Lab #1: Measurement of WiFi Signal Strength

EE447 Mobile Network, Luoyi Fu, Spring 2021

Due: *Sunday, Apr 18th*

Name: Hongjie Fang Student ID:518030910150 Email: galaxies@sjtu.edu.cn

1 Purpose and Objective

This lab aims to let us get used to the WiFi system, and accomplish the sampling and measuring of WiFi signal strength through programming in Android on smartphone.

In other words, we are going to design a positioning Android application based on Wifi signal strength, in which we will implement an effective algorithm for indoor positioning using the signal strength data of WiFi sensor in the mobile phone.

2 Indoor Positioning Algorithm

Our approach is based on the idea that the signal strength has a relationship with the distance between the router and the smartphone. From the textbook and other materials, we can find out that they follows the following equation.

$$d = 10^{\frac{|RSSI|-A}{10n}}$$

where A is the power value in the position which is 1m to the router, n is the path loss coefficient, and $RSSI$ is the signal strength of WiFi router. Both A and n is determined by the experiment environment and the type of the router. Thus we can set them fixed in the equation. Since $RSSI$ is usually negative and increases as the power increases, *i.e.*, $RSSI$ increases as the distance decreases, we take absolute value of $RSSI$ in the previous equation, which is quite reasonable.

Suppose we have n WiFi routers, namely $R_1, R_2, \cdots, R_n$. Suppose the i-th WiFi router R_i is located in (x_i, y_i) in the coordinate system, which can be measured priorly. We can measure the distance d_i between the smartphone and the i-th WiFi router R_i using the previous equation. Intuitively, the position of smartphone (x, y) should meet the following requirements.

$$(x - x_i)^2 + (y - y_i)^2 = d_i^2 \qquad (i = 1, 2, \cdots, n)$$

Unfortunately, due to the precison and accuracy reasons of both WiFi signal strength and WiFi router locations, this equation may not have a solution when $n \geq 3$, *i.e.*, when additional requirements are setted. Therefore, we are going to use machine learning methods to calculate the position (x, y) of smartphone. More formally, if the current estimate position of the smartphone is (x, y), then we can define the **loss function** of the current estimators as follows.

$$\mathcal{L}(x, y) = \frac{1}{n} \sum_{i=1}^{n} \left((x - x_i)^2 + (y - y_i)^2 - d_i^2 \right)^2$$

which is the typical square loss function. Hence, we can transform our problem into the following optimization problem.

$$\begin{aligned} \min_{x,y} \quad & \mathcal{L}(x, y) \\ \text{s.t.} \quad & (x, y) \in \mathbb{E} \end{aligned}$$

where $\mathbb{E}$ is the feasible space of the smartphone. In experimental setup, we can regard $\mathbb{E}$ as $\mathbb{R}^2$, *i.e.*, the smartphone can locate in any place in the experimental space.

Therefore, we can use **Gradient Descent Algorithm** to solve the optimization problem. More specifically, we can first derive the gradient of the loss function as follows.

$$\frac{\partial \mathcal{L}(x,y)}{\partial x} = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial x}\left((x-x_i)^2+(y-y_i)^2-d_i^2\right)^2 = \frac{4}{n}\sum_{i=1}^{n}((x-x_i)^3-d_i^2(x-x_i)+(y-y_i)^2(x-x_i))$$
$$\frac{\partial \mathcal{L}(x,y)}{\partial y} = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial y}\left((x-x_i)^2+(y-y_i)^2-d_i^2\right)^2 = \frac{4}{n}\sum_{i=1}^{n}((y-y_i)^3-d_i^2(y-y_i)+(x-x_i)^2(y-y_i))$$

Hence, let $(\hat{x}_t, \hat{y}_t)$ denote the current estimate position of the smartphone, we can update the estimators $(\hat{x}_{t+1}, \hat{y}_{t+1})$ as follows.

