Mobile Computing, Lab

Hannah Brunner, Markus Gallacher

June 21, 2019

1 Introduction

The aim of the course was to implement a smartphone app on Android, which utilizes and processes on-phone sensor data. Two assignments had to be delivered:

The first task was an activity monitoring application using the KNN algorithm. Accelerometer data is sensed for a given time window and features such as mean or min/max value are extracted. These values can then be compared to prior recorded and labeled samples to derive the current activity. Further details are explained in Section 2.2.

Second, an indoor localization application using a particle filter had to be implemented. Therefor, a map of the ITI buildings' second floor was given. For localization, a number of particles is spread on the map in random manner and moved according the users' walking direction. The filtering of unlikely paths respectively particles should reveal the users' current position. The algorithm is described in Section 2.3.

The source code of the app is available on GitHub [1].

2 Implementation

2.1 Main App

We combined both tasks (activity monitoring and localization) into a single app. It consists of four activities:

1. **MainActivity:** Shows the home screen, which appears at start of the app (see Figure 1) and contains buttons to start the desired activity.

2. **TrainActivity:** Records and stores accelerometer data associated with desired activity, which is used during activity monitoring (see Section 2.2.2).

3. **MonitorActivity:** Monitors and displays the user's current activity (see Section 2.2.3).

4. **LocalizationActivity:** Implementation of indoor localization using particle filter (see Section 2.3).

Figure 1: Home screen with three buttons to select desired activity.

2.2 Activity Monitoring

The Activity Monitoring is split into two activities, one collects training data and stores it into a file locally. The other monitors the users activity and uses the kNN algorithm to classify the user's motion. At the moment we detect Idling (doing nothing), walking, standing up and sitting down.

2.2.1 Background: Activity monitoring using KNN

kNN is a very simple and fast algorithm. It tries to cluster points into groups by looking at the k nearest neighbours. This is done by calculating the euclidean distance between a new measurement point to all other points and then taking the k nearest ones.
The **classification** is normally a simple majority vote of the neighbours labels. The new data point will receive the label that the majority of the k nearest neighbours have. We have modified this slightly as can be seen in 2.2.3.
In order to monitor the activity via kNN, accelerometer data has to be recorded for a desired time window and a feature set has to be chosen. Each feature is used in the calculate of the euclidean distance which relates to comparing the similarities of the new measurement (received from 2.2.3) point to the known data points (trained by 2.2.2). The features can be varied and are up to the developer.

2.2.2 Train Activity

This activity enables the user to collect training data for the monitoring activity. 1 of 4 activities are currently implemented, however any activity can be trained, only the label would mismatch at the moment.
Our chosen feature set contains:

- Timestamp of the recording.
- Mean of x values.
- Mean of y values.
- Mean of z values.
- Maximum x value.
- Maximum y value.
- Average of the 3 most dominant frequencies (this feature is not actively used in the kNN as it decreased the classification accuracy).

The user needs to press one of the activities and perform the motion. The accelerometer data is sampled 60 times with the android setting SENSOR DELAY GAME and the activity label and the features are stored in a local .txt file. If a faulty measurement was recorded the user can delete the last entry or delete the whole .txt file, which is stored in a csv format.

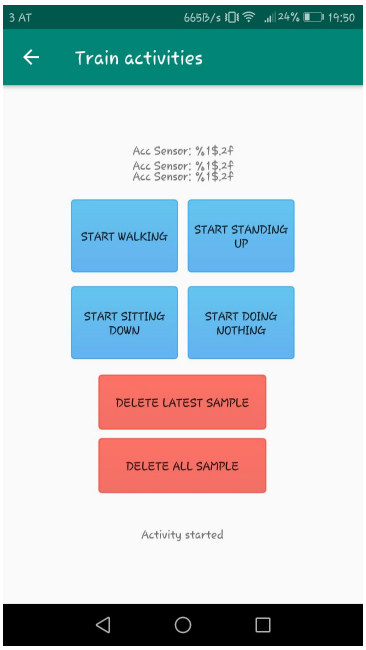

Figure 2: Train Activity. Three text boxes showing the measured accelerometer data. Four buttons to each start training a data set. Two Buttons to delete either the last or all training sets.

