5-2018

Long-Range Indoor Emitter Localization from 433MHz and 2.4GHz WLAN Received Signal Strengths

Hang Du

LONG-RANGE INDOOR EMITTER LOCALIZATION FROM 433MHZ AND 2.4GHZ WLAN RECEIVED SIGNAL STRENGTHS

A Thesis

Submitted to the Faculty

of

Rose-Hulman Institute of Technology

by

Hang Du

In Partial Fulfillment of the Requirements for the Degree

of

Master of Science in Electrical Engineering

May 2018

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Final Examination Report

Hang Du — Name

Electrical Engineering — Graduate Major

Thesis Title **Long-Range Indoor Emitter Localization from 433MHZ and WLAN Received Signal Strengths**

DATE OF EXAM: **April 27, 2018**

EXAMINATION COMMITTEE:

	Thesis Advisory Committee	Department
Thesis Advisor:	**JianJian Song**	**ECE**
	Mark Yoder	**ECE**
	Claude Anderson	**CSSE**

PASSED X **FAILED** _____

ABSTRACT

Hang Du

M.S.E.E.

Rose-Hulman Institute of Technology

May 2018

Long-Range Indoor Emitter Localization from 433MHz and 2.4GHz WLAN Received Signal Strengths

Thesis Advisor: Dr. Jianjian Song

An improved search method for localizing a radio emitter in a building from its signal strength is proposed and implemented. It starts from floor level determination, which samples the signal strength on each floor and determines the floor level of the emitter. Then the search is conducted iteratively on a specific floor. For each round of search, one-dimensional (1-D) or two-dimensional (2-D) signal strength is collected according to the actual structure of the floor. The signal strength data are processed to fit a 1-D curve or a 2-D surface with regression models to establish an indicator or trend, which can either locate the emitter or provide direction for the next round of search. The main contribution of this thesis is that the data processing results for 2-D signal strength data can locate the emitter or show the direction of the emitter through gradient, which is helpful to future search.

Our approach has been implemented with two wireless protocols: 433MHz protocol and 2.4GHz wireless local area network (WLAN) protocol. A 433MHz module with LoRa

modulation is chosen to provide long propagation distance. A 2.4GHz WLAN tester is used for close range search where 433MHz signal does not show enough attenuation spread to be effective.

433MHz implementation consists of an emitter, a radio tester and an Android APP on a smartphone. The emitter is a board with an Arduino Uno and a 433MHz transceiver. The radio tester is a board with an Arduino Uno, a 433MHz transceiver and a Bluetooth-to-serial module to communicate with a smartphone. The radio tester and the APP work together to localize the emitter.

2.4GHz WLAN implementation is composed of an emitter, which is emulated with a smartphone, a radio tester which consists of a smartphone, and a router and two Android APPs. Both phones are connected through the router and socket communication is initiated with the radio tester working as a server and the emitter working as a client. The APP on the emitter implements the client functions. The radio tester controls data acquisition process. The APP on the tester establishes the server functions and deals with received data. It compares signal strengths in different locations and finds the position that has the strongest signal strength to locate the emitter.

The innovative idea of this thesis is to use 1-D and 2-D signal strength with regression models as it is convenient to provide location or unique search direction of the emitter. 1-D data is processed with linear and polynomial regressions to fit curves in order to find possible location of the emitter in either a narrow strip or a half a plane. 2-D data is processed with multiple regressions to fit contour-line surfaces in order to find either location of the emitter on the top of a surface or a unique search direction of the location of the emitter as indicated by the highest surface gradient.

Our approach is compared with the centroid algorithm with an example. The centroid algorithm assumes the emitter is located in the search area and it is also easily influenced by sampling location biases. Our approach has two advantages over the centroid algorithm. The first advantage is that our approach can work even when the emitter is out of the initial search area since it searches iteratively. The second advantage is that when the emitter is in the initial search area, our approach is not influenced by sampling location biases.

To My Parents Chuanjiu Du (father) and Huilan Jin (mother)

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Song, for all the encouragement and instructions he provided during my Master of Science study and for guiding me to the conclusion of my study at Rose-Hulman. Dr. Song did a great job of encouraging me to pursue my interest and helping me to push my research further and further. I would also like to thank my other committee members, Drs. Claude Anderson and Drs. Mark Yoder. Their courses really impressed me and laid a solid basis for my thesis. Finally, I want to appreciate the support from my family and friends who give me lots of strength during my graduate study. I want to give special thanks to my parents who tried their best to give me the best education. Without them, I would not have the chance to go abroad, and graduate from Rose-Hulman Institute of Technology.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

TOA	Time of Arrival
AOA	Angle of Arrival
RSSI	Received Signal Strength Indicator
WLAN	Wireless Local Area Network
SPI	Serial Peripheral Interface
SNR	Signal-to-noise Ratio
AP	Access Point
MSE	Mean Square Error
GPS	Global Positioning System
1-D	One-dimensional
2-D	Two-dimensional

1. INTRODUCTION

Localization is to find the position or location (longitude and latitude) of a radio emitter by receiving and analyzing transmitted radio waveforms from the emitter. The emitter is an electronic device that radiates electromagnetic waves such as a cellphone, a radio station, a WiFi device, an access point, a 433MHz device, etc. Localization quest has been a challenge since the shortwave radio age in the 1920s. Localization technology has been developed and used mostly for defense_related systems [1]. The localization problem can be divided into two general categories: outdoor localization and indoor localization [2].

Various techniques have been developed and tested to identify the location of an emitter. Three types of information from the emitter have been investigated: radio signal strength [3], time of arrival (TOA) of radio signals [4], and angle of arrival (AOA) of radio signals [5]. A scheme for emitter indoor localization based on signal strength is proposed in this thesis, and it has been proved to work well in long-range indoor emitter localization.

1.1. Overview of Indoor Emitter Localization

Indoor localization research has been active in the past few years since indoor position information of an emitter is of great importance, especially due to the emerging Internet of Things [6, 7]. A wide range of services can be provided based on indoor localization, such as asset tracking [8] and navigation [9] in an airport or a shopping mall. Majority research on indoor localization over the decades can be divided into three categories: received radio signal strength analysis, Time of Arrival (TOA) of received radio signals, and Angle of Arrival (AOA) of received radio signals.

The main method in applying received signal strength for localization is to analyze fingerprints of access points of a wireless location area network (WLAN) such as Horus [10]. A WLAN fingerprint is signal strengths of all the access points (APs) at one location. This technique leverages on the existing WLAN access points or routers in a building and saves the cost for the specific infrastructure for indoor emitter localization. A WLAN fingerprint location system, as shown in Figure 1.1, works in two phases: an offline training phase and an online location determination phase. In the offline training phase, signals from APs in a test area are sampled at various locations of a small distance from each other, for example, three meters per sample point. For each sample point location, signal strengths from all the access points are recorded as the fingerprint at this location. Fingerprints for each sample point are saved into a database. In the online location determination phase, the emitter will collect signal strengths from all of the access points as the fingerprint of its location and will compare this fingerprint with the fingerprints of sample points in the database. The sample point in the database which has the minimum distance between its fingerprint and the emitter's fingerprint is chosen as the predicted position of the emitter.

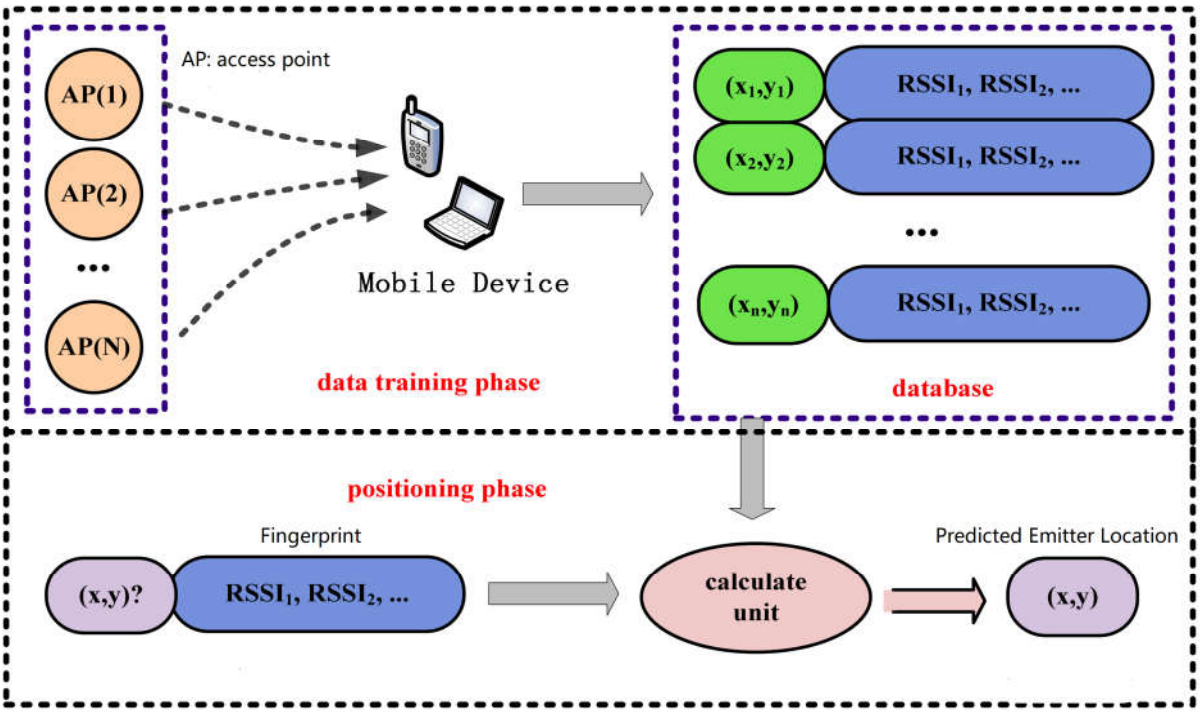

Figure 1.1 WLAN Fingerprint Method [11].

Another method of indoor emitter localization uses time of arrival (TOA) of radio signals from the emitter [4]. TOA is the propagation time of a radio signal from an emitter to a radio tester. By multiplying radio propagation time and radio propagation speed, the distance between the emitter and the radio tester can be calculated. One drawback of this approach is its high cost since the transmitter and receiver of a TOA system need to be highly synchronized to get accurate radio propagation time. Specialized hardware and techniques are required such as direct sequence spread-spectrum [12], and ultrawide band (UWB) [13, 14]. Another drawback of this approach is that it may suffer from multipath fading in a complex indoor environment.

The third method is to use Angle of Arrival. Angle of Arrival is defined as the direction of propagation of radio signals from the emitter incident on an antenna array. Since the arrival times of a radio signal on each element of the antenna array may vary, the time differences can be used to decide the direction of the emitter. The antenna array can also detect the direction of

the maximum energy of the radio signal from the emitter and hence the direction of the emitter. This solution can be quite accurate. However, a large antenna array is needed and can be expensive and bulky. A cheaper and more portable alternative approach to emulate the functionality of an antenna array is to position a cellphone in regular patterns [15].

Received radio signal strength is used in this thesis to identify the location of the emitter.

1.2. Radiation Pattern of Monopole Antenna and Energy Attenuation of Electromagnetic Waves

An antenna is an energy conversion device that can convert guided electromagnetic waves to free space waves and vice versa. The emitter's antenna used in our system is assumed to be a quarter-wave monopole antenna.

Figure 1.2 shows the radiation pattern of a monopole antenna in space. A monopole antenna radiates equal power in all azimuthal directions perpendicular to the antenna, but the radiated power varies with elevation angle, with the radiation dropping off to zero at the zenith, on the antenna axis [16]. This feature is used in Section 2.3 determine the floor level of the emitter.

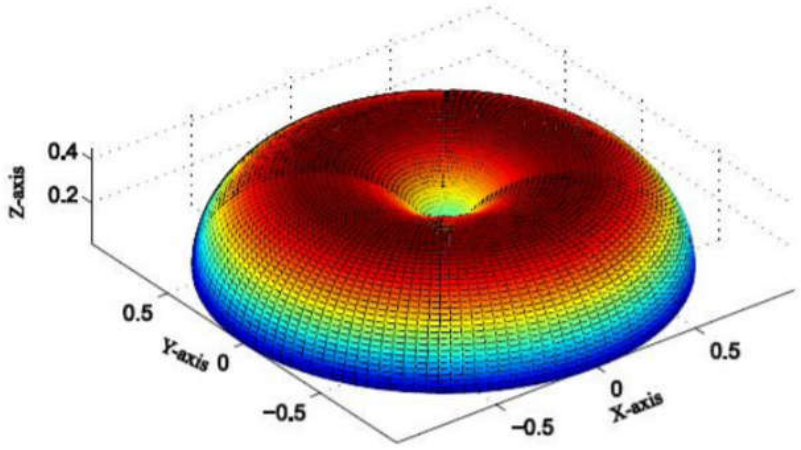

Figure 1.2 Radiation Pattern of a Monopole Antenna [17].

Since received radio signal strength from the emitter will be used to localize the emitter, it is important to understand radio signal energy propagation and attenuation in space. Figure 1.3 illustrates the directions of the electric field vector $\vec{E}$, magnetic field vector $\vec{H}$ and wave propagation of a typical planar wave. $\vec{E}$ and $\vec{H}$ are mutually perpendicular to each other. The direction of wave propagation is also the direction of energy propagation. This energy propagation can be described with power density vector or Poynting vector, $\vec{S} = \vec{E} \times \vec{H}$.

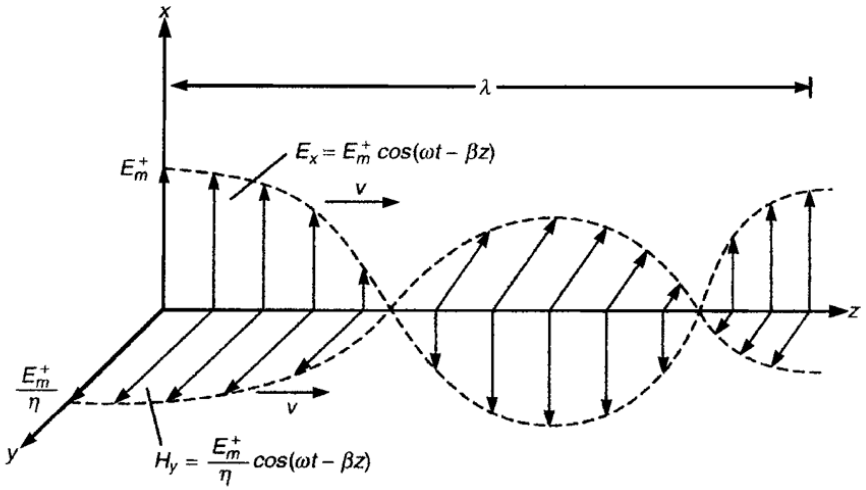

Figure 1.3 Electric and Magnetic Field Vectors of a Plane Wave [18].

