Sensor Networks: Signal Propagation Model and WiFi fingerprinting

Chris Xiaoxuan Lu, Peijun Zhao, Andrew Markham

29/01/2019

1 Objectives

This lab consists of three parts. In the first part of the lab, each group will explore how WiFi signal strength may be affected by environment features e.g. walls, people, orientation of device, distance, etc. During the second part of the lab, each group will complete the code skeleton of *Horus* fingerprint systems on two synthetic datasets of WiFi RSSI. Lastly, you will implement your own fingerprinting algorithm on a real-world dataset of WiFi signals and test it via our online submission system. Please download/clone the GitHub repository (`https://github.com/ChristopherLu/aims_lab`) to start this lab.

Please send the following to xiaoxuan.lu@cs.ox.ac.uk by **Monday, February 4th**: (1) a group report. (2) the source code of your fingerprinting algorithm. In the meantime, please take a screenshot of the testing result after running the *submission.py script*, and paste it in the report.

1.1 Python environment

The python script you will be using in this lab requires Python version 3.5 along with some specific packages. We recommend using Anaconda for managing these dependencies, but feel free to use whatever works best for you.

If you choose to use Anaconda, please install the version of Anaconda that best suits your needs from the link above and then follow the instructions below:

```
# Create conda environment
conda create -n ips python=3.5
# Activate environment
source activate ips
# Install dependencies
conda install numpy matplotlib scipy statsmodels \
requests
```

2 WiFi signal

2.1 Signal variability

Use one of the available mobile phones, along with the pre-installed Wifi Analyser app, to measure the Received Signal Strength Indicator (RSSI) of packets sent by an Access Point (AP). Explore how signal strength may be affected by environment features e.g. walls, people, orientation of device, distance, etc. **Explain your findings in the report**.

2.2 Path loss

Obtain RSSI data at different distances (1 m, 1.5m, 2m, ...) away from the AP. Plot log (distance) on x-axis and RSSI on y-axis and fit a log-distance path loss model. **Include the experimental distance-RSSI figure in the report and compare it with a theoretical one. Explain potential reasons for any deviations observed.**

3 *Horus* on Synthetic Data

Horus is an effective WiFi based fingerprinting system. In this experiment, you will understand its working principle by completing **fingerprinting_skeleton.py**.

3.1 Synthetic Datasets

You were given two datasets, namely **set1** and **set2**, containing synthetic WiFi RSSI data for 3 and 5 access points (APs), respectively. Inside each set you will find a **APloc.txt** file, which gives the ground truth positions for all APs in that dataset. For each dataset and inside the **wifiData** folder you will find a number of **Wifi_xxx.txt** files, each of which containing the (x, y) position where the measurements were taken (first line) followed by a series of RSSI measurements.

The **testWifiData** folder, which has the same structure as the **wifiData** folder, contains only RSSI data and will be used for testing your positioning system.

3.2 Useful Functions

Several useful functions are there to help you understand the *Horus*. Specifically,

1. Data Load: The function **load_wifi_data(data_folder,n_samples,n_ap)** takes as input the data-folder (i.e. set1 or set2), the number of samples per location (i.e. 60), and the number of access points (i.e. 3 for set1 or 5 for set2) and creates a Wifi database as illustrated in Fig. 1. The function returns two databases namely trainDB and testDB. The first contains the fingerprints collected during the offline phase (121 locations, 60 sample per location for each AP). The second contains the test data (63 locations, 50 samples per location for each AP).

2. Visualizing fingerprints: Function **show_fingerprints(trainDB,n_ap)** visualizes the locations where WiFi RSS measurements are synthesized during the offline phase.

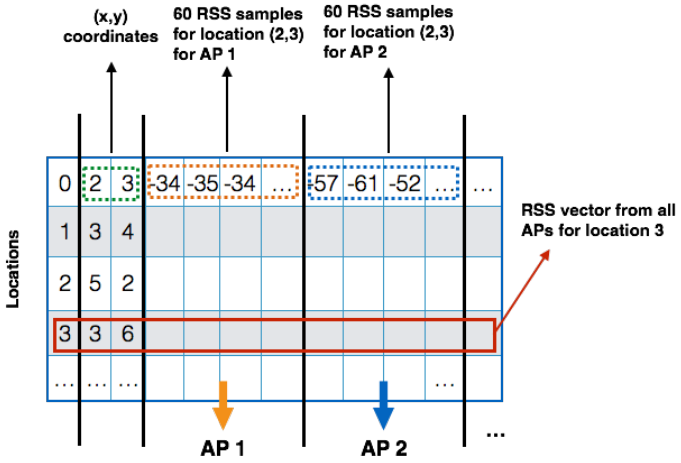

Figure 1: Illustration of the synthetic dataset.

