CSIT 6910A Independent Project

Indoor BLE Positioning Based on Fingerprinting Algorithm

Student Name: Wang Zhongyi

Student id: 20618841

E-mail: zwangem@connect.ust.hk

Supervisor: Prof. CHAN Gary

Contents

1 Introduction

1.1 Overview

Localization techniques concerning Bluetooth Low Energy has been attracted attention to a variety of services. The presentation of Bluetooth low energy(BLE, now BLE 5.0) have made broad applications popular. BLE beacon is the useful tool that can be implemented in indoor positioning.

Localization systems and realization have long been a central topic among researchers. Systems based on the Angle of Arrival(AOA), Time of Arrival(TOA) and the Time Difference of Arrival(TDOA) have been proposed. In recent years, location fingerprinting algorithm is pervasive, especially when amalgamating with machine learning and deep learning techniques. Deterministic location estimation and probabilistic location estimation are two widely used methods in fingerprinting algorithm.

Location fingerprinting is the method of determining the location of the mobile unit(MU). There are two procedures, first is called the calibration phase, second is the estimation phase. In the calibration phase, a simulated setting is built to collect Received Signal Strength Indicator(RSSI) from access points(APs) in fixed locations(calibration points) where we will assign sensors in the true setting. Collected data are used to construct the radio map and models concerning different methods can be applied. Once the model is built, when moving into the real environment, newly input RSSI can be used to predict the location of the host MU(estimation phase).

1.2 Project Objective

Location fingerprinting is the basis in this project. This project will focus on building the framework for the whole process of location fingerprinting, from data loading, re-formatting, feature creation(construct the radio map) to model construction(calibration phase). Then use the trained model to estimate location(estimation phase).

2 Get to know raspberry pi & router

2.1 Raspberry pi and configuration

Raspberry pi is a new tool for me, the first thing is to know the configuration of raspberry pi and router.

(1) Install Linux operation system on raspberry pi

There is no need for me to install the Linux operation system myself since Linux has already been installed on raspberry, but it doesn’t hurt to know the process. First, download the Linux release image suitable for raspberry pi(e.g. Respbian, Ubuntu MATE).

(2) Set up the SD card

Use Balena Etcher tool to write the Linux image into SD card.

(3) Set up raspberry pi

Once the Linux is installed, I can move forward to config the router.

2.2 Router connection and configuration

Connect the raspberry pi to the Internet(directly connect to the home router), and another router we use is mainly for management. Login on to the management website of the router to change the configuration. Once the raspberry pi is connected to the Internet, it can be used to detect Bluetooth devices.

Type in “scan on” command in the raspberry command window, the raspberry will then begin to sniffer neighbor Bluetooth devices and return devices with their MAC address.

2.3 Basic concepts of Bluetooth communication

Broadcast/scan: a Bluetooth device(BD) broadcast its existence to the outside environment, others devices scan to obtain BD’s information such as MAC address. Raspberry pi can be set as a peripheral device (also a server which stores data), or as beacon to simply detect adjacent Bluetooth devices, this is our choice.

3 Positioning Basics

3.1 Terminology

(1) Received signal strength indicator(RSSI): RSSI is a measurement of the power presented in a received radio signal. The RSSI from the ith transmitter can be formulated as:

$$P(d_i) = P_0 - 10n_p log_{10}\frac{d_t}{d_0} + S_i$$

Where $P(d_i)$ is the estimator of the received power(in dB),d_i, d_0 is the reference distance, S_i is the zero-mean Gaussian random variable in dB. n_p is the path loss exponent.

(2) Time of Arrival(ToA): ToA is the measured time at which first arrives at a receiving end.

(3) Time Difference of Arrival(TDoA): The receiving time difference between two signals propagated through different medium with different propagation speed.

(4) Angle of Arrival(AoA): measurement is a method for determining the direction of propagation of a radio-frequency wave.

(5) Line of Sight(LOS), no line of sight(NLOS): whether barriers exist between two nodes.

3.2 Range-based Positioning

(1) Trilateration: Suppose the knowncoordinates for nodes x_1, x_2, x_3, are (x_a, y_a), (x_b, y_b), (x_c, y_c), respectively. The distances d_i between unknown node and fixed nodes are given. The unknown node's coordinate (x, y) is computed using equation (3-1).

$$\begin{cases} \sqrt{(x-x_a)^2+(y-y_a)^2} = d_a \\ \sqrt{(x-x_b)^2+(y-y_b)^2} = d_b \\ \sqrt{(x-x_c)^2+(y-y_c)^2} = d_c \end{cases} \tag{3-1}$$

(2) MLE(Maximum Likelihood Estimation): Suppose we have known n nodes for which the coordinates are (x_1, y_1), (x_2, y_2), $\cdots$, (x_n, y_n), their distances to the unknown(user) node D(x, y) are d_1, d_2, $\cdots$, d_n, respectively.

$$\begin{cases} \sqrt{(x-x_1)^2+(y-y_1)^2} = d_1 \\ \vdots \\ \sqrt{(x-x_n)^2+(y-y_n)^2} = d_n \end{cases}, \quad A = \begin{bmatrix} 2(x_1-x_n) & 2(y_1-y_n) \\ 2(x_2-x_n) & 2(y_2-y_n) \\ \vdots & \\ 2(x_{n-1}-x_n) & 2(y_{n-1}-y_n) \end{bmatrix}, \quad b = \begin{bmatrix} x_1^2 - x_n^2 + y_1^2 - y_n^2 + d_n^2 - d_1^2 \\ \vdots \\ x_{n-1}^2 - x_n^2 + y_{n-1}^2 - y_n^2 + d_n^2 - d_{n-1}^2 \end{bmatrix}$$

$$X = \begin{bmatrix} x \\ y \end{bmatrix} = (A^T A)^{-1} A^T b \tag{3-2}$$

The coordinate of unknown node D is computed using equation (3-2).

3.3 Fingerprinting algorithm

The abstract procedure of fingerprinting algorithm is shown as below.