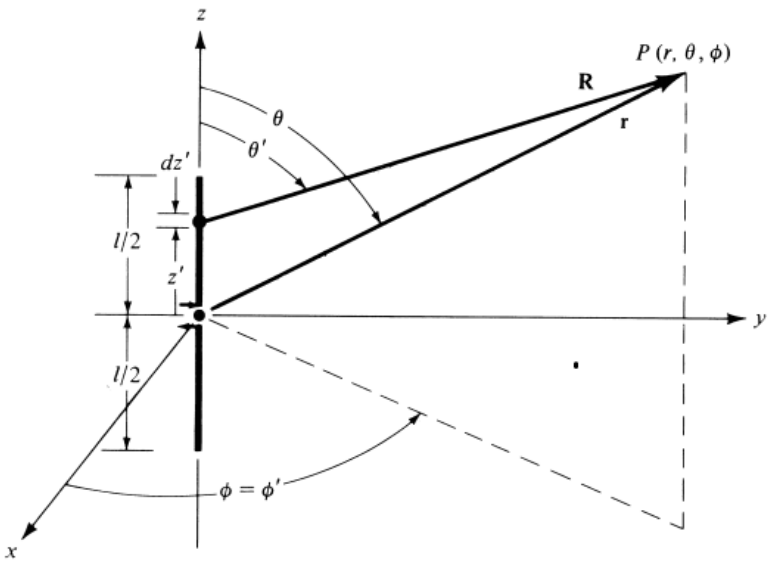

Figure 1.4 Finite Dipole Geometry [19].

Given the geometry of a finite dipole in Figure 1.4 [19], the equations for E_θ, H_ϕ and S_r are derived in [19]. The equation for E_θ is showed in (Eq. 1.1). The equation for H_ϕ is showed in (Eq. 1.2). S_r is the power density Poynting vector component in ρ direction [20]. A monopole antenna should have the same E_θ and H_ϕ like those of a dipole antenna. However, a monopole antenna can radiate only half of the power of the corresponding dipole. It can be seen from (Eq. 1.3) that the power density and therefore energy of an electromagnetic wave from a monopole antenna source is attenuated as $\frac{1}{r^2}$ with distance *r* from the source. This relationship is simplified to free-space path loss formula which is shown is (Eq. 1.4).

$$E_\theta \simeq j\eta \frac{I_0 e^{-jkr}}{2\pi r}\left[\frac{\cos(\frac{kl}{2}\cos\theta)-\cos(\frac{kl}{2})}{\sin\theta}\right] \quad \text{(Eq. 1.1)}$$

$$H_\phi \simeq \frac{E_\theta}{\eta} \simeq j\frac{I_0 e^{-jkr}}{2\pi r}\left[\frac{\cos(\frac{kl}{2}\cos\theta)-\cos(\frac{kl}{2})}{\sin\theta}\right] \quad \text{(Eq. 1.2)}$$

$$\vec{S} = \vec{E}\times\vec{H} = E_\theta\vec{a_\theta}\times H_\phi\vec{a_\phi} = -\eta\frac{{I_0}^2 e^{-2jkr}}{4\pi^2 r^2}\left[\frac{\cos(\frac{kl}{2}\cos\theta)-\cos(\frac{kl}{2})}{\sin\theta}\right]^2\vec{a_r} \quad \text{(Eq. 1.3)}$$

This relationship of power attenuation is expressed as the ratio of transmitted and received powers as shown is (Eq. 1.4), where Pt is signal power at transmitting antenna, Pr is signal power at receiving antenna, λ is wavelength, d is propagation distance between two antennas and c is the speed of light in free space.

$$\frac{Pt}{\mathrm{Pr}} = \frac{(4\pi d)^2}{\lambda^2} = \frac{(4\pi fd)^2}{c^2} \quad \text{(Eq. 1.4)}$$

Equation (Eq. 1.4) is plotted in Figure 1.5 which looks like a mountain. Radiated power attenuation in one dimension is plotted in Figure 1.6. The concept of radiated energy attenuation of electromagnetic waves is heavily utilized in Section 2.4 and 2.5.

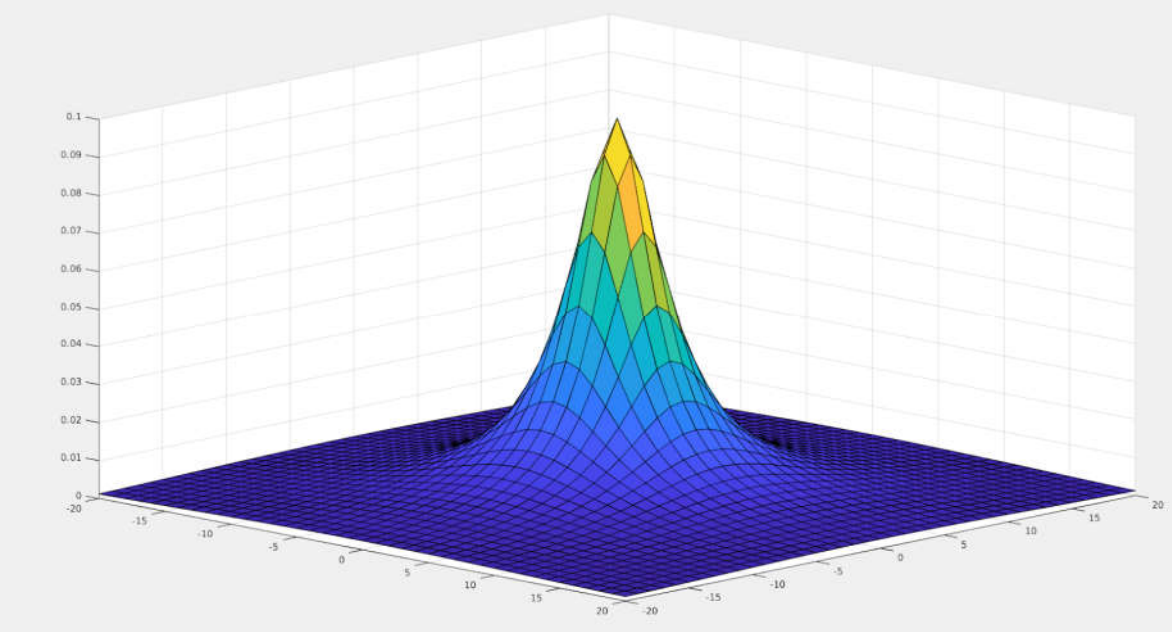

Figure 1.5 Radiated Power Attenuation in 2-D Free Space.

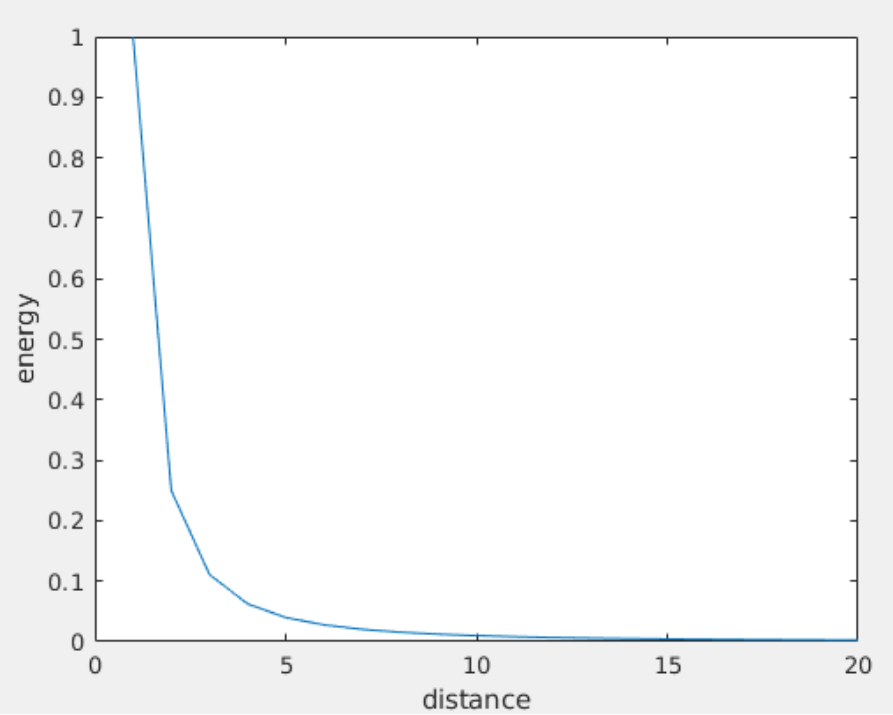

Figure 1.6 Radiated Power Attenuation in 1-D Free Space.

1.3. Our Proposed Scheme for Indoor Emitter Localization

The paper proposes a new scheme for indoor emitter localization with both 433MHz and 2.4GHz WLAN signal strengths. A 433MHz tester is used to receive the 433MHz radio signal from the emitter to take advantage of long-distance transmission of the 433MHz signal. Assume the building where the emitter resides is known, the 433MHz tester can identify the floor in the building where the emitter is and zoom a small area on the floor. 2.4GHz WLAN signals are used when there is some difficulty to find the emitter in small range in the final stage, where the 433MHz signal does not have enough attenuation to be effective. WLAN signals are used to screen the small area to identify the location of the emitter. This approach takes advantages of low attenuation of 433MHz transmission and high density and high attenuation of 2.4GHz WLAN signals.

Our approach has been implemented with two wireless protocols: 433MHz protocol and 2.4GHz wireless local area network (WLAN) protocol. A 433MHz module with LoRa modulation is chosen to provide long wave propagation distance. LoRa is a technology which can provide longer range transmission for radio signals. A WLAN tester is used for close range search where the 433MHz signal does show enough attenuation spread to be effective.

The 433MHz implementation consists of an emitter, a radio tester and an Android APP on a smartphone. The emitter is a board with an Arduino Uno and a 433MHz transceiver. The radio tester is a board with an Arduino Uno, a 433MHz transceiver and a Bluetooth-to-serial module to be connected to the smartphone. The radio tester and the APP work together to localize the emitter. To start collecting signal strength from the emitter, the Android APP sends commands to the radio tester through the Bluetooth module. Then the radio tester sends out a LoRa packet. After the emitter receives the LoRa packet, it sends a LoRa packet back. The radio

tester, after receiving the LoRa packet, fetches signal strength data and sends the data to the APP. After the APP finishes data collection, the data is copied to a laptop and is processed with regression models in MATLAB.

The 2.4GHz WLAN implementation is composed of an emitter, which is emulated with a smartphone, a radio tester which consists of a smartphone and a router and two Android APPs. Both phones are connected through the router and socket communication is initiated with the radio tester working as a server and the emitter working as a client. The APP on the emitter implements the client functions. The radio tester controls data acquisition process. The APP on the tester establishes the server functions and deals with received data. It compares signal strengths in different locations and finds the position that has the strongest signal strength to locate the emitter.

The system architecture of our approach is given in Figure 1.7. Localization of the emitter inside a building is done by two phases: vertical floor determination and horizontal location determination on the floor. For vertical floor determination, received signal strengths differences from 433MHz signal strengths in different floors are used to determine the floor where the emitter resides. Once the floor is identified, different search strategies are applied according to the actual structure of the floor. For a narrow path such as a hallway, 1-D localization is used. Otherwise, 2-D localization is used. Several rounds of searches are carried out until the emitter location is identified. Each round of search can provide some hints of direction for the next round of search.

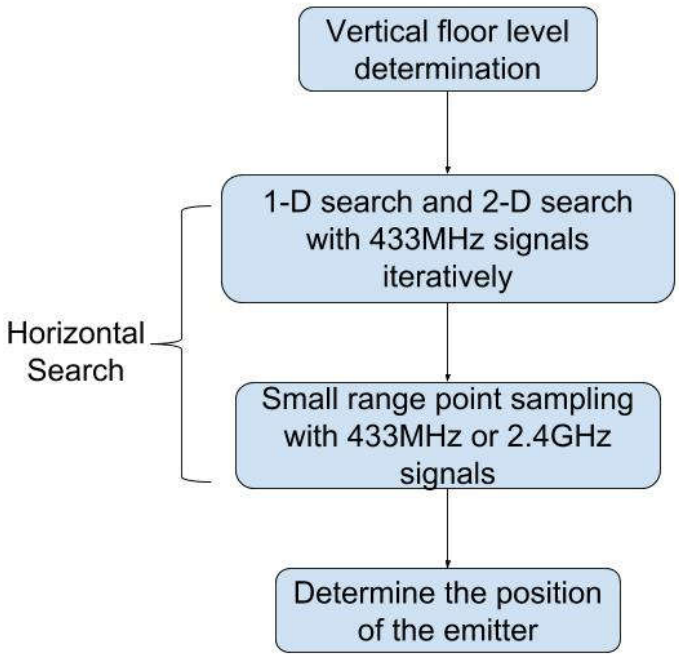

Figure 1.7 System Architecture of Our Proposed Indoor Emitter Localization Approach.

2. DESIGN AND IMPLEMENTATION OF THE 433MHZ METHOD

The 433MHz radio signal is used for two reasons. One reason is that 433.050 - 434.090 MHz band could be used without a license in many countries such as all European countries and some Asian countries. The other reason is that 433MHz signal can travel through walls and obstacles, which is good for long-range emitter localization because of its longer wavelength or lower frequency than those of WLAN signals.

Section 2.1 introduces Taylor series and math models used in the localization algorithms. Section 2.2 discusses design details of the 433MHz method, including the hardware and the software. Section 2.3, 2.4 and 2.5 present the localization algorithms. Section 2.6 discusses the performance of the localization algorithms.