$$\hat{x}_{t+1} = \hat{x}_t - \eta_t \left.\frac{\partial \mathcal{L}(x,y)}{\partial x}\right|_{x=\hat{x}_t} = \hat{x}_t - \frac{4\eta_t}{n}\sum_{i=1}^{n}((\hat{x}_t-x_i)^3-d_i^2(\hat{x}_t-x_i)+(\hat{y}_t-y_i)^2(\hat{x}_t-x_i))$$
$$\hat{y}_{t+1} = \hat{y}_t - \eta_t \left.\frac{\partial \mathcal{L}(x,y)}{\partial x}\right|_{y=\hat{y}_t} = \hat{y}_t - \frac{4\eta_t}{n}\sum_{i=1}^{n}((\hat{y}_t-y_i)^3-d_i^2(\hat{y}_t-y_i)+(\hat{x}_t-x_i)^2(\hat{y}_t-y_i))$$

where η_t is the learning rate in the t-th iteration.

Thus, we continue updating the position estimators $(\hat{x}, \hat{y})$ until the process converges, *i.e.*,

$$|\mathcal{L}(\hat{x}_{t+1}, \hat{y}_{t+1}) - \mathcal{L}(\hat{x}_t, \hat{y}_t)| < \varepsilon$$

where ε is the thershold and is usually set as 10^{-8}.

In conclusion, our indoor positioning algorithm is based on the relationship between the distance and the signal strength, and we form the problem as an optimization problem and apply Gradient Descent Algorithm to solve it.

3 Implementation

3.1 Given Codes

We are given some initial codes to complete the lab, so we should understand these codes first.

The initial codes has two Java classes, namely `MainActivity` and `SuperWiFi`, and the latter one is the extension of the first one. We focus on `MainActivity` first. It asks for the necessary permission and then shows the application layout, which contains two buttons `Scan`, `Clean` and a textbox. When pressing the `Scan` button, it will call methods in `SuperWiFi` class to scan for WiFi signals and calculate the $RSSI$, then it will print the result in the textbox. The main logic of the initial given application can be illustrated as follows.

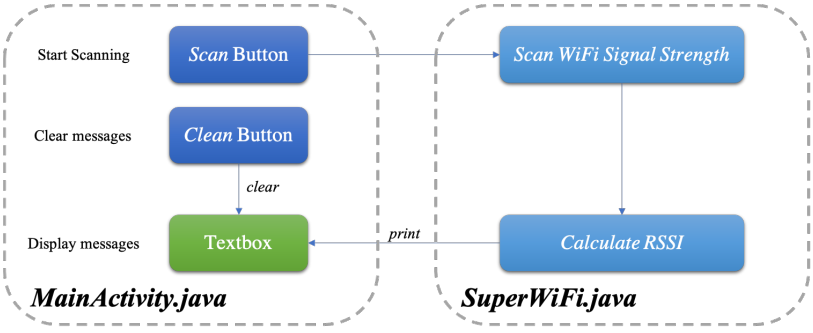

Fig. 1. The logic of the initial given application

3.2 My Implementations

Some critical modifications are made to the original applications. First, we apply the algorithm introduced in Section 2 in `SuperWiFi` module to calculate the position of the smartphones. Here are some specific details of the experimental setup.

- A, which is the power value in the position which is 1m to the router, is set to the average value of 60;
- n, which is the path loss coefficient, is set to 3.25 as usual;
- We use 4 Wi-Fi routers named `406`, `408`, `507`, `508` in the experiment, and their positions are $(-3.6, 0.4)$, $(4.5, -2.3)$, $(2.3, 1.9)$, $(3.2, -5.2)$ respectively;
- η_t, which is the learning rate, is set to 10^{-3} for all t;
- ε, which is the threshold, is set to 10^{-8} as usual.

Some implementation details are listed as follows.

- First, we check the condition of all routers. If some routers are offline, their $RSSE$ value will be 0 and we won't consider them in the following positioning process.
- Second, we calculate the $RSSE$ value of all valid routers. If the number of valid routers is less than 3, then we do not perform positioning since the information is not enough to produce accurate positioning results.
- Then, we apply the algorithm introduced in Section 2 to perform positioning, which is shown in Appendix A.
- Finally, we calculate the necessary values for further visualization process, which is shown in Appendix A.

Apart from the previous modifications, we also add some features in the layout. We add a `Visualization` button to show the positioning result in a more intuitive way. Therefore, the main logic of my modified application can be illustrated as follows.