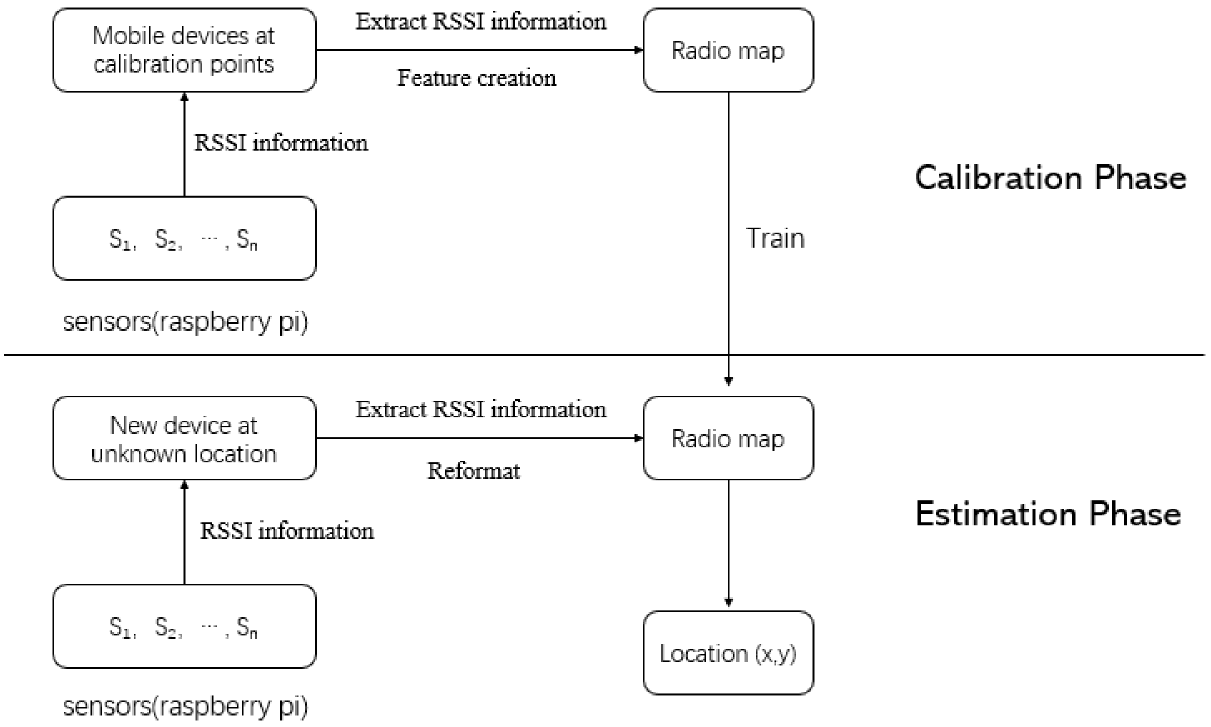


Figure 3.1 Abstract concept of localization fingerprinting algorithm

Concretely, there are two estimation perspectives, one is deterministic location estimation, the other is probabilistic location estimation. For deterministic method, mainly two methods are applied:(1) distance measurement(n-norm distance); (2) KNN algorithm. This method estimates location based on distance and similarity of RSSI between unknown location and those fixed points. The more similar or smaller the distance, the closer the unknown location is to the calibrated point. Probabilistic estimation method is based on Bayes' rule with the aim to compute the conditional pdf(probability density function) of the state X given measurements y.

4 Framework Construction

4.1 Experimental setting

Figure 4.1 and figure 4.2 show the experimental setting. Raspberry pi sensors are numbered 1 to 23. Beacons will be collecting fingerprinting data at each calibration point. 24 calibration points are set in this environment along the path from sensor 1 to 23. At each calibration point, a radio map can be recognized through analyzing the RSSI information.

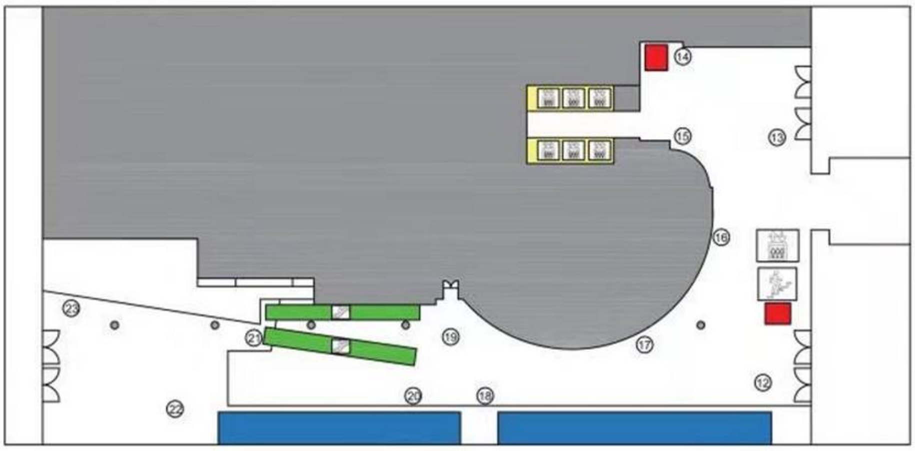


Figure 4.1 experimental setting

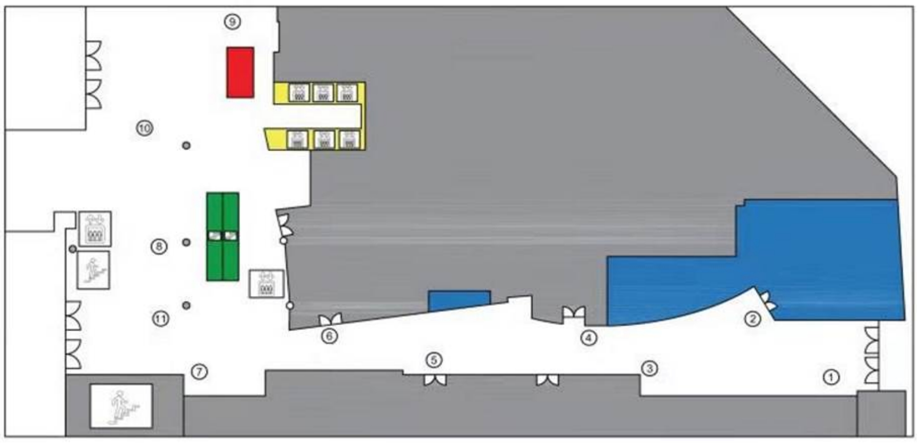


Figure 4.2 experimental setting

4.2 Radio map introduction

At every calibration point, the radio map can be created in the form of a matrix. The radio map has a form of (4-1)

$$RM_i = \begin{bmatrix} M_1 & \\ M_2 & \\ \vdots & , \quad \theta_i \\ M_m & \end{bmatrix} \tag{4-1}$$

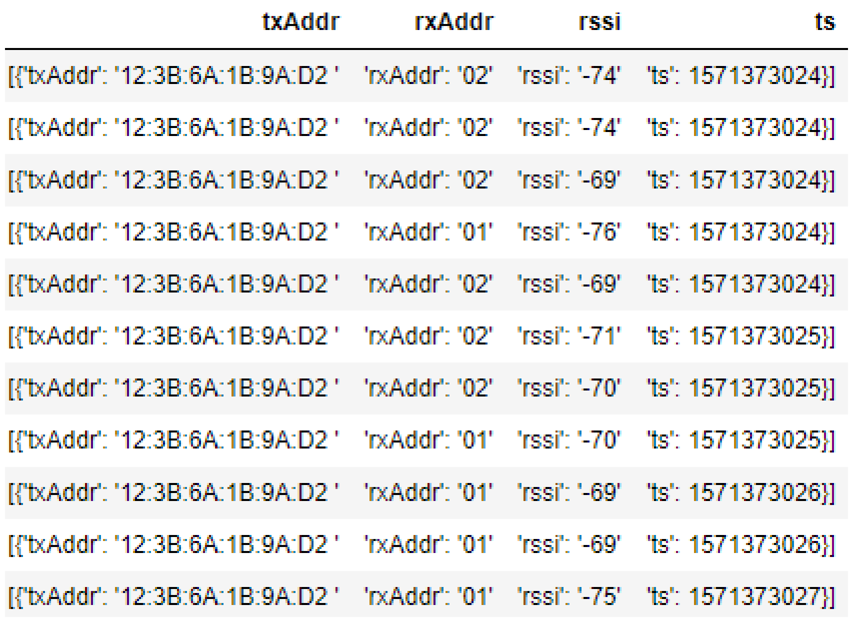
where M_i is the RSSI information of the ith calibration point. Here, $A_{i,j}$ denotes RSSI from jth sensor at the ith calibration point. Specifically, in the setting discussed in section 4.1, m equals to the number of calibration points and n equals to the number of sensors which are 24 and 23, respectively. The parameter k refers to the number of samples from sensor i. In fact, it is not required to receive equal number of RSSI from sensors. The parameter θ_i contains other information needed in the location estimation phase, and can be for example the direction θ_i = {north, south, east, west} of the MU.

$$M_i = \begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,k} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,k} \\ \vdots & & & \\ A_{n,1} & A_{n,2} & \cdots & A_{n,k} \end{bmatrix} \tag{4-2}$$

Basically, mean of RSSI detected at a calibration point of some sensors is used.