2.1. Taylor Series, Linear Regression, Polynomial Regression and Multiple Regression

Radio signals in an indoor environment can be noisy and distorted because of irregular radiation paths, complex attenuation patterns and multiple reflections of the signals. Received signals from a 433MHz emitter need to be processed to remove noises for ease of pattern recognition and trend identification. We propose to process the received signal strengths and position data with linear regression, polynomial regression, and multiple regression.

Linear regression is a linear approach for modeling the relationship between a scalar dependent variable and an explanatory variable denoted as x [21]. Equation (Eq. 2.1) shows one explanatory variable case, which is used in Section 2.4.1 to process 1-D signal strength data.

$$f(x) = a \times x + b \quad \text{(Eq. 2.1)}$$

Polynomial regression is a form of regression analysis in which the relationship between the independent variable x and the dependent variable is modeled as an n^{th} degree polynomial in x [22]. Equation (Eq. 2.2) shows the second-degree polynomial regression, which is used in Section 2.4.2 to process 1-D signal strength data.

$$f(x) = a \times x^2 + b \times x + c \quad \text{(Eq. 2.2)}$$

Multiple regression is a form of regression analysis in which a scalar dependent variable and multiple explanatory variables are involved. Equation (Eq. 2.3) shows the x's degree=1 and y's degree=1 case. Equation (Eq. 2.4) shows the x's degree=2 and y's degree=1 case. Equation (Eq. 2.5) shows the x's degree=1 and y's degree=2 case. Equation (Eq. 2.6) the x's degree=2 and y's degree=2 case. These equations are used in Section 2.5 to process 2-D signal strength data.

$$f(x,y) = p00 + p10 \times x + p01 \times y \quad \text{(Eq. 2.3)}$$
$$f(x,y) = p00 + p10 \times x + p01 \times y + p20 \times x^2 + p11 \times xy \quad \text{(Eq. 2.4)}$$
$$f(x,y) = p00 + p10 \times x + p01 \times y + p11 \times xy + p02 \times y^2 \quad \text{(Eq. 2.5)}$$
$$f(x,y) = p00 + p10 \times x + p01 \times y + p20 \times x^2 + p11 \times x \times y + p02 \times y^2 \quad \text{(Eq. 2.6)}$$

A Taylor series is a representation of a function as an infinite sum of terms that are calculated from the values of the function's derivatives at a single point [23]. A function can be approximated by using a finite number of terms of its Taylor series as shown in Equation (Eq. 2.7). The polynomial expression formed by taking some initial terms of the Taylor series is called a Taylor polynomial. The Taylor series can also be generalized to functions of more than one variable. A second-order Taylor series can be used to approximate any function that depends on two variables. The fact that any function can be approximated by using a finite number of

terms of its Taylor series is the reason of why the mathematical models of linear regression, polynomial regression, and multiple regression are used for fitting data in the proposed localization algorithms.

$$f(x) \approx \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n \qquad \text{(Eq. 2.7)}$$

$f^{(n)}(a)$ ：nth derivative of f evaluated at the point a.

2.2. Design of the 433MHz Method

The design of the 433MHz method includes two parts: hardware and software. As shown in Figure 2.1, the hardware part includes the emitter and the radio tester that receives signals from the emitter. The software part includes an Android application for interfacing with the tester and MATLAB code for data processing.

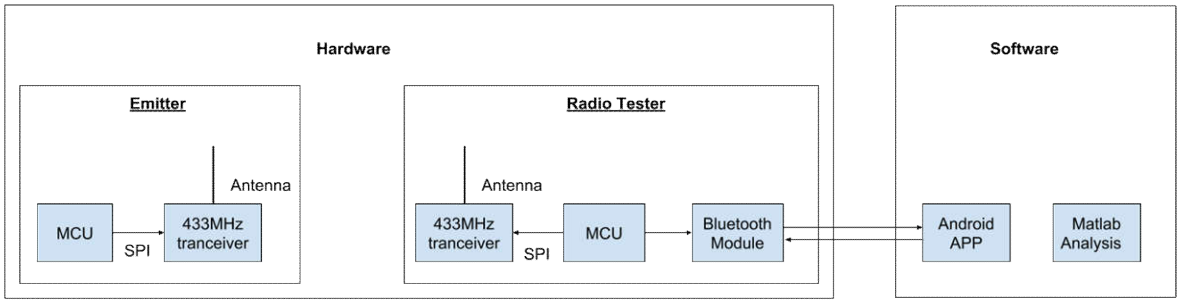

Figure 2.1 433MHz Method Design.

2.2.1. Revised Signal Strength Data

In the 433MHz method, strength data of 433MHz signals with LoRa modulation is collected. LoRa is a proprietary, chirp spread spectrum (CSS) radio modulation technology for a low-power wide-area network from Semtech which uses a license-free sub-gigahertz radio

frequency band [24]. It offers long range and low power data transmission. According to [25], it is possible to receive packets below the noise floor due to the nature of the LoRa modulation. In this situation, it is more accurate to use the signal-to-noise ratio (SNR) in conjunction with the packet RSSI to compute the signal strength of the received packet. We call this revised signal strength, and it is computed with Equations (Eq. 2.10) and (Eq. 2.11). SNR is defined as the ratio of signal power to the noise power as showed in Equation (Eq. 2.8). Equation (Eq. 2.9) shows how SNR is expressed in logarithmic decibel scale.

$$SNR = \frac{P_{signal}}{P_{noise}} \quad \text{(Eq. 2.8)}$$

$$SNR_{dB} = 10\log_{10}\left(\frac{P_{signal}}{P_{noise}}\right) \quad \text{(Eq. 2.9)}$$

$$\text{Re}visedRSSI = PacketRssi \ (\text{when} \ SNR_{dB} >= 0\ ) \quad \text{(Eq. 2.10)}$$

$$\text{Re}visedRSSI = PacketRssi + PacketSnr \times 0.25\,(\text{when} \ \ SNR_{dB} < 0\ ) \quad \text{(Eq. 2.11)}$$

2.2.2. Emitter and Tester Hardware Modules

As shown in Figure 2.1, hardware modules include two microcontrollers (MCU), two 433MHz transceivers with antennas, and one Bluetooth module.

Arduino Uno boards are used as the microcontroller boards to construct both the emitter and the radio tester. Arduino boards use a variety of microprocessors and controllers equipped with sets of digital and analog input/output pins to interface with other boards [26]. There are also many software libraries available for the Arduino, which makes it easy to build a quick software prototype. RadioHead library is used for radio communications between the Arduino and the 433MHz transceiver [27].

The RFM96 LoRa Radio module from Adafruit is chosen to be used for the 433MHz transceiver as showed in Figure 2.2. It is an SX1276 LoRa based module with serial peripheral

interface (SPI) interface. The power output capacity of the RFM96 can be up to 20dBm or 100mW. The RFM96 has a range of about 2km line of sight with tuned unidirectional antennas [28]. Long wireless transmission distance is the main advantage of this module. This is because SX1276 features the LoRa long range modem that provides an ultra-long range spread spectrum from Semtech. The RFM96 interfaces with an Arduino Uno through SPI interface.

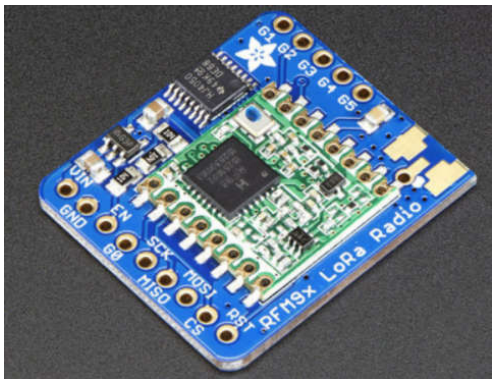

Figure 2.2 The RFM96 433MHz Transceiver [28].

As for the Bluetooth part, the HC-06 Bluetooth to serial module as shown in Figure 2.3 is chosen.

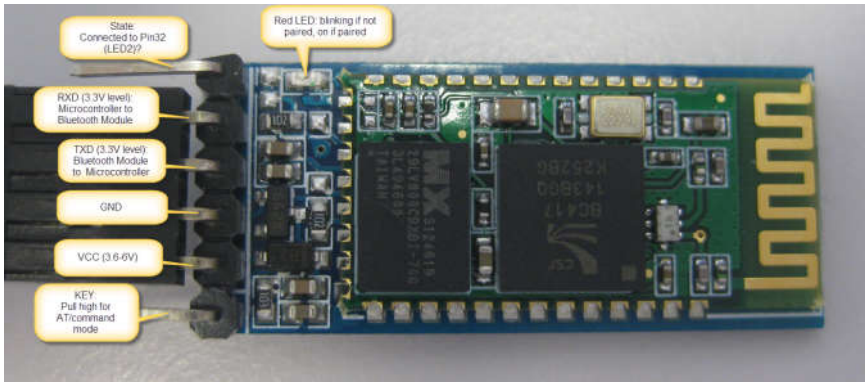

Figure 2.3 The HC-06 Bluetooth to Serial Module [29].

A wire antenna as a quarter-wave monopole antenna is used for the RFM96 on both the emitter and the radio tester. The length of the antenna is calculated by $\frac{1}{4}\times\lambda=\frac{1}{4}\times\frac{c}{f}=17.3cm$, where c=3x$10^8$m/s, f=433MHz.

The main hardware operation is the two-way communications between the emitter and the radio tester. To obtain a sample of signal strength data, the following steps are carried out:

(1) The radio tester sends one LoRa packet out.

(2) The emitter receives this packet and then sends one LoRa packet back.

(3) The radio tester receives this packet and then fetches RSSI data.

Software flowcharts for the radio tester and the emitter are presented in Figure 2.4 and Figure 2.5. The schematics of the radio tester and the emitter are displayed in Figure 2.6 and Figure 2.7. Photos of the radio tester and the emitter are shown in Figure 2.8 and Figure 2.9.

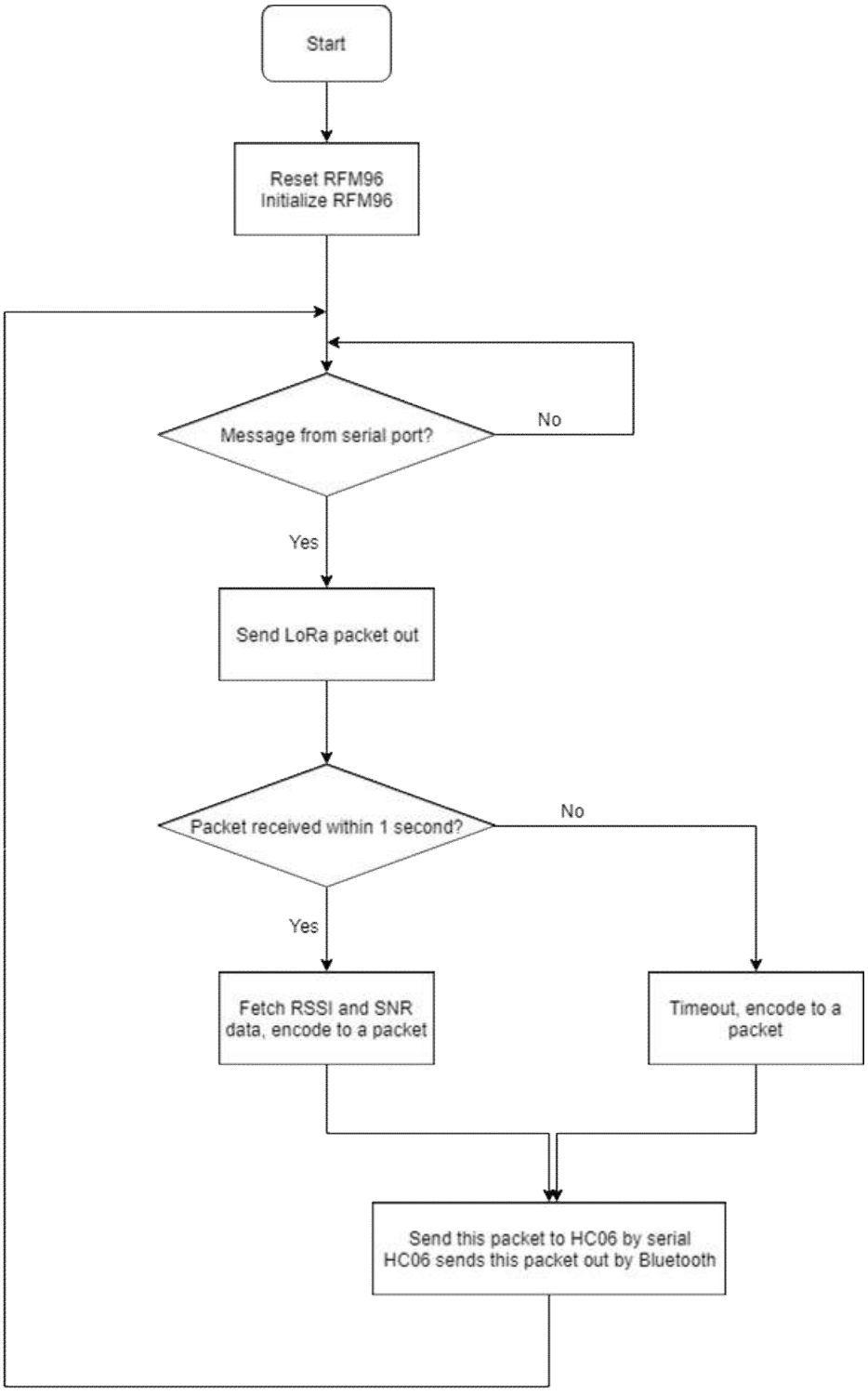

Figure 2.4 Software Flowchart for the 433MHz Radio Tester.