2.2.3 Monitoring

The second activity tries to classify the activity currently performed by the user. When the button is pressed, the accelerometer data is sampled with the same window size as the training data and the same features are extracted from the measurements as were in the training data. The matched activity is displayed together with the probabilities for each of our 4 predefined activities. When the "continuous monitoring" box is ticked, this prediction process (sample, kNN, classify) is repeated endlessly.

kNN:
The kNN algorithm as described in 2.2.1 finds the k nearest neighbours of our new features. We used $k = 3$.

Modified classification:
Our classification is a modification of the simple classification mentioned in 2.2.1 by weighting all neighbours with $\frac{1}{\text{euclidean distance}}$ and adding two weights if neighbours have the same label. The higher the weight of a label, the smaller the euclidean distance is between feature points or the more neighbours with the same label are found. When calculating $\frac{\text{weight}}{\text{total weights}}$ one receives the probabilty that the new data point matches one of its neighbours.

Justification for altering the classical classification:
We do this because our relatively small training set can have large gaps between neighbours, therefore a really close neighbour can be prioritised over two very distant neighbours if it has a higher weight. With this method we achieved a higher accuracy.

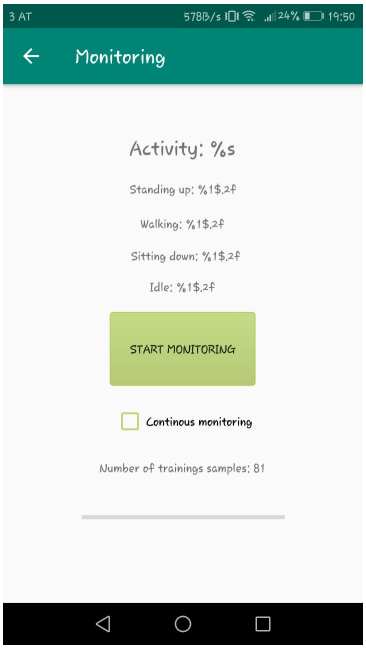

Figure 3: Montoring Activity. Four text boxes showing the probability of each label being classified. A start button to collect sensor data. A tick box to perform continuous monitoring.

2.2.4 Limitation/Challenges

A big challenge was to reach a high accuracy of correct activity predictions with the motions standing up and sitting down as the motions are very similar. Furthermore our training data is self-obtained, meaning that it is comparatively small, making it more difficult to reach a highly precise prediction.

Python script:
We have made a python script which calculates the accuracy of our algorithm by checking how many predictions are actually true. It achieves around 80%. The accuracy would be higher if we tried to predict more distinguishable motions rather than standing up and sitting down. For example running or jumping. Moreover, the accuracy is highly dependent on the size and quality of the trainings set.

2.3 Localization

2.3.1 Background: Particle Filter

Particle filters are a rather easy and very suitable approach for indoor localization. The basic idea is to spread a high number of particles randomly in the environment. If the user moves, the particles move in the same direction. Particles, which violate physical constraints (e.g. walk into a wall) are deleted and thus, only a set of possible particles will survive to represent the user's location.
Localization using a particle filter is an iterative process consisting of the following steps:

1. **Spread N particles:** An arbitrary, but high number (5000-1000) of particles are spread on the map.

2. **Move:** If the user moves, the same movement is applied on all particles. To compensate for sensing errors, a variance in movements length and direction is added.

3. **Sense:** It is checked, if physical constraints are violated (particles walking through a wall). If so, the particles' weight (likelihood) is set to zero.

4. **Resample:** To avoid particle depletion, particles are regenerated each cycle. Thus, the number of particles stays constant. Resampling is done by replacement. Particles with weight equal to zero will disappear, while particles with weight greater than zero might have multiple copies.

5. **Compute position:** The position can be obtained by finding the point with the highest particle density.

The repitition of steps 2 to 5 will eventually lead to convergence and reveal the current position.