4.3 Raw data preprocessing

Data collected by sensors at each calibration points have the form same as table 4.1.



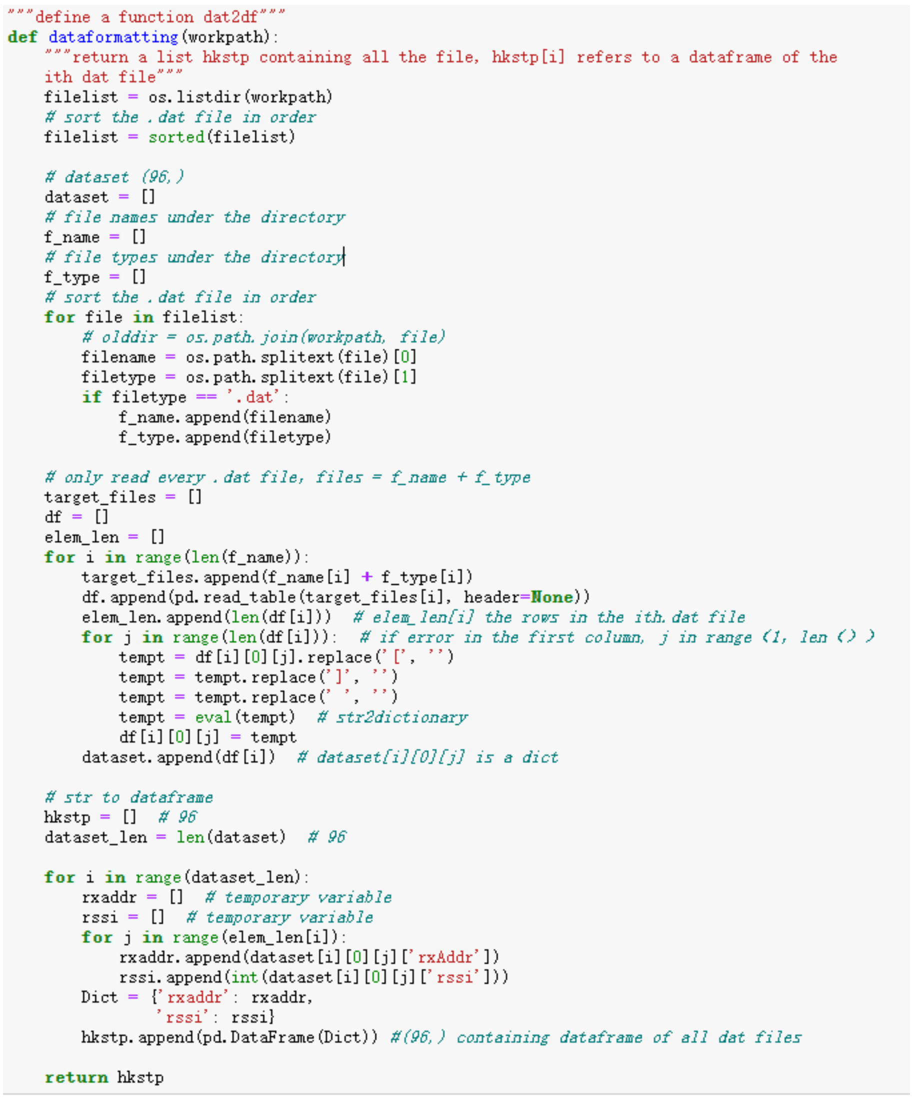
txAddr	rxAddr	rssi	ts
[{'txAddr': '12:3B:6A:1B:9A:D2 '	'rxAddr': '02'	'rssi': '-74'	'ts': 1571373024}]
[{'txAddr': '12:3B:6A:1B:9A:D2 '	'rxAddr': '02'	'rssi': '-74'	'ts': 1571373024}]
[{'txAddr': '12:3B:6A:1B:9A:D2 '	'rxAddr': '02'	'rssi': '-69'	'ts': 1571373024}]
[{'txAddr': '12:3B:6A:1B:9A:D2 '	'rxAddr': '01'	'rssi': '-76'	'ts': 1571373024}]
[{'txAddr': '12:3B:6A:1B:9A:D2 '	'rxAddr': '02'	'rssi': '-69'	'ts': 1571373024}]
[{'txAddr': '12:3B:6A:1B:9A:D2 '	'rxAddr': '02'	'rssi': '-71'	'ts': 1571373025}]
[{'txAddr': '12:3B:6A:1B:9A:D2 '	'rxAddr': '02'	'rssi': '-70'	'ts': 1571373025}]
[{'txAddr': '12:3B:6A:1B:9A:D2 '	'rxAddr': '01'	'rssi': '-70'	'ts': 1571373025}]
[{'txAddr': '12:3B:6A:1B:9A:D2 '	'rxAddr': '01'	'rssi': '-69'	'ts': 1571373026}]
[{'txAddr': '12:3B:6A:1B:9A:D2 '	'rxAddr': '01'	'rssi': '-69'	'ts': 1571373026}]
[{'txAddr': '12:3B:6A:1B:9A:D2 '	'rxAddr': '01'	'rssi': '-75'	'ts': 1571373027}]

Table 4.1 data in a raw file

“txAddr” refers to beacons(here as mobile devices), “rxAddr” refers to sensors number, “rssi” is the value of received signal strength, “ts” stands for the timestamp. For instance, in table 4.1, the beacon address is 12:3B:6A:1B:9A:D2, the sensors signals received at this calibration point are have label 01 and 02. At each timestamp, RSSI values are detected and obtain at the sensor ends, the information is uploaded from the beacon to the server.

4.3.1 Data loading

Here, the data collected from the sensors(raspberry pi) are stored in some ".dat" files. At each calibration point, RSSI are collected from four direction, namely, θ_i = direction{1,2,3,4}. In reading the files, only relevant files in a sorted manner are loaded. This is finished by the function "dataformatting".