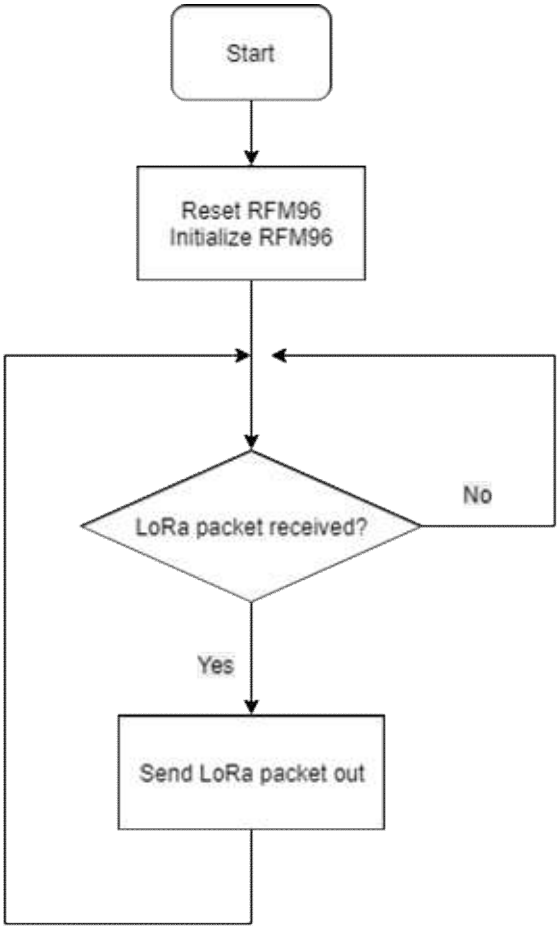

Figure 2.5 Software Flowchart for the 433MHz Emitter.

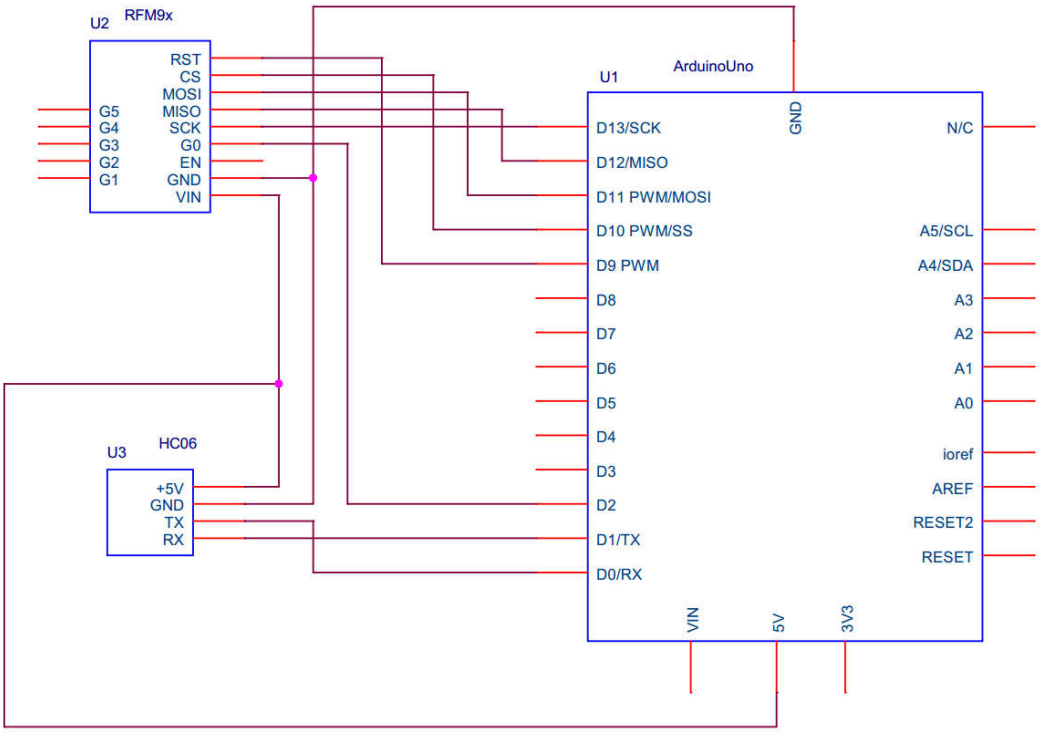

Figure 2.6 The Schematic of the 433MHz Radio Tester.

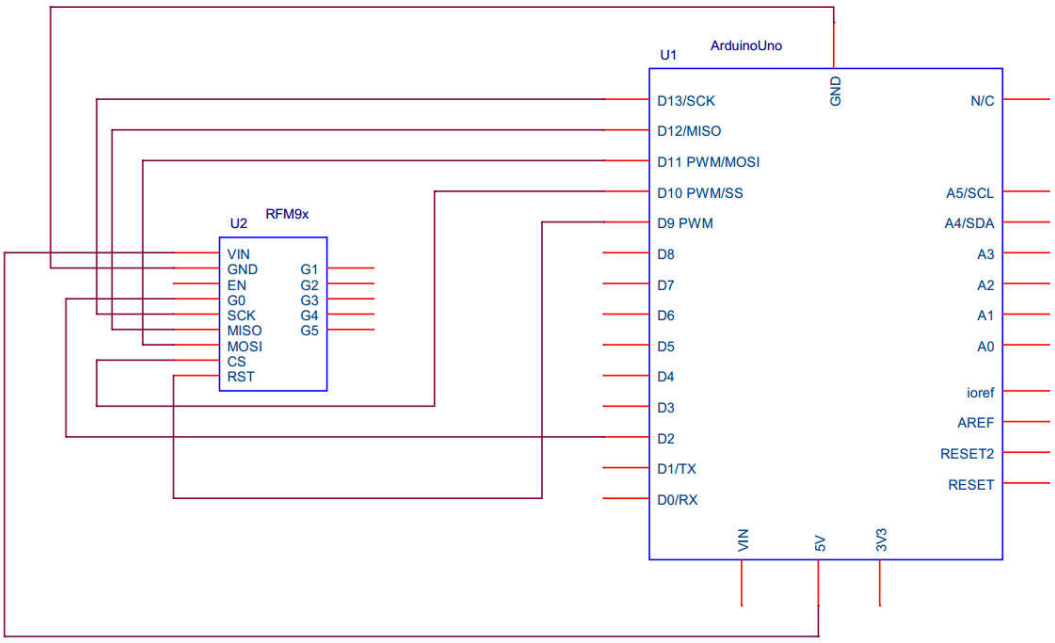

Figure 2.7 The Schematic of the 433MHz Emitter.

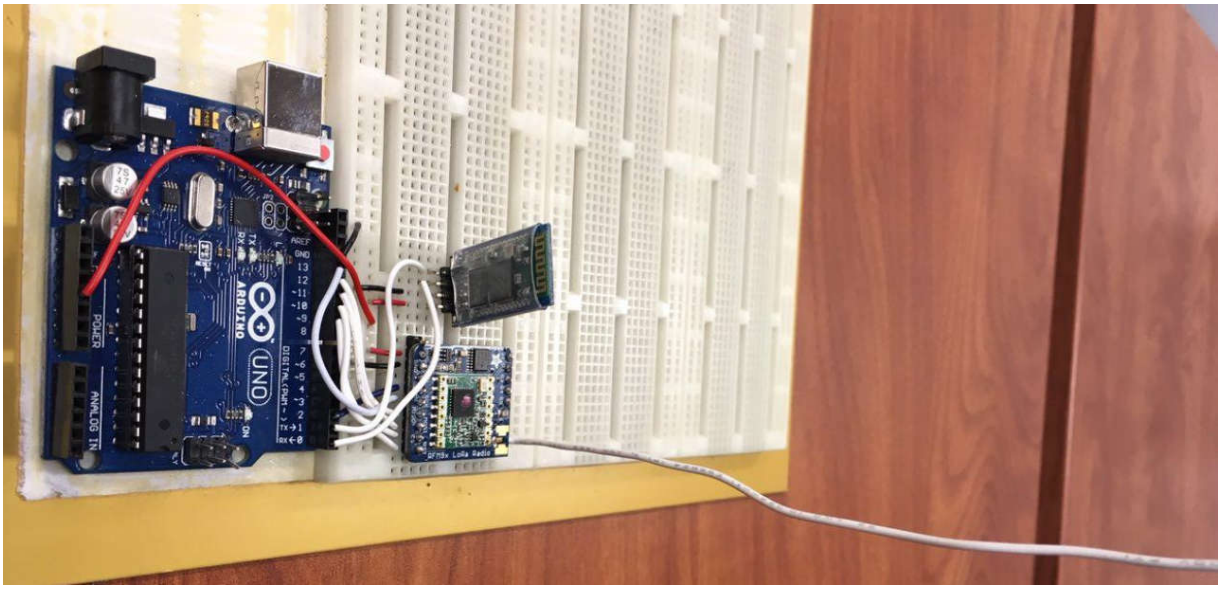

Figure 2.8 The 433MHz Radio Tester Board.

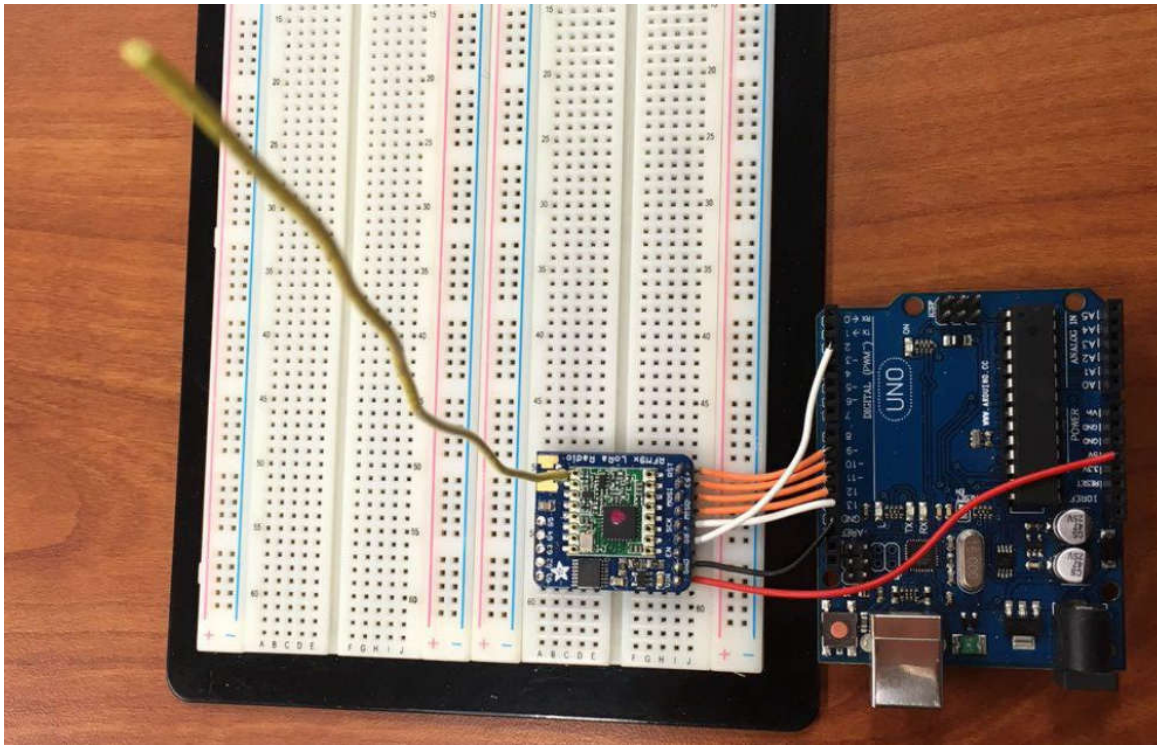

Figure 2.9 The 433MHz Emitter Board.

2.2.3. Android APP for Interfacing with the 433MHz Tester

The APP running on an Android device aims at sending commands to the radio tester and receiving signal strength data from the radio tester. Then the data is moved from the Android APP to a laptop to be processed with the math models discussed in Section 2.1 that are implemented in MATLAB.

Figure 2.10 describes the workflow of the Android APP to interface with the radio tester. The first main entry point of the APP is to make a Bluetooth connection with the radio tester. The HC-06 Bluetooth module can cover a distance of about nine meters. It can be assumed that the distance between the radio tester and the Android phone is within a reasonable range to make Bluetooth connection successfully since both the radio tester and the phone must be held by the same engineer at work. After this step of Bluetooth connection, the APP can go to the search mode in which RSSI data for each position is collected. To obtain a sample of data at a location, the following steps will be carried out:

(1) APP sends a command to the radio tester by Bluetooth.
(2) After receiving the command, the radio tester sends one LoRa packet out to the emitter.
(3) The emitter receives this LoRa packet and then sends one LoRa packet back to the radio tester.
(4) The radio tester receives this LoRa packet, fetches RSSI data and encodes data to a packet.
(5) The radio tester sends this packet back to the APP by Bluetooth.

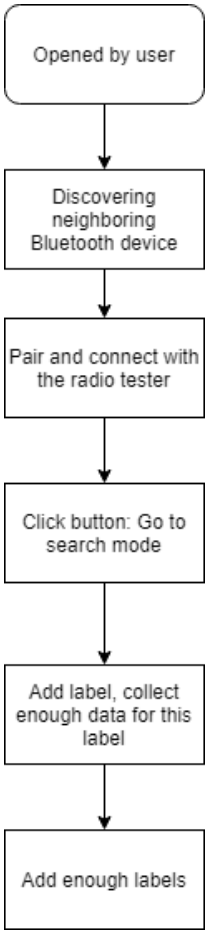

Figure 2.10 Software Flowchart of the Android APP to Interface with the 433MHz Tester.

Figure 2.11 shows the main page of the Android APP which is in charge of making a Bluetooth connection with the radio tester. From the main page, the APP can go to 1-D data sampling in Figure 2.12 or 2-D data sampling in Figure 2.13. Regression analysis may be applied in Figure 2.14 after 1-D data sampling.

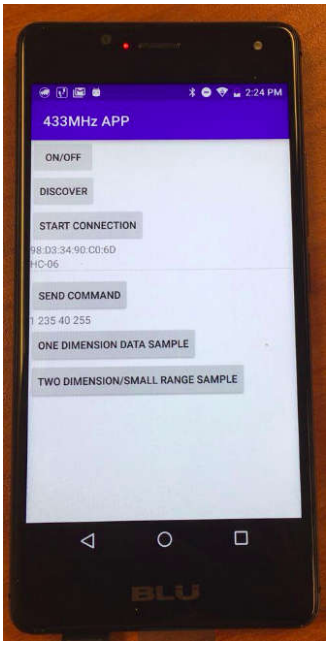

Figure 2.11 Making Bluetooth Connection with the 433MHz Radio Tester.

