motion from Cell 1 to Cell 3 was not detected, and the system displays incorrect results until convergence is performed again. Further testing showed that the algorithm is not able to detect a small motion, for instance between the centers of Cells 1 and 2, which are only $1.75m$ apart. The window size of 2 seconds is too large for this distance.

III. Novelty for Particle Filter

- Developed motion sensor based on paper [1] to calculate step count and detect motion state.
- Included dynamic step length based on paper. [2]
- Include higher weights for longer survived particles and zero weights for particles that pass through walls.
- Calculate best alive particle candidate in one pass for higher processing performance (programming).
- Add noise to the particle's position when re-assigning.
- Dynamic Canvas scaling and positioning, according to the phone's screen (programming).
- Window interpolation. Initially we used non-uniform sampling, and thus interpolation was used to handle irregularly spaced samples during correlation. This was removed afterwards, when we switched to uniform sampling.

References

[1] Xiang He, Jia Li, and Daniel Aloi. "WiFi based indoor localization with adaptive motion model using smartphone motion sensors". In: *2014 International Conference on Connected Vehicles and Expo (ICCVE)*. ISSN: 2378-1297. Nov. 2014, pp. 786–791. DOI: 10.1109/ICCVE.2014.7297659.

[2] Haifeng Xing et al. *Pedestrian Stride Length Estimation from IMU Measurements and ANN Based Algorithm*. en. Research Article. Feb. 2017. DOI: https://doi.org/10.1155/2017/6091261. URL: https://www.hindawi.com/journals/js/2017/6091261/ (visited on 06/16/2021).

IV. Individual Workload

	Minas	Nishad
Bayes - AP Filtering		✓
Bayes - Cross Validation	✓	
Bayes - Data Collection	✓	✓
Bayes - Data Export		✓
Bayes - Mobile Network Scanning		✓
Bayes - Probability Interpolation	✓	
Bayes - Serial & Parallel		✓
Bayes - UI		✓
PF - Idle/Walking Detection	✓	✓
PF - Autocorrelation	✓	✓
PF - Canvas	✓	
PF - Constraint Conditions		✓
PF - Particle Movement	✓	
PF - Sensor Reading	✓	
PF - Stride Calculation	✓	