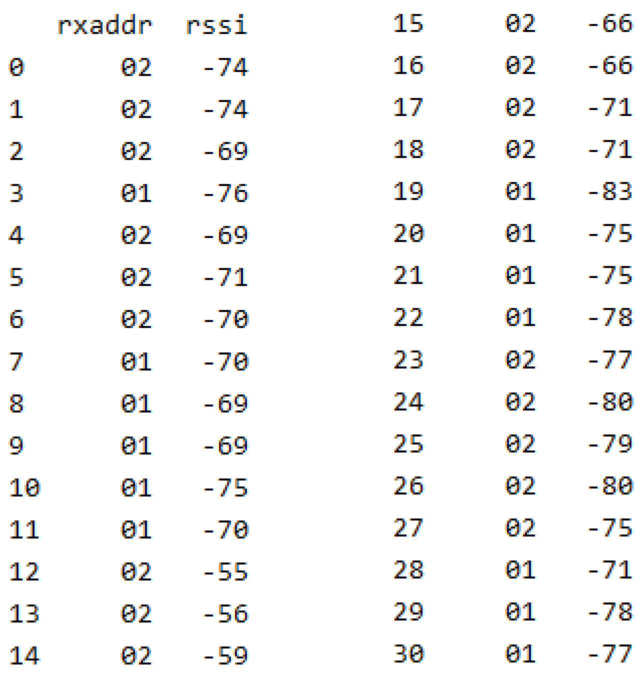
```python
"""define a function dat2df"""
def dataformatting(workpath):
    """return a list hkstp containing all the file, hkstp[i] refers to a dataframe of the
    ith dat file"""
    filelist = os.listdir(workpath)
    # sort the .dat file in order
    filelist = sorted(filelist)

    # dataset (96,)
    dataset = []
    # file names under the directory
    f_name = []
    # file types under the directory
    f_type = []
    # sort the .dat file in order
    for file in filelist:
        # olddir = os.path.join(workpath, file)
        filename = os.path.splitext(file)[0]
        filetype = os.path.splitext(file)[1]
        if filetype == '.dat':
            f_name.append(filename)
            f_type.append(filetype)

    # only read every .dat file, files = f_name + f_type
    target_files = []
    df = []
    elem_len = []
    for i in range(len(f_name)):
        target_files.append(f_name[i] + f_type[i])
        df.append(pd.read_table(target_files[i], header=None))
        elem_len.append(len(df[i]))  # elem_len[i] the rows in the ith.dat file
        for j in range(len(df[i])):  # if error in the first column, j in range (1, len () )
            tempt = df[i][0][j].replace('[', '')
            tempt = tempt.replace(']', '')
            tempt = tempt.replace(' ', '')
            tempt = eval(tempt)  # str2dictionary
            df[i][0][j] = tempt
        dataset.append(df[i])  # dataset[i][0][j] is a dict

    # str to dataframe
    hkstp = []  # 96
    dataset_len = len(dataset)  # 96

    for i in range(dataset_len):
        rxaddr = []  # temporary variable
        rssi = []  # temporary variable
        for j in range(elem_len[i]):
            rxaddr.append(dataset[i][0][j]['rxAddr'])
            rssi.append(int(dataset[i][0][j]['rssi']))
        Dict = {'rxaddr': rxaddr,
                'rssi': rssi}
        hkstp.append(pd.DataFrame(Dict)) #(96,) containing dataframe of all dat files

    return hkstp
```

Figure 4.3 file loading function

The input is the path of data files, and it returns a list in which each row consists of sensor numbers and corresponding RSSI. The experiment data I have contains 96 files(24 calibration points multiplies

4 directions). The RSSI data may not be sufficient enough, but it doesn't affect to build up the process procedure. The result of defined function "dataformatting" actually returns a list of 96 elements, each element has the similar pandas dataframe form shown in figure 4.4.

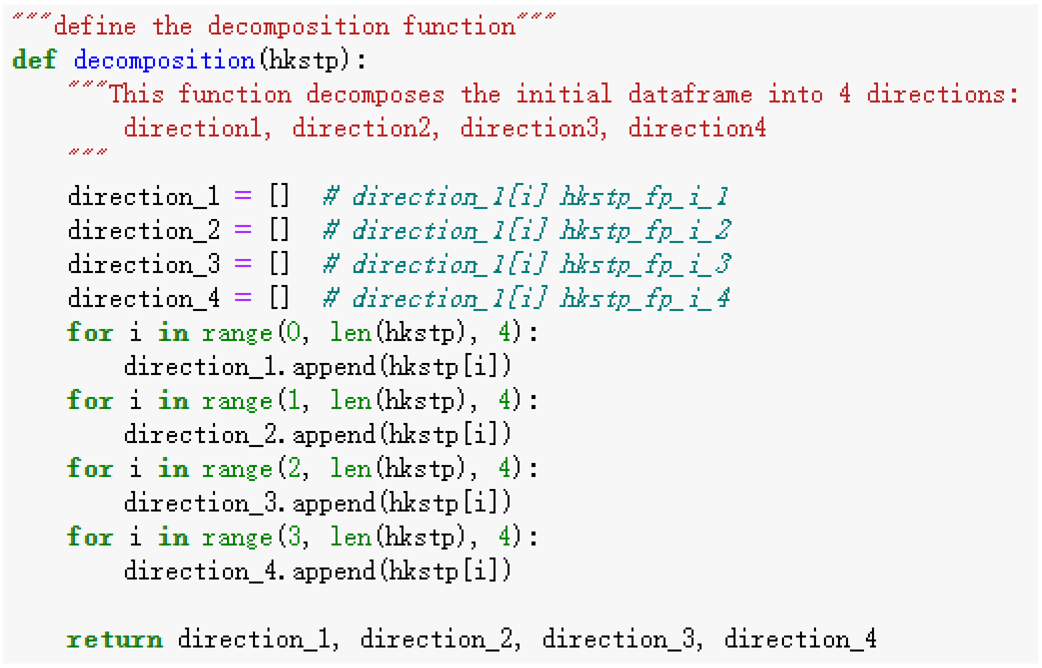
The next thing to do is to separate the list of data formatting into 4 directions.



```
    rxaddr  rssi          15     02    -66
0       02   -74          16     02    -66
1       02   -74          17     02    -71
2       02   -69          18     02    -71
3       01   -76          19     01    -83
4       02   -69          20     01    -75
5       02   -71          21     01    -75
6       02   -70          22     01    -78
7       01   -70          23     02    -77
8       01   -69          24     02    -80
9       01   -69          25     02    -79
10      01   -75          26     02    -80
11      01   -70          27     02    -75
12      02   -55          28     01    -71
13      02   -56          29     01    -78
14      02   -59          30     01    -77
```

Figure 4.4 sample of dataformatting

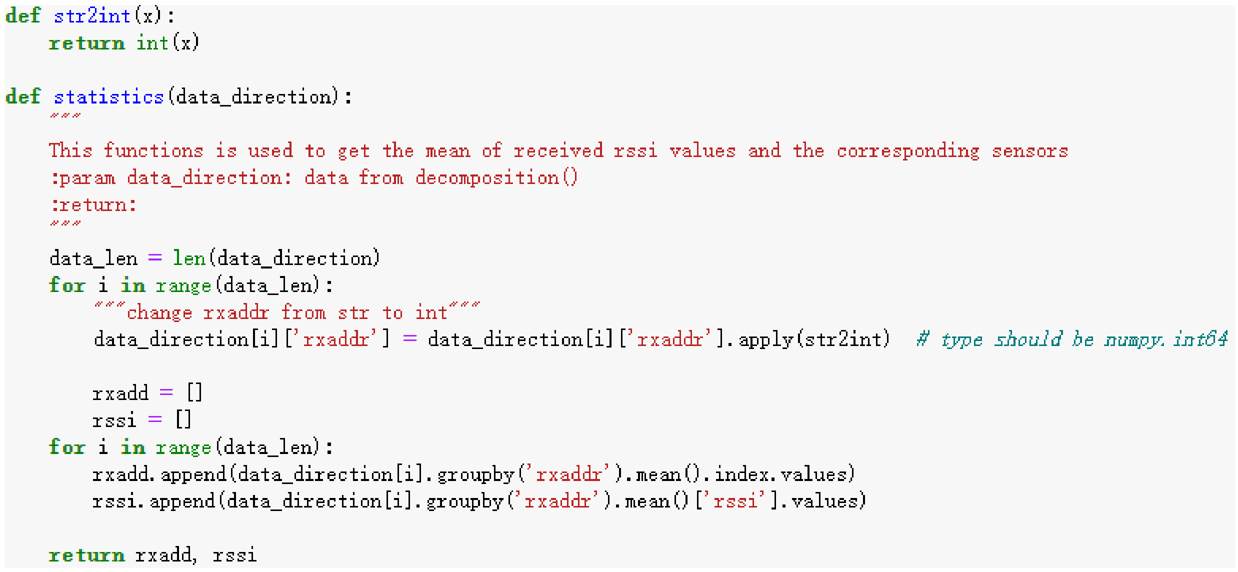
4.3.2 Direction decomposition and information extraction



```python
"""define the decomposition function"""
def decomposition(hkstp):
    """This function decomposes the initial dataframe into 4 directions:
       direction1, direction2, direction3, direction4
    """
    direction_1 = []  # direction_1[i] hkstp_fp_i_1
    direction_2 = []  # direction_1[i] hkstp_fp_i_2
    direction_3 = []  # direction_1[i] hkstp_fp_i_3
    direction_4 = []  # direction_1[i] hkstp_fp_i_4
    for i in range(0, len(hkstp), 4):
        direction_1.append(hkstp[i])
    for i in range(1, len(hkstp), 4):
        direction_2.append(hkstp[i])
    for i in range(2, len(hkstp), 4):
        direction_3.append(hkstp[i])
    for i in range(3, len(hkstp), 4):
        direction_4.append(hkstp[i])

    return direction_1, direction_2, direction_3, direction_4
```