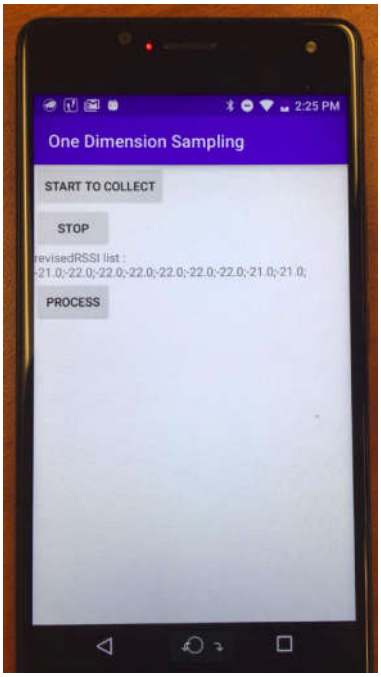

Figure 2.12 1-D Data Sampling

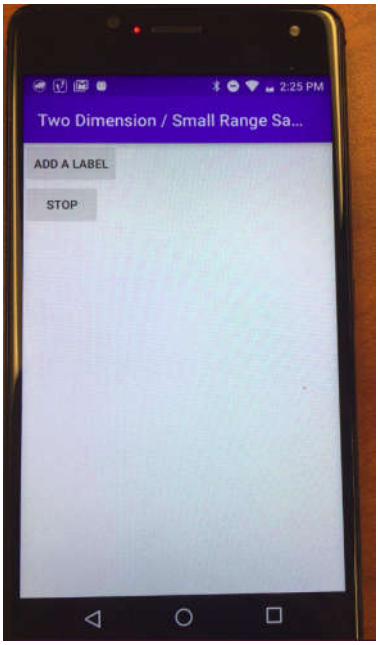

Figure 2.13 2-D Data Sampling.

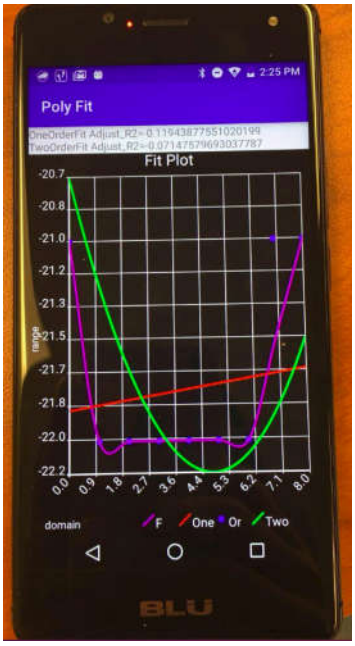

Figure 2.14 1-D Data Regression Analysis.

Figure 2.15 describes the 1-D data sampling process using the APP. For 1-D data sampling, only one signal strength is collected for each position. The user is expected to walk at constant speed. During this process, the RSSI data is sampled every two seconds.

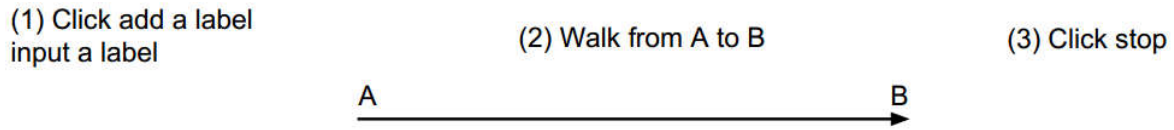

Figure 2.15 1-D Data Sampling Process.

Figure 2.16 describes the 2-D data sampling process using the APP. For 2-D data sampling, multiple signal strengths are collected for each position to be averaged to increase accuracy. The grid in Figure 2.16 is composed of four paths: from point A1 to point B1, A2 to B2, A3 to B3 and A4 to B4. For each path, sampling point locations which are represented as red points are spaced equally. For each location, the user clicks the button of adding a label and inputting a label, waits for some time to collect enough data for averaging and then clicks the Stop button. Then the averaged data for the position is saved to the APP. When the data sampling for all the locations in Figure 2.16 is completed, the 2-D data sampling process is completed.

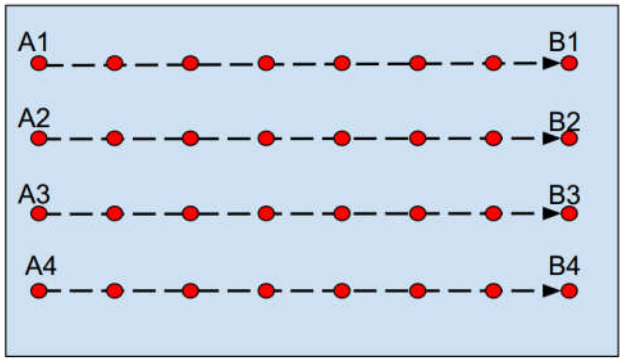

Figure 2.16 2-D Data Sampling Process.

2.3. Determination of Floor Level of the Emitter

The objective of the paper is to locate the emitter in a building. Since a building usually has several floors, the floor level where the emitter resides needs to be determined first. The algorithm for floor level determination is quite simple. Signal strength data is collected on different floors where the stairs are. For example, there are three stairs on the Second Floor of Moench Hall of our school as showed in Figure 2.17 and signal strength data is collected at position (1), (2) and (3) on each floor. Multiple signal strengths are collected at each location to be averaged to increase accuracy. To determine the floor level of the emitter, the signal strength data for each floor at different stairs are averaged to represent the signal strength for this floor. Assume there are in total m stairs, the equation to average the signal strengths of a floor is as follows where $s[i][j]$ is the signal strength at the ith stair on the jth floor.

$$floor = \max_{j}\{\frac{1}{m}\sum_{i=1}^{m} s[i][j]\} \qquad \text{(Eq. 2.12)}$$

Two experiments are designed to show the validity of this algorithm. In the first experiment, the emitter is placed in Room D210 on the 2nd Floor of Moench Hall in Figure 2.17, and the algorithm predicts that the emitter is on the second floor according to signal strength data in Table 2.1. In the second experiment, the emitter is placed in D101 on the 1st Floor of Moench Hall as shown in Figure 2.18 and the algorithm again makes the right prediction according to Table 2.2.

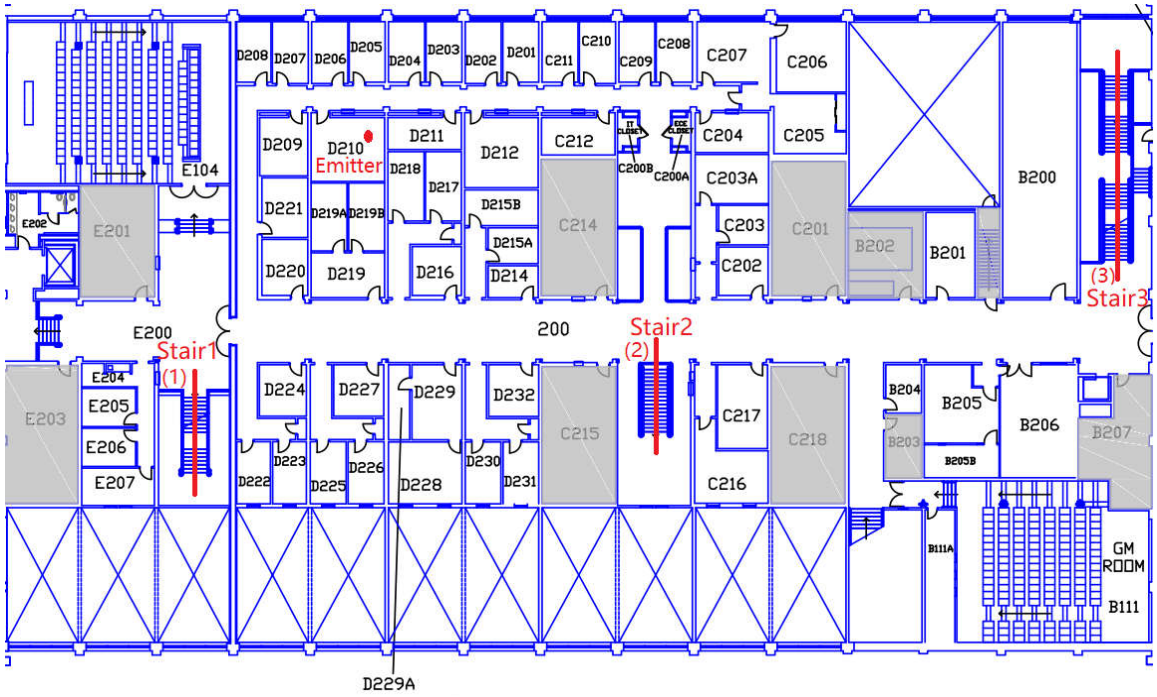

Figure 2.17 Three Stairs and the Emitter in Room D210 on the Second Floor of Moench Hall.

Table 2.1 Floor Level Signal Strength Data When the Emitter is in D210.

	Stair 1	Stair 2	Stair 3	Average
Floor -1	-64.97dB	-63.33dB	-72.47dB	-66.92dB
Floor 1	-53.9dB	-50.43dB	-71.21dB	-58.51dB
Floor 2	-49.5dB	-56.4dB	-62.83dB	-56.24dB

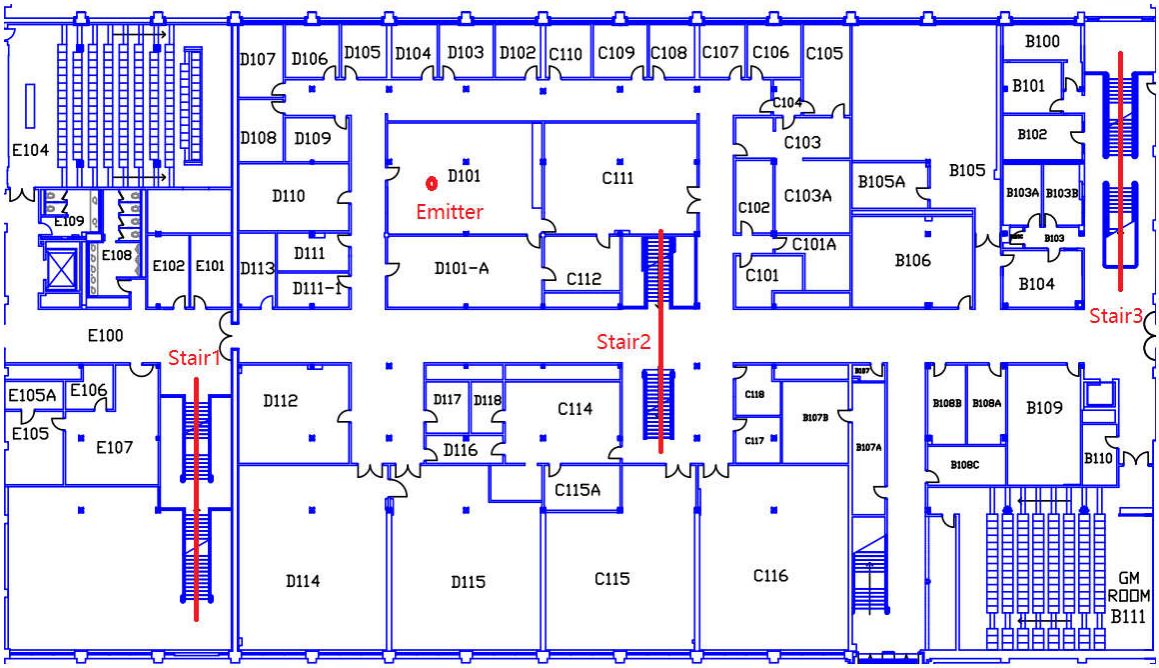

Figure 2.18 The Emitter in Room D101 on the First Floor of Moench Hall.

Table 2.2 Floor Level Signal Strength Data When the Emitter is in D101.

	Stair 1	Stair 2	Stair 3	Average
Floor -1	-49.97dB	-40.24dB	-65.0dB	-51.74dB
Floor 1	-39.89dB	-40.73dB	-54.27dB	-44.96dB
Floor 2	-46.03dB	-47.5dB	-61.7dB	-51.74dB

2.4. Emitter Localization from 1-D Data with Linear Regression and Polynomial Regression

1-D data means the sequential signal strengths from a straight line. The data is processed with a program in MATLAB as shown in Appendix B. Wherever the emitter is, there are three possible cases for the emitter:

Case I: The radio tester moves away the emitter in Figure 2.19.

Case II: The radio tester approaches the emitter in Figure 2.21.

Case III: The radio tester passes through the emitter in Figure 2.23.

In both Case I and Case II, received signal strength data is changing monotonically with distance. Two experiments are designed to verify our approach. The first experiment is described in Figure 2.19. Signal strength vs. relative distance for this experiment is plotted as shown in Figure 2.20. The horizontal axis in Figure 2.20 is relative distance represented by the index of the sequential signal strength data. The initial position of the radio tester is the origin of the horizontal axis. When the radio tester is moving at a constant speed, signal strength is collected at a fixed frequency, and these signal strengths are indexed sequentially starting from index 0. Four legends are showed in Figure 2.20. “Sampled data” means the scatter plot of the original signal strength data. “Filter plot” means average filtering [35] is applied to the original data and a plot is made from the filtered data. “Linear fit” and “2nd-order fit” are obtained by linear regression from Equation (Eq. 2.1) and polynomial regression Equation (Eq. 2.2) respectively.

From Figure 2.20, it is observed that the original signal strength data can be quite messy. After averaging filtering with window size=2, the plot for filtered data is much smoother. The linear fitting result is $f(x) = 0.9417x - 21.7557$ while the 2nd-order fitting result is $f(x) = 0.0189x^2 - 1.6989x - 16.5816$ with the axis of symmetry $x = 44.87$. In this case, both linear fit and 2nd-order fit show the same trends.

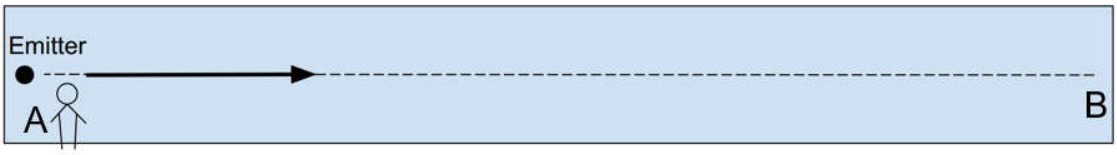

Figure 2.19 Case I: the Radio Tester Moves Away from the Emitter.