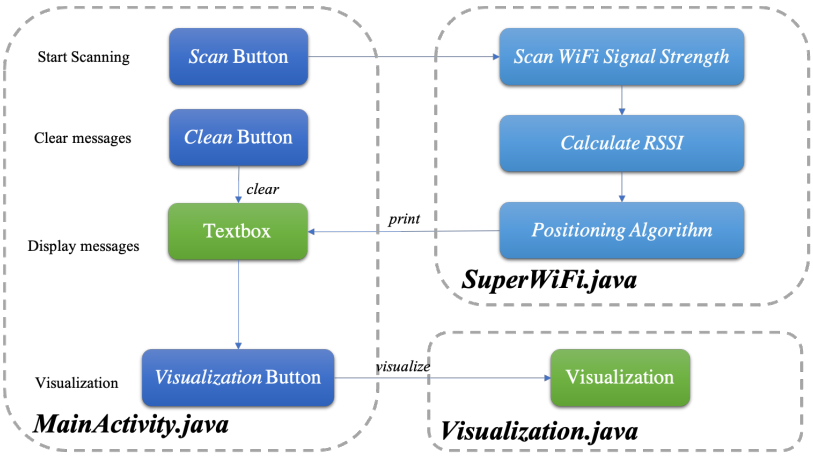

Fig. 2. The logic of my modified application

We mainly use `Canvas` package in Java for visualization, and the effect of visualization will be illustrated later in Section 4

4 Experiments

We use Huawei P20 smartphone in real-world setup testing. The experimental results are shown in Fig. 3 and Fig. 4.

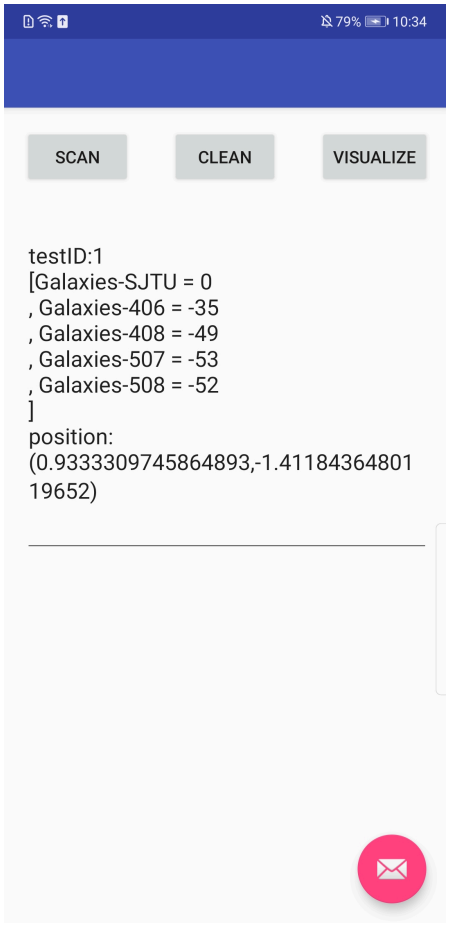

Fig. 3. Experimental Results

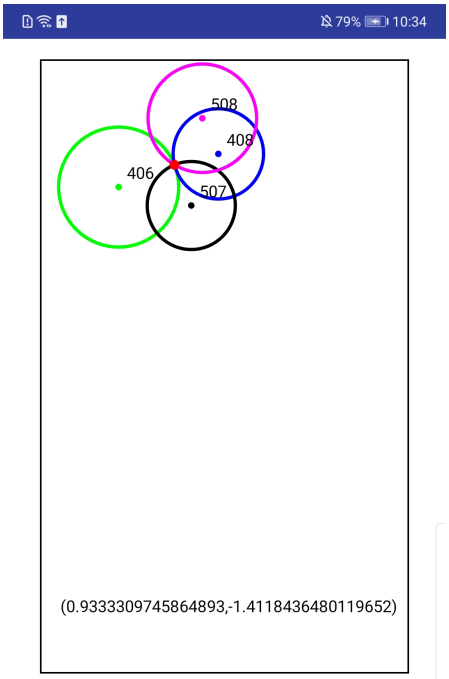

Fig. 4. Visualizations

From Fig. 3 we can see that the application detects the signal strength of all testing WiFis, and does not find the WiFi SJTU since it is not covered yet in the dormitory area. Use the four testing WiFis, we successfully calculate the position of the smartphone, and Fig. 4 illustrates the relative position of the smartphone and the testing WiFis.