Figure 4.5 function of direction decomposition

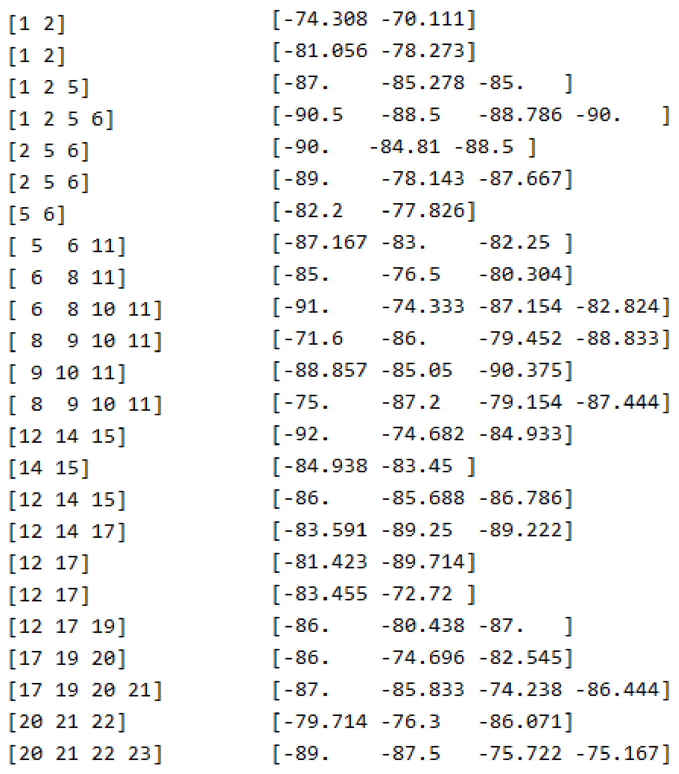
This function just divides the file of 96 length into 4 equal length lists. Since my goal is to build up the radio map. Mean values of RSSI and its corresponding receivers are needed. In others words, further statistics of the data is eager to obtain.



```python
def str2int(x):
    return int(x)

def statistics(data_direction):
    """
    This functions is used to get the mean of received rssi values and the corresponding sensors
    :param data_direction: data from decomposition()
    :return:
    """
    data_len = len(data_direction)
    for i in range(data_len):
        """change rxaddr from str to int"""
        data_direction[i]['rxaddr'] = data_direction[i]['rxaddr'].apply(str2int)  # type should be numpy.int64

        rxadd = []
        rssi = []
    for i in range(data_len):
        rxadd.append(data_direction[i].groupby('rxaddr').mean().index.values)
        rssi.append(data_direction[i].groupby('rxaddr').mean()['rssi'].values)

    return rxadd, rssi
```

Figure 4.6 statistics function

For direction 1, there are 24 calibration points(refer to 24 rows). At each points, there are different sensor sequences and RSSI values, these information(indeed, more information can be digged out depending on usage) is what I need to construct the radio map, as shown in figure 4.7.



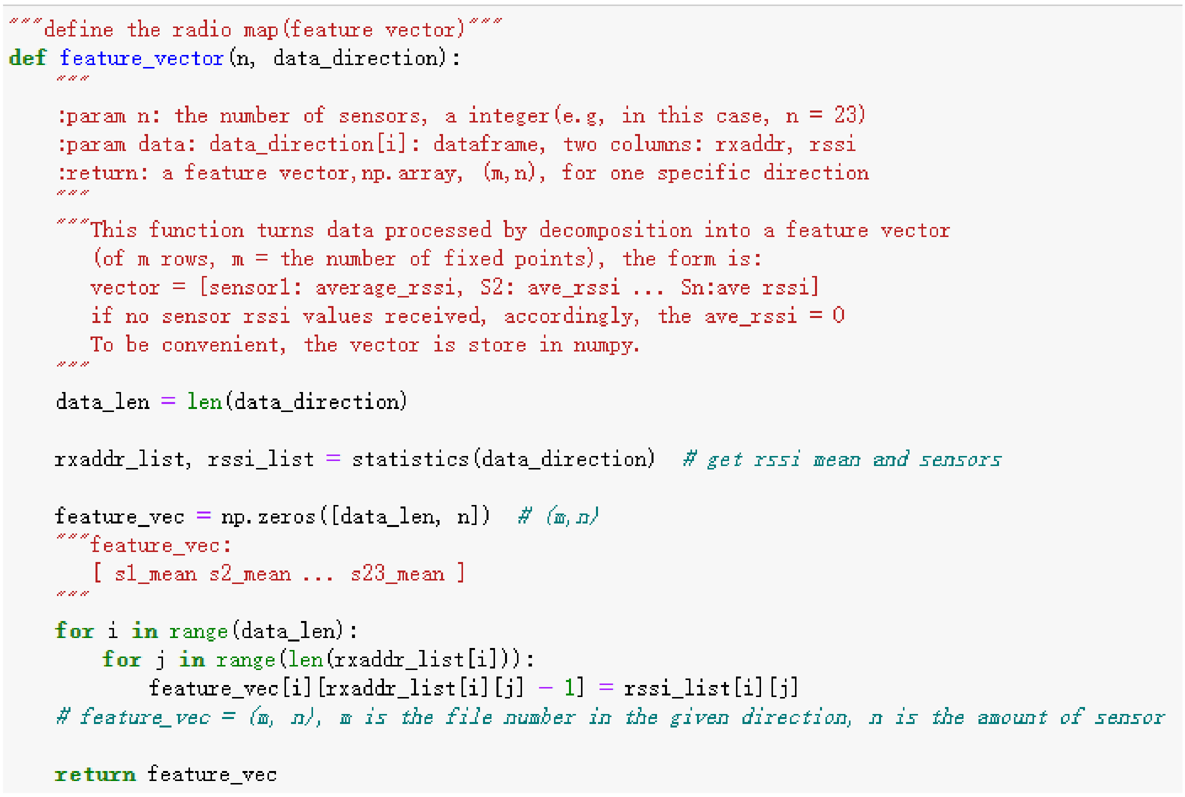
```
[1 2]                 [-74.308 -70.111]
[1 2]                 [-81.056 -78.273]
[1 2 5]               [-87.    -85.278 -85.   ]
[1 2 5 6]             [-90.5   -88.5   -88.786 -90.   ]
[2 5 6]               [-90.   -84.81 -88.5 ]
[2 5 6]               [-89.    -78.143 -87.667]
[5 6]                 [-82.2   -77.826]
[ 5  6 11]            [-87.167 -83.    -82.25 ]
[ 6  8 11]            [-85.    -76.5   -80.304]
[ 6  8 10 11]         [-91.    -74.333 -87.154 -82.824]
[ 8  9 10 11]         [-71.6   -86.    -79.452 -88.833]
[ 9 10 11]            [-88.857 -85.05  -90.375]
[ 8  9 10 11]         [-75.    -87.2   -79.154 -87.444]
[12 14 15]            [-92.    -74.682 -84.933]
[14 15]               [-84.938 -83.45 ]
[12 14 15]            [-86.    -85.688 -86.786]
[12 14 17]            [-83.591 -89.25  -89.222]
[12 17]               [-81.423 -89.714]
[12 17]               [-83.455 -72.72 ]
[12 17 19]            [-86.    -80.438 -87.   ]
[17 19 20]            [-86.    -74.696 -82.545]
[17 19 20 21]         [-87.    -85.833 -74.238 -86.444]
[20 21 22]            [-79.714 -76.3   -86.071]
[20 21 22 23]         [-89.    -87.5   -75.722 -75.167]
```