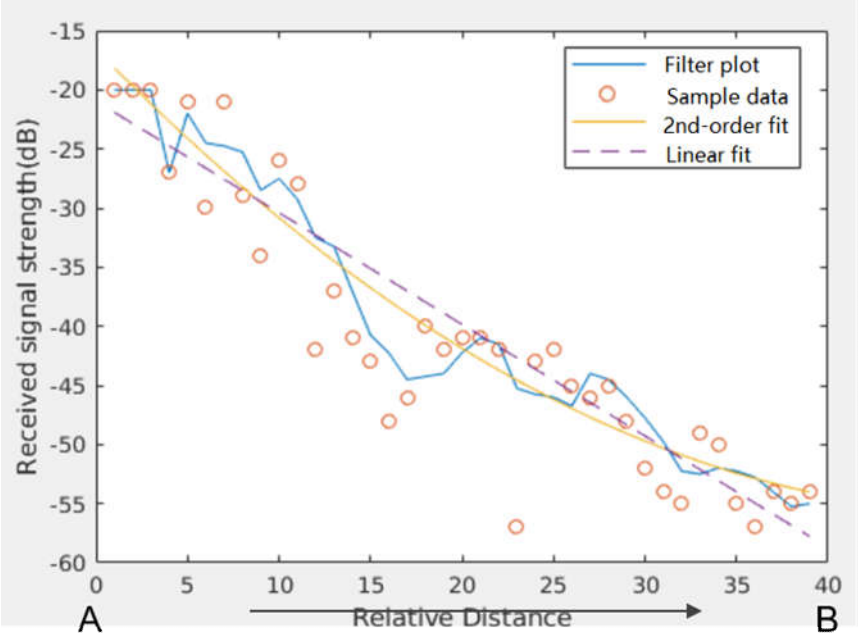

Figure 2.20 1-D Plot of Signal Strength vs. Distance for Case I.

Figure 2.21 shows one example of Case II. The fitting result is illustrated in Figure 2.22. The Linear fitting result is $f(x) = 0.9053x - 58.5346$ while the 2nd-order fitting result is $f(x) = 0.0129x^2 + 0.3753x - 54.8246$ with the axis of symmetry $x = -14.518$. In this case, both linear fit and 2nd-order fit present the same trend.

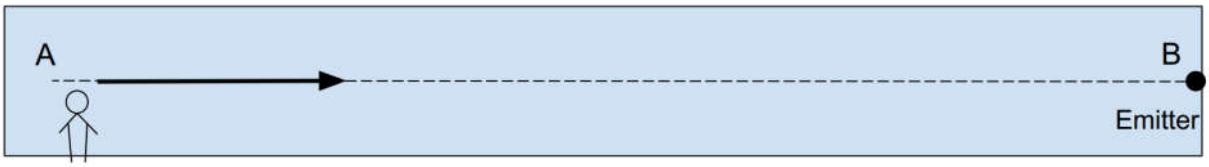

Figure 2.21 Case II: the Radio Tester Approaches the Emitter.

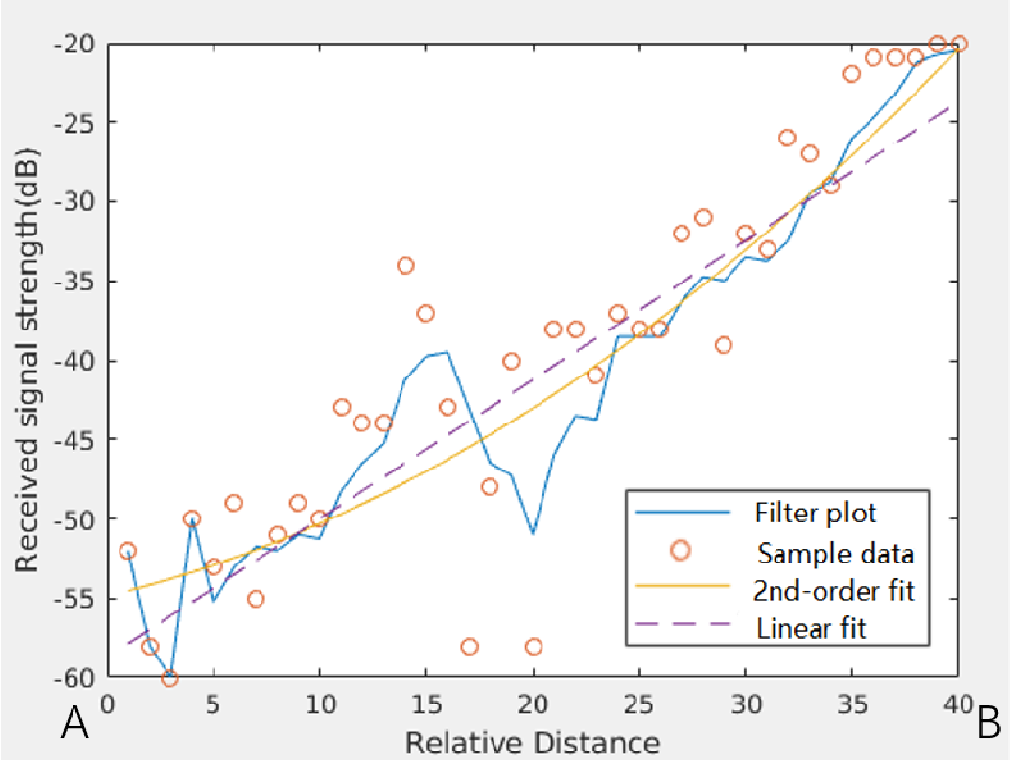

Figure 2.22 1-D Plot of Signal Strength vs. Distance for Case II.

In Case III where the radio tester passes through the emitter, received signal strength data no longer changes monotonically with distance. The experiment is designed in Figure 2.23, and the fitting result is in Figure 2.24. The linear fitting result is $f(x) = -0.3127x - 32.018$. The 2nd-order fitting result is $f(x) = -0.0705x^2 + 2.0853x - 46.0064$ with the axis of symmetry $x = 14.7834$ .In this case, linear fit and 2nd-order fit show different trends.

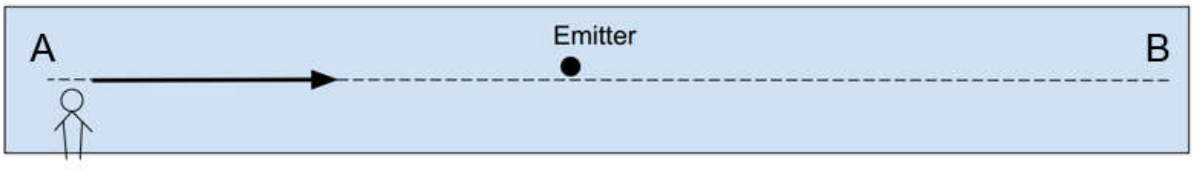

Figure 2.23 Case III: the Radio Tester Passes through the Emitter.

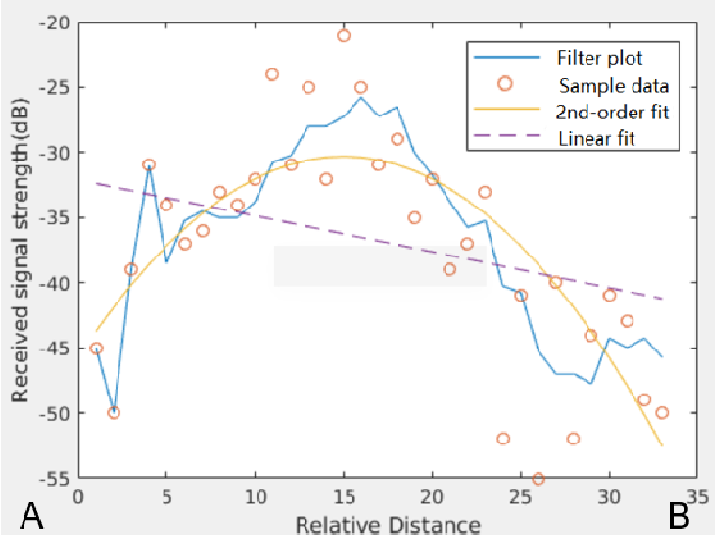

Figure 2.24 1-D Plot of Signal Strength vs. Distance for Case III.

Here comes a problem. When linear fit and 2nd-order fit show different trends, which result should we choose to trust? Adjusted R-squared can help to make this choice. First, we will introduce R-squared which is also called coefficient of determination. It is the proportion of the variation in the dependent variable that is predictable from the independent variables in the regression model [30]. The larger the R-squared is, the more variability is explained by the regression model. However, it can sometimes be misleading since more terms in the regression model can always result in better R-squared. The adjusted R-squared is designed to address this problem. The adjusted R-squared is a modified version of R-squared that has been adjusted for the number of predictors in the model. The adjusted R-squared increases only if the new term improves the model more than would be expected by chance [30]. It decreases when a predictor

improves the model by less than expected by chance [30]. The equations for the calculation for R-squared and adjusted R-squared are shown in (Eq. 2.16) and (Eq. 2.17). The conclusion is that the fitting result of the regression model with higher adjusted R-squared should be trusted.

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \quad \text{(Eq. 2.13)}$$

$$SS_{tot} = \sum_i (y_i - \bar{y})^2 \quad \text{(Eq. 2.14)}$$

$$SS_{res} = \sum_i (y_i - f_i)^2 \quad \text{(Eq. 2.15)}$$

$$R^2 \equiv 1 - \frac{SS_{res}}{SS_{tot}} \quad \text{(Eq. 2.16)}$$

$$R^2_{adj} = 1 - (\frac{n-1}{n-p})\frac{SS_{res}}{SS_{tot}} \quad \text{(Eq. 2.17)}$$

Let's apply the above conclusion to case III. Adjusted R-squared for linear fit is 0.1188 while the adjusted R-squared for 2nd-order fit is 0.6641. Since 2nd-order fit has higher adjusted R-squared value, 2nd-order fitting result is chosen to extract trend which is moving toward the emitter and then moving away.

2.5. Emitter Localization from 2-D Data with Multiple Regression and Gradient

2-D data are signal strengths collected from an area. These signal strength data with their locations in (x, y) coordinates are processed with the multiple regression program in MATLAB to fit a surface. There are in total four regression models to be considered as shown in Section 2.1 from (Eq. 2.3) to (Eq. 2.6). As for how to choose among these regression models, it is the same with Section 2.4 that fitting result with the highest adjusted R-squared value is chosen. The

main idea is to find the emitter direction from the fitting surface according to its gradient descent.

A simple experiment called Case 2D-I is designed on an outdoor playground in Figure 2.25 to verify this idea of gradient surface. The seven dashed lines in Figure 2.25 are the walking paths to collect signal strength data. The data is fit to a surface with the multiple regression model in Equation (Eq. 2.6) and Figure 2.26 shows the fitting result which looks like the half of a mountain. Figure 2.27 is the top view of Figure 2.26. From color blue to yellow, the signal strength is becoming stronger. It is very easy to figure out the direction of the emitter from Figure 2.27 as shown by three arrows.

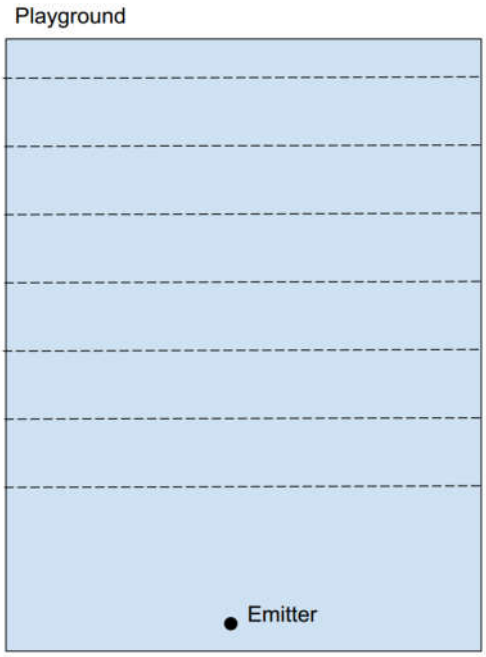

Figure 2.25 Case 2D-I: The Emitter is Placed on an Outdoor Playground.

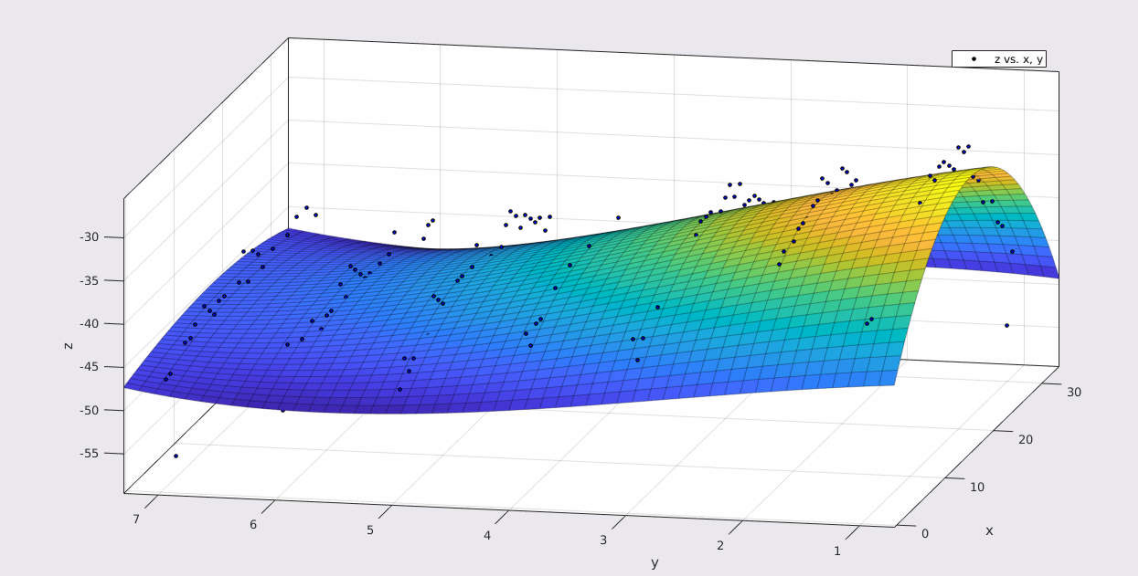

Figure 2.26 Side View of 2-D Surface of the Outdoor Playground for Case 2D-I.