3. Modeling using Gaussian distributions: Instead of just using a single RSS value to represent a location, we can take advantage the variability (i.e. variance) of the measurements by modelling the readings in each location using a Gaussian distribution. The function **fit_data()**, approximates the RSS measurements in each location using a Gaussian distribution and transforms the trainDB as shown in Fig. 2.

3.3 Tasks

3.3.1 RSS Distribution

Run the function **plot_histogram()** to visualize the RSS measurements collected at a particular location during the offline phase and observe the Gaussian approximation. Then answer the following questions:

1. Is the Gaussian distribution a good choice? If not, can you think a better way?
2. What are the pros and cons of using Gaussian distribution over the histogram in this scenario? Explain your answers.

3.3.2 Location Estimation

Next, navigate to function **predict()** and write your code in order to implement the location estimation process of *Horus*. The function should take as input the radio map and the test-set and estimate the user's location. The function should return 2 separate arrays (of the same size, m x 2, where m is the number of locations in the test), one that contains the ground truth locations and a second array that contains the estimated locations. Once you finish with your implementation you can visualize the estimated locations by uncommenting the function plot_path() and you can plot the localization error using the function plot_error(). **Save the figure and paste them in your report. Explain your results as well.**

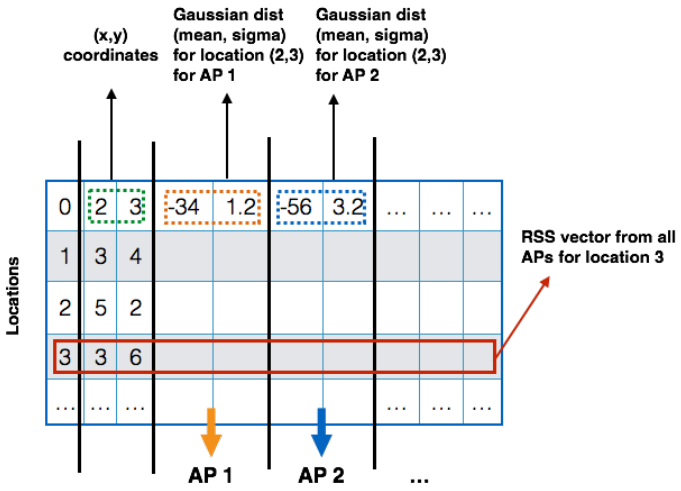

Figure 2: Gaussian distribution on synthetic databased.

3.3.3 Impact of Number of APs

Change the dataset to "set2" and re-run your code. Plot the error and compare it with the previous results on "set1". What do you observe? How does the number of APs affect the localization accuracy? **Explain your finding in the report.**

4 Developing Fingerprinting On Real Data

In this part you will develop your own fingerprinting algorithm in **real_ips.py** and test it on a real-world dataset. You can design own fingerprinting algorithms using other data-driven methods (e.g., deep neural networks) or modify the code of *Horus* so that it adapts to the new dataset. Importantly, you need to test and submit the location predictions through **submit.py**.

4.1 Dataset

Navigate to the folder "real_data":

The training folder inside further contains the WiFi RSSI collected at 15 locations saved in .csv files. These 15 locations are specified in "location.txt" inside training folder as well. Note that the index order of .csv files match the line in the "location.txt" file. For example, the location for collection train.csv is the first line in the txt and (x,y) values are separated by a comma.

The testing folder only contains the WiFi RSSI of 3 unknown locations and you are asked to use fingerprinting algorithm to predict their locations based these WiFi signals. In the meantime, we place a **dummy** "location.txt" file in the test folder. This file is just for your format reference and you need to replace these zeros with your algorithm predictions.

In the **real_ips.py**, function **load_real_csv()** bootstraps you how the real-world WiFi signals are framed in the .csv file. In summary, each .csv file has five

columns corresponding to (timestamp, ssid, mac_address, wifi_channel, RSSI). Keep in mind that using "mac_address" (e.g., 00:81:c4:85:07:a0) to distinguish different WiFi access points rather than ssid (e.g., eduroam).

4.2 Testing

After designing your fingerprinting algorithm and get the predictions on test data, you can pack predictions in the location.txt and test as in the following example:

```
# Testing your solution
(ips) {your path} > python submit.py
```

Successful submission will prompt you the detail prediction errors (in meter) of each location as well as an average error. A key design goal is to make the average error as low as possible. Take a screenshot of your testing result and compare it with your training results. **Explain the gap between training and testing in your report.**

The source code of your fingerprinting algorithm is required as part of the grading.