Figure 4.7 sensors received(left), mean RSSI value of each sensor(right) at each point

4.3.3 Feature vector creation

The kind of final step in creating radio map is building up the feature vector. As mentioned in section 4.2, in our case, I have to ultimately draw a 24×23 metrix RM.

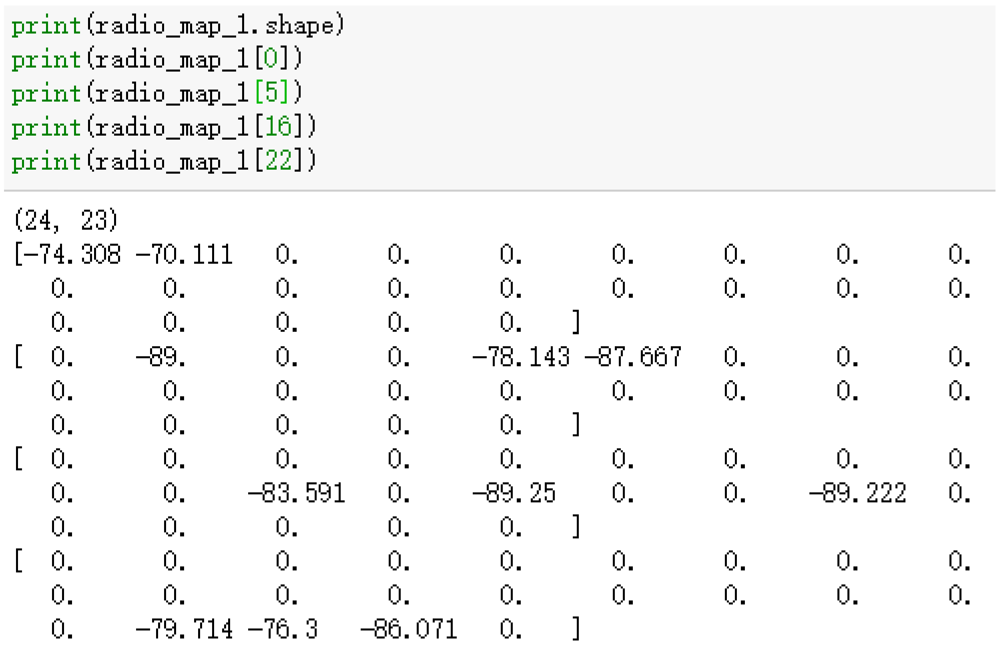
$$RM = \begin{pmatrix} \text{calibration point 1: sensor1 average RSSI} & \cdots & \text{sensor23 average RSSI} \\ \text{calibration point 2: sensor1 average RSSI} & \cdots & \text{sensor23 average RSSI} \\ & \vdots & \\ \text{calibration point 24: sensor1 average RSSI} & \cdots & \text{sensor23 average RSSI} \end{pmatrix} \quad (4\text{-}3)$$



```
"""define the radio map(feature vector)"""
def feature_vector(n, data_direction):
    """
    :param n: the number of sensors, a integer(e.g, in this case, n = 23)
    :param data: data_direction[i]: dataframe, two columns: rxaddr, rssi
    :return: a feature vector,np.array, (m,n), for one specific direction
    """
    """This function turns data processed by decomposition into a feature vector
       (of m rows, m = the number of fixed points), the form is:
       vector = [sensor1: average_rssi, S2: ave_rssi ... Sn:ave rssi]
       if no sensor rssi values received, accordingly, the ave_rssi = 0
       To be convenient, the vector is store in numpy.
    """
    data_len = len(data_direction)

    rxaddr_list, rssi_list = statistics(data_direction)  # get rssi mean and sensors

    feature_vec = np.zeros([data_len, n])  # (m,n)
    """feature_vec:
       [ s1_mean s2_mean ... s23_mean ]
    """
    for i in range(data_len):
        for j in range(len(rxaddr_list[i])):
            feature_vec[i][rxaddr_list[i][j] - 1] = rssi_list[i][j]
    # feature_vec = (m, n), m is the file number in the given direction, n is the amount of sensor

    return feature_vec
```

Figure 4.8 feature vector function

In figure 4.9, 4 sample feature vector is presented, they correspond to calibration points at position 1, 6, 17 and 23, respectively. For our experiment setting here, the detected sensor numbers are usually 2 or 3, so the vector or the whole matrix can be quite sparse. In real practice, more sensors can be deployed instead to acquire a better outcome.



```
print(radio_map_1.shape)
print(radio_map_1[0])
print(radio_map_1[5])
print(radio_map_1[16])
print(radio_map_1[22])
```

```
(24, 23)
[-74.308 -70.111   0.      0.      0.      0.      0.      0.      0.
   0.      0.      0.      0.      0.      0.      0.      0.      0.
   0.      0.      0.      0.      0.    ]
[  0.    -89.      0.      0.    -78.143 -87.667   0.      0.      0.
   0.      0.      0.      0.      0.      0.      0.      0.      0.
   0.      0.      0.      0.      0.    ]
[  0.      0.      0.      0.      0.      0.      0.      0.      0.
   0.      0.    -83.591   0.    -89.25    0.      0.    -89.222   0.
   0.      0.      0.      0.      0.    ]
[  0.      0.      0.      0.      0.      0.      0.      0.      0.
   0.      0.      0.      0.      0.      0.      0.      0.      0.
   0.    -79.714 -76.3   -86.071   0.    ]
```