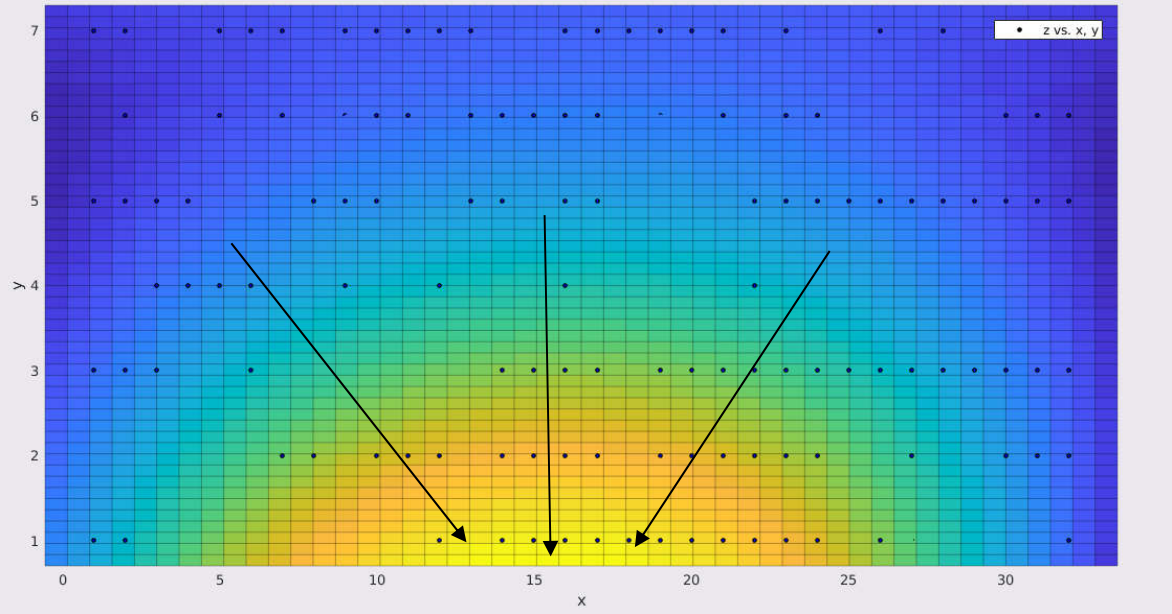

Figure 2.27 Top View of 2-D Surface of the Playground for Case 2D-I.

In the top view of the surface fitting result in Figure 2.27, the yellow color (or light color in black and white) corresponds to positions with higher signal strength in this area. If the yellow color (or light color) is in the middle of the surface, it is concluded that the emitter is in the middle part of this area. However, if the yellow color is at the edge of the surface, there are two

possible situations. One is that the emitter is at some position of the yellow (or light color) area. The other is that the emitter is out of this area and the arrow retrieved from this surface by gradient descent points to the direction of the emitter. We can differentiate these two situations by the actual signal strength data in the yellow color area. It is known that the strongest signal strength is about -20dB. If the signal strength data is close to -20dB in the yellow color area, it can be concluded that the emitter is within the yellow color area. Otherwise, it can be concluded that the emitter is out of the area and we need to conduct another round of searching in the area the arrow points toward.

Two more experiments, Case 2D-II and Case 2D-III are designed in an indoor environment to further explore this gradient area idea. The experiment designs of two cases are showed in Figure 2.28 and Figure 2.30. In Case 2D-II, there are two areas where signal strength data are collected. The emitter is in the middle of grid2. Equation (Eq. 2.3) is used to fit a surface, and Figure 2.29 is the top view of this surface. Although there is an arrow in Figure 2.28, no more searching is suggested in the arrow's direction because the signal strength data in the yellow color area is very close to -20dB, which is the strongest strength. Therefore, it is concluded that the emitter is in the right part of this area. In Case 2D-III, the emitter is in the middle of the whole testing space. Equation (Eq. 2.4) is used to fit a surface, and Figure 2.31 is the top view of this surface. From Figure 2.31, it is concluded that the emitter is in the middle part of this area.

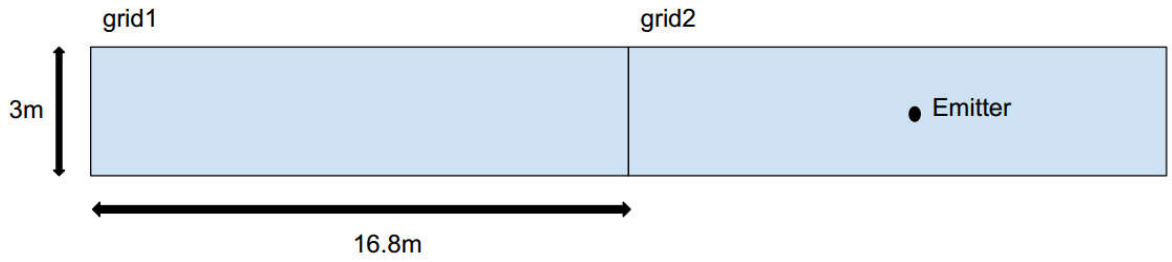

Figure 2.28 Case 2D-II: The Emitter is Placed In the Middle of Grid2 of an Indoor Area.

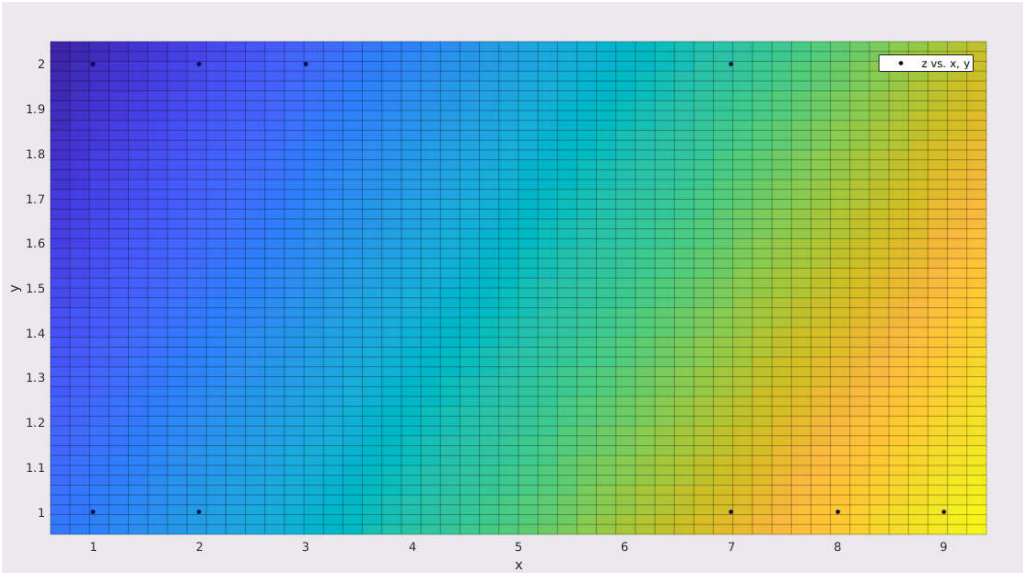

Figure 2.29 Top View of the Fitting Surface for Case 2D-II of an Indoor Area.

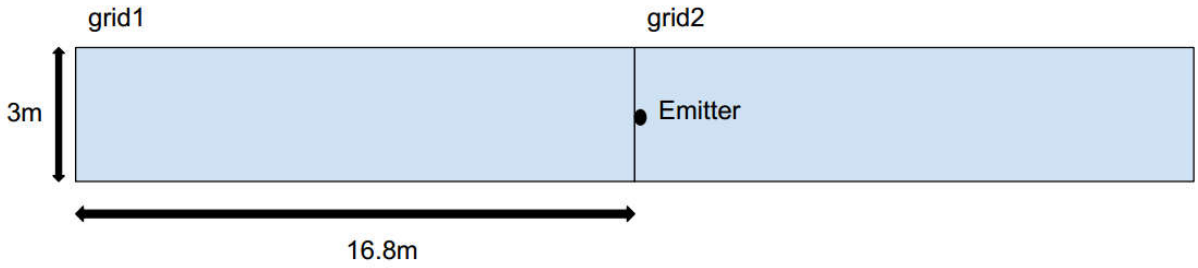

Figure 2.30 2D Case III:The Emitter is Placed in the Middle of the Space of an Indoor Area.

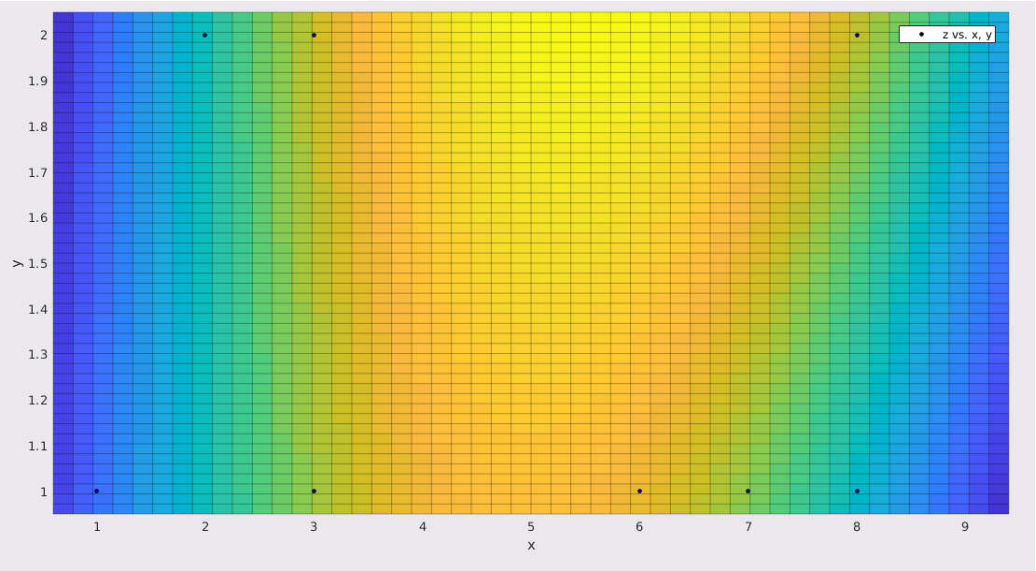

Figure 2.31 Top View of the Fitting Surface for Case 2D-III of an Indoor Area.

2.6. Performance Evaluation for 1-D and 1-D Localization

An experiment is designed to combine the 2-D case and the 1-D case when trying to locate the emitter. The testing situation is shown in Figure 2.32 which includes part of Moench Hall and part of Crapo Hall at Rose-Hulman Institute of Technology. The emitter is placed in Room G220 on the second floor of Crapo Hall in Figure 2.32. The initial signal strength data is collected in the red area Area 1, in Moench Hall in Figure 2.32. The 2-D signal strength data is fit to a surface, as shown in Figure 2.33 from the regression model in Equation (Eq. 2.3). From the signal strength data in the yellow color area, it is concluded that the emitter is in the direction

of the derived arrow. Next, signal strength data is collected along the red line corridor in Figure 2.32. The 1-D signal strength data is fit to a curve, and the fitting result is showed in Figure 2.34. The corresponding position of peak signal strength in Figure 2.34 is where the emitter is.

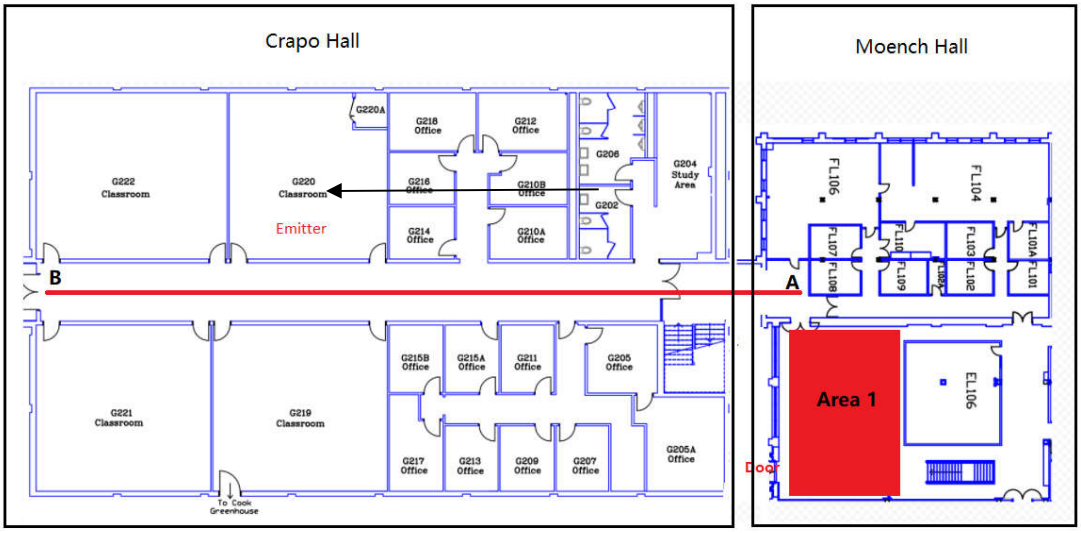

Figure 2.32 Map of Lower Level of Moench Hall and Second Floor of Crapo Hall.

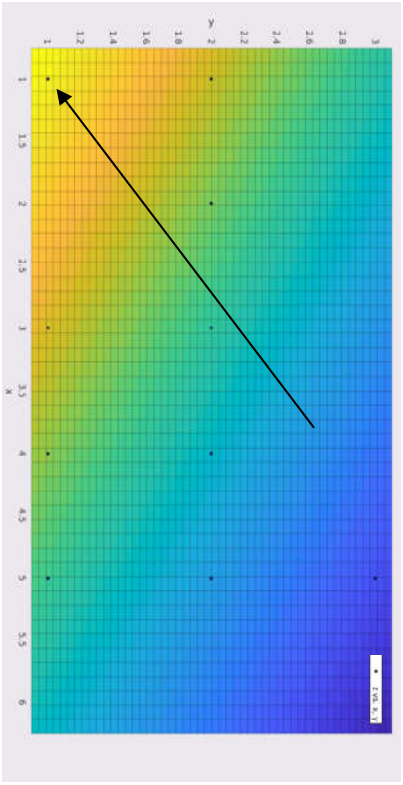

Figure 2.33 Top View of the Fitting Surface of Area 1 in Moench Hall.