Therefore, our application completes all the required functions, and is able to perform positioning based on the WiFi signal strength of several testing WiFis.

5 Questions and Answers

1. **Q: Why is necessary to record all the measured value rather than only the average value? Please give your own explanation.**

   A: If we only record average value of the strength of WiFi signals, we will lose some important information for debug. For example, during my debugging process, I once find that the scanning results seems incorrect. After checking all the records, I find that there are many outliers samples in the records, which may be caused by the distance between the WiFi and the smartphone is too far. When I choose another WiFi which is closer to the smartphone for testing, the results becomes reasonable. Therefore, I benifit a lot from the full records during debugging, and this is an important reason for recording all the measured value rather than only the average value.

2. **Q: Besides the WiFi signal strength, what other information of the Routers can be got in the test?**

   A: After calling `WiFiManager.getScanResults()`, we can get a list of the scanning results shown as follows.

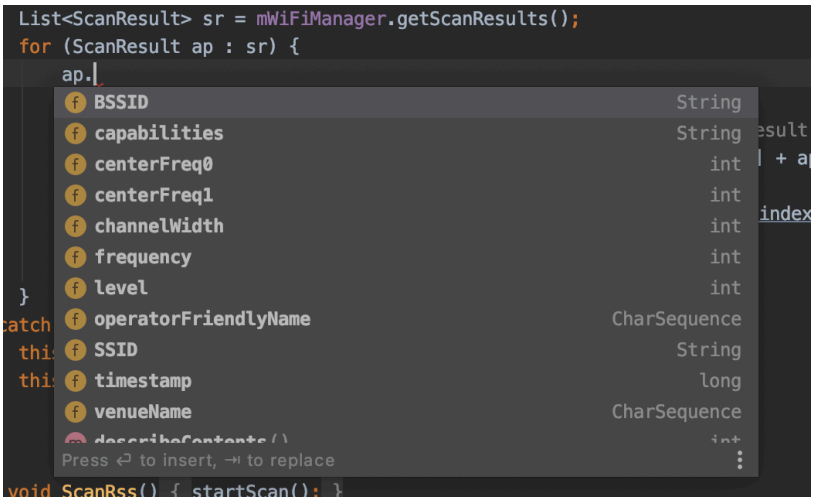

Fig. 5. The information contained in the `ScanResult`

Here are some explanations for the information.

- `BSSID`: the address of the access point.
- `capabilities`: describe the authentication schemes, encryption schemes and other schemes that is supported by the access point.
- `centerFreq0`: center frequency (of the first segment) used if AP bandwidth is greater than $20MHz$.
- `centerFreq1`: center frequency of the second segment only used if AP bandwidth is $80 + 80MHz$.
- `channelWidth`: the channel width.
- `frequency`: the primary frequency of the channel.
- `level`: the detected signal strength level in the unit of `dBm`, *a.k.a.*, $RSSI$.
- `operatorFriendlyName`: passpoint operator name published by the access point.
- `SSID`: the network name.
- `timestamp`: the timestamp of the connection.
- `venueName`: venue name published by the access point.

3. **Q: Why does the scanning need to be operated in thread "scanThread"?**

   A: I think the main reason is **efficiency**. Since the scanning process may take up lots of time, if we implement the scanning in the main thread, it may block other threads and reduce efficiency. Put the scanning process in separate threads can let Android OS dispatch other events while waiting for the scanning results, and the scanning threads won't holds too much resources. Other advantages include better synchronization, parallelism-friendly and so on.

6 Conclusion

To complete this lab, I have read many materals about WiFi-based positioning, and finally choose to use distance as an intermediate the perform positioning. I also form the positioning problem as an optimization problem and use gradient descent algorithm to solve the optimization problem. I benifit a lot from designing the algorithms and implementing it. What's more, I also make a visualization page to visualize the positioning results. The main disadvantage of the algorithm is its inaccuracy. To improve the algorithm, we should find a more accurate method to perform positioning, maybe not relied on the intermediate such as distance, which may be the future direction of the project.