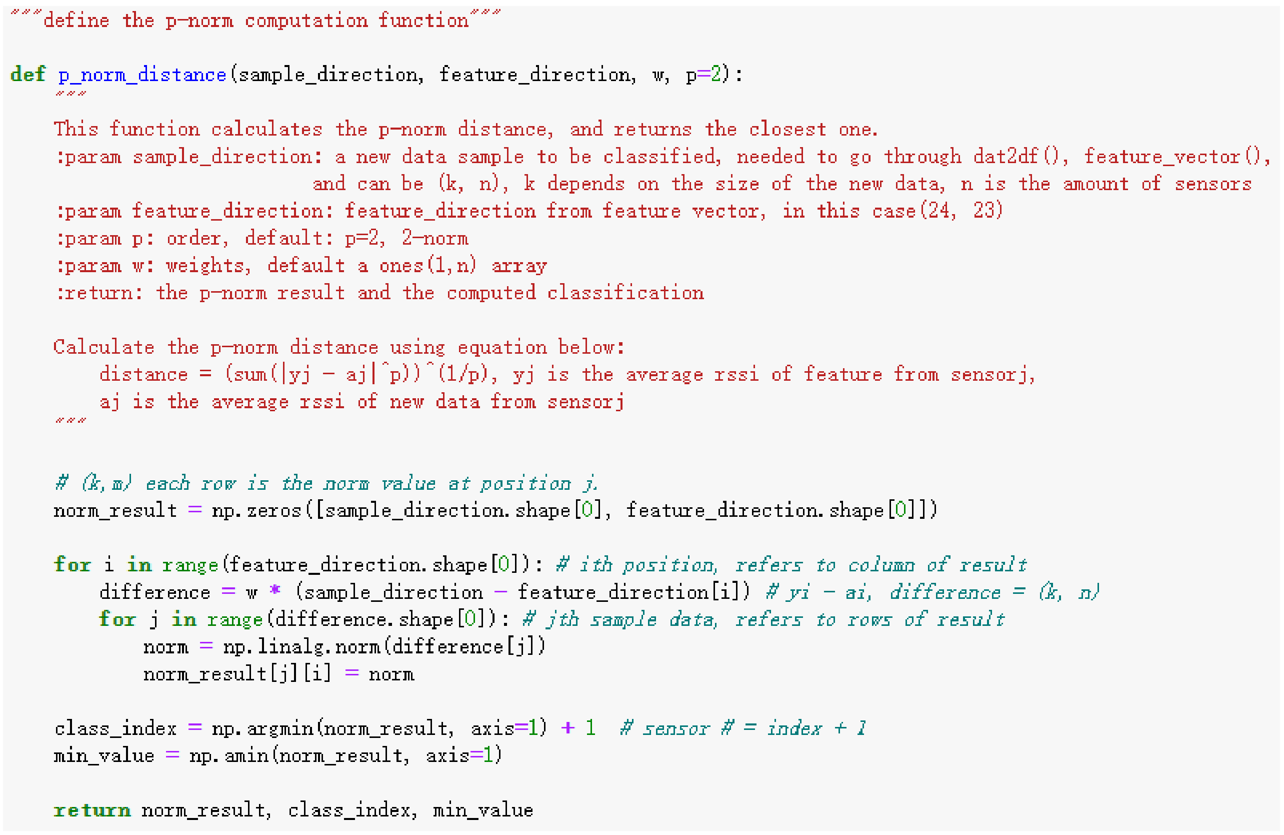
Figure 4.9 Vectorization result

4.3.4 Distance measurement

With the data at hand, distance or similarity computation can be made to classify a new coming RSSI data from a MU. Different measures are used to find the best match between observations and the radio map. The common choice for the comparison measure is to use the p-norm to assign a non-negative value to the fingerprint. Concretely, 1-norm and 2-norm are two most common measurements in distance computation.

$$\| x \|_p = (\sum_{i=1}^{n} | x_i |)^{1/p} \tag{4-4}$$

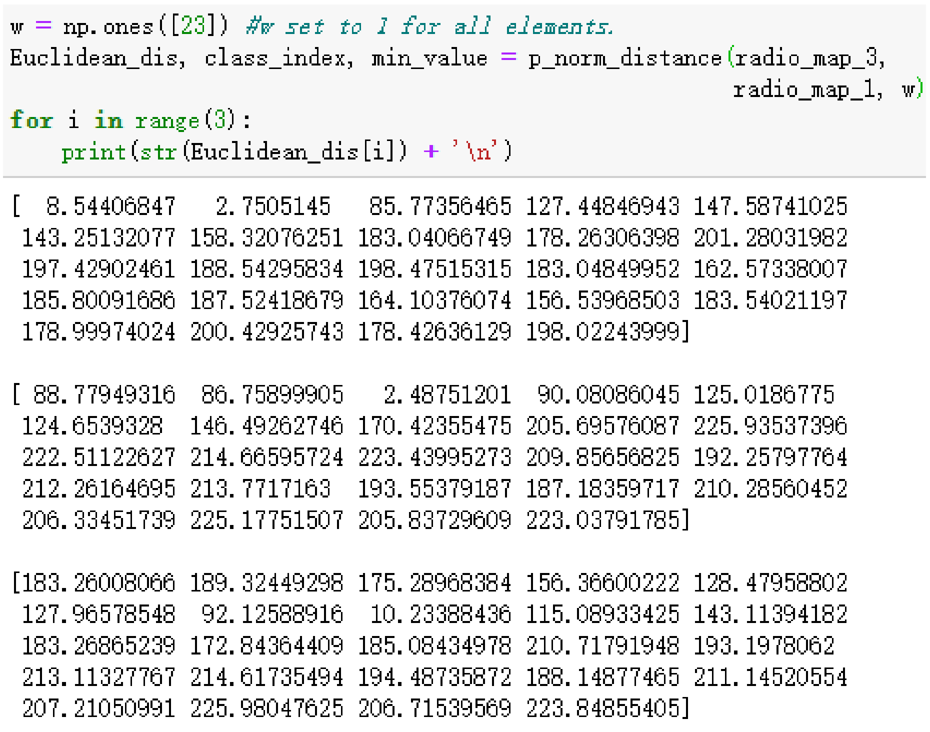
Formula (4-4) shows how to compute p-norm distance. x_i refers to elementwise difference between two feature vectors, the new one and the calibration one. Through computing distance with calibrated feature vector one by one in the radio map, the one with the smaller distance is chosen as the closer location. Suppose we have a MU with a feature vector [-78 -73 0 0 0 … 0] to be localized, through computing with 96 feature vectors in the radio map matrix, the position will be classified as closest to calibration point 1.



```
"""define the p-norm computation function"""

def p_norm_distance(sample_direction, feature_direction, w, p=2):
    """
    This function calculates the p-norm distance, and returns the closest one.
    :param sample_direction: a new data sample to be classified, needed to go through dat2df(), feature_vector(),
                             and can be (k, n), k depends on the size of the new data, n is the amount of sensors
    :param feature_direction: feature_direction from feature vector, in this case(24, 23)
    :param p: order, default: p=2, 2-norm
    :param w: weights, default a ones(1,n) array
    :return: the p-norm result and the computed classification

    Calculate the p-norm distance using equation below:
        distance = (sum(|yj - aj|^p))^(1/p), yj is the average rssi of feature from sensorj,
        aj is the average rssi of new data from sensorj
    """

    # (k,m) each row is the norm value at position j.
    norm_result = np.zeros([sample_direction.shape[0], feature_direction.shape[0]])

    for i in range(feature_direction.shape[0]): # ith position, refers to column of result
        difference = w * (sample_direction - feature_direction[i]) # yi - ai, difference = (k, n)
        for j in range(difference.shape[0]): # jth sample data, refers to rows of result
            norm = np.linalg.norm(difference[j])
            norm_result[j][i] = norm

    class_index = np.argmin(norm_result, axis=1) + 1  # sensor # = index + 1
    min_value = np.amin(norm_result, axis=1)

    return norm_result, class_index, min_value
```

Figure 4.10 weighted p-norm distance

As a matter of fact, a weight vector can be introduced to strike some significance in some sensor average RSSI to make the prediction more accurate.