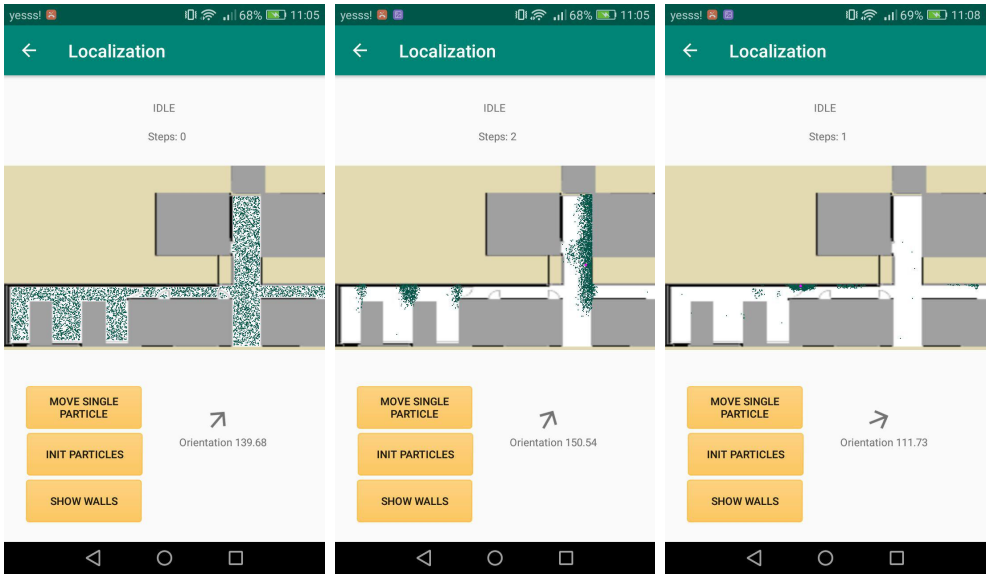

Figure 4: UI of localization activity. 1) Main screen after initializing. 2) After a few steps. 3) After convergence.

2.3.2 Implementation

GUI

As shown in Figure 4, the localization activity displays a map of the ITI building's first floor, text fields to report the users movement, an arrow indicating the user's current orientation and three buttons to

- move a single particle
- init particles and start localization
- show the walls of the building on the map (mainly debug purposes)

Movement detection

In order to detect the user's movement to create an appropriate movement model, we utilized the smartphone's built-in accelerometer and magnetic sensor. The movement detection is implemented in a background service, similar to the implementation of 2.2.3.
The user's orientation is retrieved using the Android method *getOrientation*, which utilizes the acceloremeter and magentic sensor. Since the value is quite inaccurate, the median value of the measurement while walking is used.
To detect if the user started respectively stopped walking, we record 40 samples of accelerometer data (sample rate = SENSOR DELAY GAME) periodically and extract the standard deviation and autocorrelation, as described in the lecture notes [2]. For appropriate results and correct distinctin between walking and idle we had to adjust the threshold values.
The travelled distance was approximated by measuring the walking time and assuming a constant walking speed of the user.

Particle Filter

In the following, we will give some details about the implementation of steps 1-5, explained in 2.3.1

1. **Spread N particles:** We chose N = 6000, which gave a reasonable tradeoff between computation time and accuracy.

2. **Move:** The movement is sensed as explained in 2.3.2 and we add a variance of ±20cm for step width and ±20° for the orientation angle.

3. **Sense:** To check if particles moved out of the valid area, we introduced walls in a hardcoded manner. We then compute the line between the last position and the newly computed position. If the lines of the wall and movement intersect, a violation of the physical constraints is detected.
   The initial idea with wall pixels turned out to be too slow for real-time computing.

4. **Resample:** For resampling, an easy approach was implemented. Each deleted particle is replaced by a randomly chosen valid one.

5. **Compute position:** To compute the final position, we calculate the median of x and y coordinates indepentendly.

The filter process is performed in an AsyncTask in order to prevent a working overload of the UI thread.

2.3.3 Limitations/Challenges

The main challenge was the inaccuracy of the orienatation measurements. During the debugging process, we discovered big problems when passing by metal doors or walls. Outdoors, it was much more accurate.

References

[1] https://github.com/hanuka24/ActMonitoringLocalizationApp.git.

[2] https://github.com/osaukh/mobile_computing_lab/.

The rest of theoratical content is based on the lecture notes [2]. Other ressources related to the implementation can be found in the source code.