The full implementation codes of the lab is available in my github repository.

Appendix A: Indoor Positioning Algorithm Implementations

Here we show the important part of the implementation of the indoor positioning algorithm.

```
// Localization
ValidWifiNumber = 0;
for (int index = 1; index <= NumberOfWiFi; index++) {
    RSS_Value_Record[index - 1] /=
        RSS_Measurement_Number_Record[index - 1];
    isValid[index - 1] = RSS_Value_Record[index - 1] != 0;
    if (isValid[index - 1])
        ValidWifiNumber ++;
}
if (ValidWifiNumber >= 3) {
    // Calculating Distance
    for (int index = 1; index <= NumberOfWiFi; index++) {
        if (isValid[index - 1])
            dist_WiFi[index - 1] = Math.pow(10,
                (-1 * RSS_Value_Record[index - 1] - A_WiFi[index - 1])
                / (10 * n_Path_Loss_Coeff));
    }
    // Positioning: Using Gradient Descent Algorithm to
    // Solve the Optimization
    Random rand = new Random();
    positionX = (rand.nextDouble() - 0.5) * 20;
    positionY = (rand.nextDouble() - 0.5) * 20;
    double loss = 0;
    for (int index = 1; index <= NumberOfWiFi; index ++) {
        if (isValid[index - 1]) {
            double delta_x = (x_WiFi[index - 1] - positionX)
                * (x_WiFi[index - 1] - positionX);
            double delta_y = (y_WiFi[index - 1] - positionY)
                            * (y_WiFi[index - 1] - positionY);
            loss += (delta_x + delta_y -
                dist_WiFi[index - 1] * dist_WiFi[index - 1])
                * (delta_x + delta_y - dist_WiFi[index - 1] *
                dist_WiFi[index - 1]);
            }
    }
```

```
    loss = loss / ValidWifiNumber;
    double last_loss = 0;
    while (Math.abs(loss - last_loss) > epsilon) {
        // Updating X, Y
        double gradientX = 0, gradientY = 0;
        for (int index = 1; index <= NumberOfWiFi; index++) {
            if (isValid[index - 1]) {
                double delta_x = positionX - x_WiFi[index - 1];
                double delta_y = positionY - y_WiFi[index - 1];
                double dist = dist_WiFi[index - 1];
                gradientX += 4 * delta_x * delta_x * delta_x
                    - 4 * dist * dist * delta_x
                    + 4 * delta_y * delta_y * delta_x;
                gradientY += 4 * delta_y * delta_y * delta_y
                    - 4 * dist * dist * delta_y
                    + 4 * delta_x * delta_x * delta_y;
            }
        }
        gradientX = gradientX / ValidWifiNumber;
        gradientY = gradientY / ValidWifiNumber;
        positionX = positionX - eta * gradientX;
        positionY = positionY - eta * gradientY;

        // Calculate new loss
        last_loss = loss;
        loss = 0;
        for (int index = 1; index <= NumberOfWiFi; index++) {
            if (isValid[index - 1]) {
                double delta_x = (x_WiFi[index - 1] - positionX)
                    * (x_WiFi[index - 1] - positionX);
                double delta_y = (y_WiFi[index - 1] - positionY)
                    * (y_WiFi[index - 1] - positionY);
                loss += (delta_x + delta_y -
                    dist_WiFi[index - 1] * dist_WiFi[index - 1])
                    * (delta_x + delta_y - dist_WiFi[index - 1] *
                    dist_WiFi[index - 1]);
            }
        }
        loss = loss / ValidWifiNumber;
    }
    for (int index = 1; index <= NumberOfWiFi; index ++) {
        if (isValid[index - 1]) {
            double delta_x = (x_WiFi[index - 1] - positionX)
                * (x_WiFi[index - 1] - positionX);
            double delta_y = (y_WiFi[index - 1] - positionY)
                * (y_WiFi[index - 1] - positionY);
            radius[index - 1] = Math.sqrt(delta_x + delta_y);
        }
    }
}
```