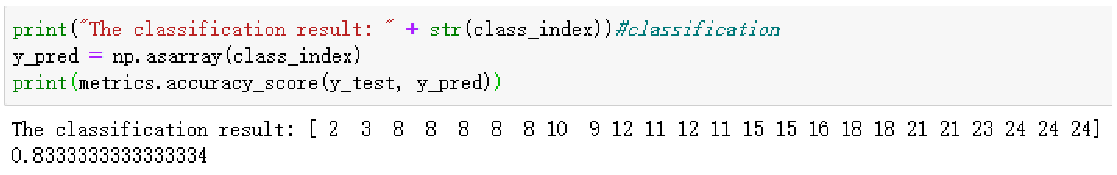


```
w = np.ones([23]) #w set to 1 for all elements.
Euclidean_dis, class_index, min_value = p_norm_distance(radio_map_3,
                                                        radio_map_1, w)
for i in range(3):
    print(str(Euclidean_dis[i]) + '\n')
```

```
[  8.54406847   2.7505145   85.77356465 127.44846943 147.58741025
 143.25132077 158.32076251 183.04066749 178.26306398 201.28031982
 197.42902461 188.54295834 198.47515315 183.04849952 162.57338007
 185.80091686 187.52418679 164.10376074 156.53968503 183.54021197
 178.99974024 200.42925743 178.42636129 198.02243999]

[ 88.77949316  86.75899905   2.48751201  90.08086045 125.0186775
 124.6539328  146.49262746 170.42355475 205.69576087 225.93537396
 222.51122627 214.66595724 223.43995273 209.85656825 192.25797764
 212.26164695 213.7717163  193.55379187 187.18359717 210.28560452
 206.33451739 225.17751507 205.83729609 223.03791785]

[183.26008066 189.32449298 175.28968384 156.36600222 128.47958802
 127.96578548  92.12588916  10.23388436 115.08933425 143.11394182
 183.26865239 172.84364409 185.08434978 210.71791948 193.1978062
 213.11327767 214.61735494 194.48735872 188.14877465 211.14520554
 207.21050991 225.98047625 206.71539569 223.84855405]
```

Figure 4.11 2-norm(Euclidean) distance calculation

After computing the distance, the classification(localization) result should also be returned. Here, I use some artificial data to see the accuracy. The accuracy is 83.3%, good start for a baseline.



```
print("The classification result: " + str(class_index))#classification
y_pred = np.asarray(class_index)
print(metrics.accuracy_score(y_test, y_pred))
```

```
The classification result: [ 2  3  8  8  8  8  8 10  9 12 11 12 11 15 15 16 18 18 21 21 23 24 24 24]
0.8333333333333334
```

Figure 4.12 accuracy test

5. Conclusion and prospects

Due to limited time, data and some other issues, some other interesting aspects still remain to be explored. The experiment setting is kind of a simplified version, and the data collected is insufficient and not comprehensive enough. With more data, the radio map will be more adequate and precise.

Previously, the radio map in my design only includes RSSI mean and corresponding sensor numbers. This may cause the radio map sparse and not make full potentials of the information. Improvements considering several different orientations can be further made.

First, the real environment can be divided into grids and center points can be approximation to represent the coordinate of the grids, so new coming MU with RSSI can be classified to a grid with closest distance or based on other methods.

Another promising way to handle this is extracting other information such as RSSI standard deviation, sensor ratios and so forth, and put them into the radio map as features. Combining with some machine learning techniques, such as SVM, Naïve Bayes and a multilayer perceptron or neural networks, to learn the weight parameters, converting it into a classification problem. As long as the data is adequate for this task, implementing these tools is nothing tricky.

The independent project actually cultivates my comprehensive ability in searching and understanding relative materials such as literature, critical thinking in a pragmatic manner, and augments my deeper understanding between theoretical algorithms and practical realization. The fingerprinting algorithm seems to be a once-and-for-all elixir in localization, but when making it to the ground, revisions and changes still need to be done according to situations, from putting features into the radio map to the choosing of models, they are of crucial importance. A prominent outcome often lies in how I build the experiment setting, what I choose to be the features and how fine I tune the features and parameters. Models and algorithms they are something to be added but even not the decisive part once the data and preprocess are completed.

Appendix A

Minutes of the 1st Project Meeting

Date:	12 Sep. 2019 (Thursday)
Time:	16:00pm
Place:	3142 Room, Academic building
Attending:	Prof. CHAN Gary,
	Engineer Zhongming Lin,
	Wang Zhongyi
Absent:	None
Recorder:	Wang Zhongyi

1. Approval of minutes

   This is first formal group meeting, no minutes to approve.

2. Discussion items

   - Specific topic

     Decided to choose the topic Indoor BLE Positioning

   - Clarified main work tasks and direction

   - Pick up raspberry pi & router, get to know about the implementation of the device

3. Meeting adjournment and next meeting

   The meeting is held on time. The next meeting will be held in two or three weeks later.

Minutes of the 2nd Project Meeting

Date:	24 Sep. 2019 (Tuesday)
Time:	14:00pm
Place:	3142 Room, Academic building
Attending:	Prof. CHAN Gary,
	Engineer Zhongming Lin,
	Wang Zhongyi
Absent:	None
Recorder:	Wang Zhongyi

1. Approval of minutes

   Approved.

2. Discussion items

   - The problems faced when doing the configuration of raspberry pi & router
   - The experiment setting and data collection specifics
   - Methods and algorithms to be used in localization

3. Meeting adjournment and next meeting

   The meeting is held on at 14:30. The next meeting will be held on 8 Oct. 2019, tentatively.

Minutes of the 3rd Project Meeting

Date:	14 Oct. 2019 (Monday)
Time:	14:00pm
Place:	3142 Room, Academic building
Attending:	Prof. CHAN Gary,
	Engineer Zhongming Lin,
	Wang Zhongyi
Absent:	None
Recorder:	Wang Zhongyi

1. Approval of minutes

   Approved.

2. Discussion items
   - Framework building up for radio map creation
   - Detail discussion on fingerprinting algorithm
   - Solving difficulties faced in personal work

3. Meeting adjournment and next meeting

   The meeting is held on at 14:00. The next meeting will be two or three weeks later, tentatively.

Minutes of the 4th Project Meeting

Date:	5 Nov. 2019 (Tuesday)
Time:	16:00pm
Place:	3142 Room, Academic building
Attending:	Prof. CHAN Gary,
	Engineer Zhongming Lin,
	Wang Zhongyi
Absent:	None
Recorder:	Wang Zhongyi

1. Approval of minutes

   Approved.

2. Discussion items

   - Outcome of the framework
   - Result of fingerprinting algorithm
   - Improvements and revisions

3. Meeting adjournment and next meeting

   The meeting is held on at 16:00. The next meeting will be held by appointment to discuss details about the final report.

Minutes of the 5th Project Meeting

Date:	11 Nov. 2019 (Monday)
Time:	16:00pm
Place:	3142 Room, Academic building
Attending:	Prof. CHAN Gary,
	Wang Zhongyi
Absent:	None
Recorder:	Wang Zhongyi

1. Approval of minutes

   Approved.

2. Discussion items

   None.

3. Meeting adjournment and next meeting

   The meeting is canceled due to safety concerns. Details of the final report is discussed through e-mail.