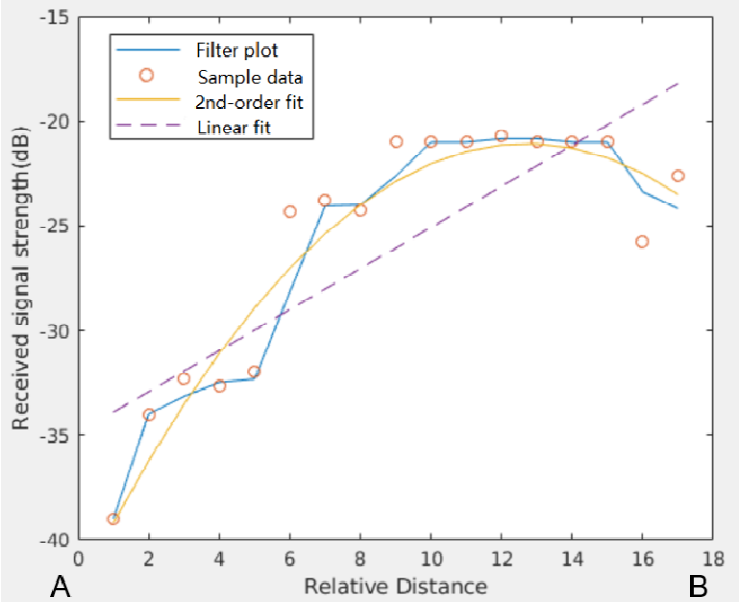

Figure 2.34 1-D Plot of Signal Strength vs. Distance along a Line from Points A to B in Figure 2.32.

3. DESIGN AND IMPLEMENTATION OF THE 2.4GHZ WLAN METHOD

If the emitter is close to the radio tester, say within ten meters of each other, and they are in indoor open space without obstruction, 433MHz wave energy would not show clear attenuation to identify the location of the emitter further. This chapter discusses how wireless local area network (WLAN) can be used to fine-tune emitter localization to identify the location of the emitter with more accuracy. The range of WLAN carrier frequencies is 2.4GHz to 5.9GHz for IEEE 802.11 standard. Electromagnetic waves at these frequencies tend to attenuate much faster than that at 433MHz. this faster attenuation would help to locate the emitter within a close neighborhood. Section 3.1 discusses design details of WLAN method. Section 3.2 presents performance test results.

3.1. Design of the 2.4GHz WLAN Method

In the design of the WLAN method, the emitter is simulated with a smartphone which works well with WLAN signals. The user holds a router and the cellphone during the WLAN method in Figure 3.1. A router is needed because the emitter and the user's cellphone are going to communicate with each other based on the local area network provided by the router. So, we don't need to connect to the WiFi in the building for communication. It will not be influenced even when the building just doesn't have any APs. The emitter is expected to connect to the router once it is in the range of the router. Further, the emitter and the cellphone of the user will create a socket connection and communicate with each other. The APP on the cellphone is working as the server of this socket connection, and the emitter is working as a client. The pictures for the emitter and the radio tester are in Figure 3.2 and Figure 3.3. The workflow in the WLAN method is as follows:

(1) The radio tester is connected to the specified router.
(2) The radio tester remains listening to the socket for the emitter to make a connection request.
(3) Once the emitter is in the range of the specified router, the emitter is also connected to the specified router.
(4) The emitter makes a request for connection by specifying the hostname and the port number.
(5) The radio tester accepts the request, and socket connection is initiated between the radio tester and the emitter.
(6) From now on, the radio tester can ask for signal strength data from the emitter.

Only when the searching range is narrowed to quite small, the 2.4GHz WLAN method is used. At this time, several interesting points are chosen. For each point, signal strength data is collected and averaged. The position which has the strongest signal strength is chosen as the predicted position of the emitter.

Figure 3.1 The 2.4GHz WLAN Method Design.

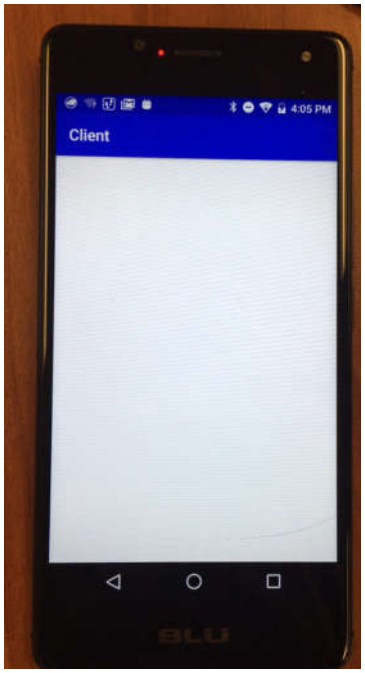

Figure 3.2 The Emitter of the 2.4GHz Wireless System.

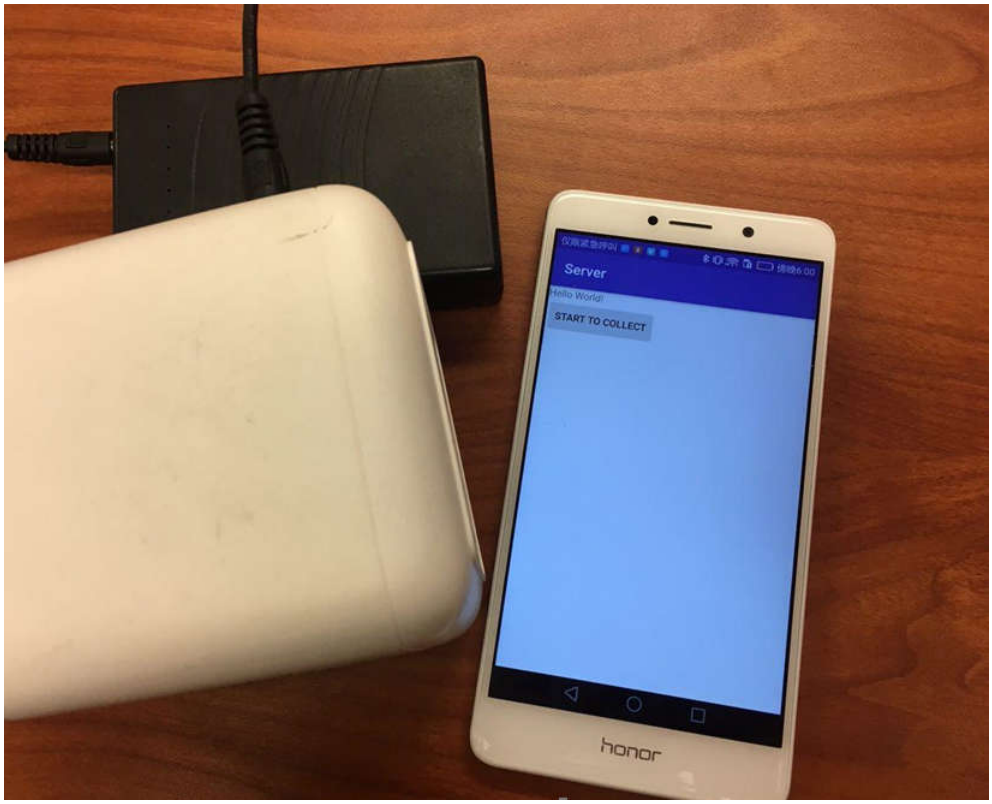

Figure 3.3 The Radio Tester of the 2.4GHz Wireless System.

3.2. Evaluation of the WLAN Method

To show that the 2.4GHz WLAN method is needed in conjunction with the 433MHz method, a comparison experiment is designed. The emitter is placed in the hallway near the door of Room D201 in Figure 3.4. Signal strength data is collected at the doors of D209, D206, D205, D204, D203, D212, D201, C211, C210, C209 for the 433MHz signal and WLAN signal separately. 433MHz signal strength data is shown in Table 3.1. It is obvious that signal strengths from D204 to C210 are very similar to each other which results in localization error of about 13m. The WLAN signal strength data is shown in Table 3.2. Since the energy for the WLAN signal attenuates more quickly than the 433MHz signal, it does a better job to differentiate each other from D204 to C210, and it is quite clear that the signal strength at the door of D201 is the strongest.

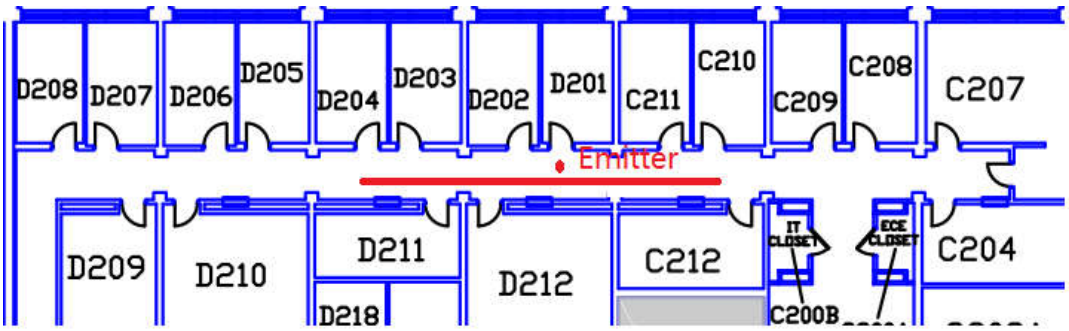

Figure 3.4 Position of the Emitter in the Hallway.

Table 3.1 433MHz Signal Strength Data.

Position	D209	D206	D205	D204	D203	D212	D201	C211	C210	C209
433MHz RSSI	-30.3	-28.8	-26.5	-22	-22	-23	-22	-22	-22.2	-29

Table 3.2 2.4GHz WLAN Signal Strength Data.

Position	D209	D206	D205	D204	D203	D212	D201	C211	C210	C209
WLAN RSSI	-43.9	-41.6	-43.2	-41.2	-34.4	-34.5	-23.6	-34	-31.4	-41.3

4. OVERALL SYSTEM PERFORMANCE EVALUATION

To show the effectiveness of the scheme proposed in this thesis, some system experiments are designed. The whole searching starts from floor level determination and goes through all the steps in Figure 4.1.

An experiment called System Case I is done in Moench Hall in Figure 4.2, and the emitter is placed in the room of D210. First, the signal strength data on three different floors from A0 to B0 are measured in Table 4.1. From the data, it is concluded that the emitter is on the second floor. Then, 1-D data from A1 to B1 are collected, and the processing result is in Figure 4.3. From the 2nd-order fitting curve, it is concluded that the emitter is in the middle range. Next, 1-D data from A2 to B2 are collected. From Figure 4.4, it is concluded that the emitter is close to B2. Then, 1-D data from A3 to B3 is collected. From Figure 4.5, it is concluded that the emitter is close to A3. Finally, small range data sampling is done from D208 to D211 as in Table 4.2. The predicted position of the emitter is shown in Figure 4.2. In this experiment, the localization error is less than 10 meters.

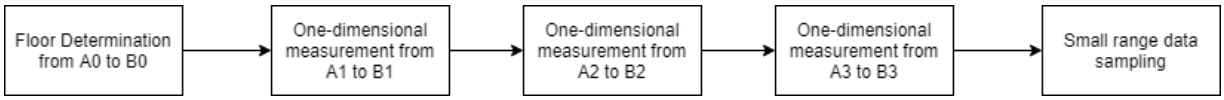

Figure 4.1 Searching Steps.

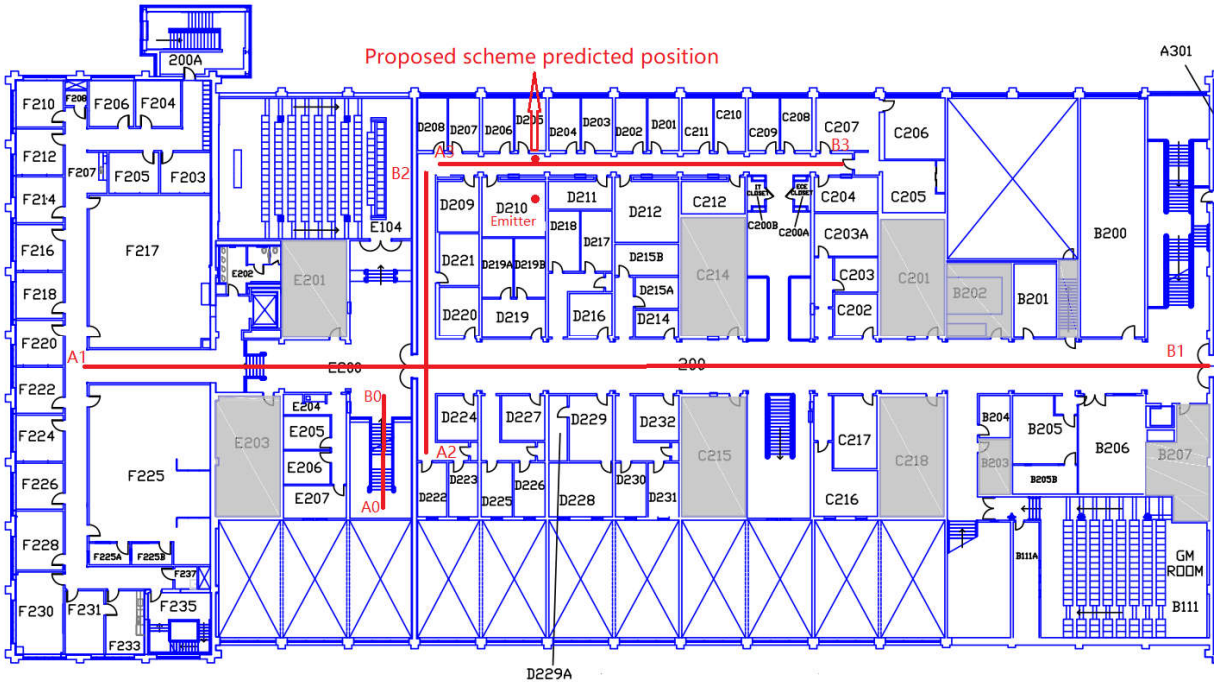

Figure 4.2 System Case I: System Evaluation Environment.

Floor determination:

Table 4.1 Floor Determination of System Evaluation.

Floor	Signal strength
Lower level 1	-60.5dB
Level 1	-64.5dB
Level 2	-48dB

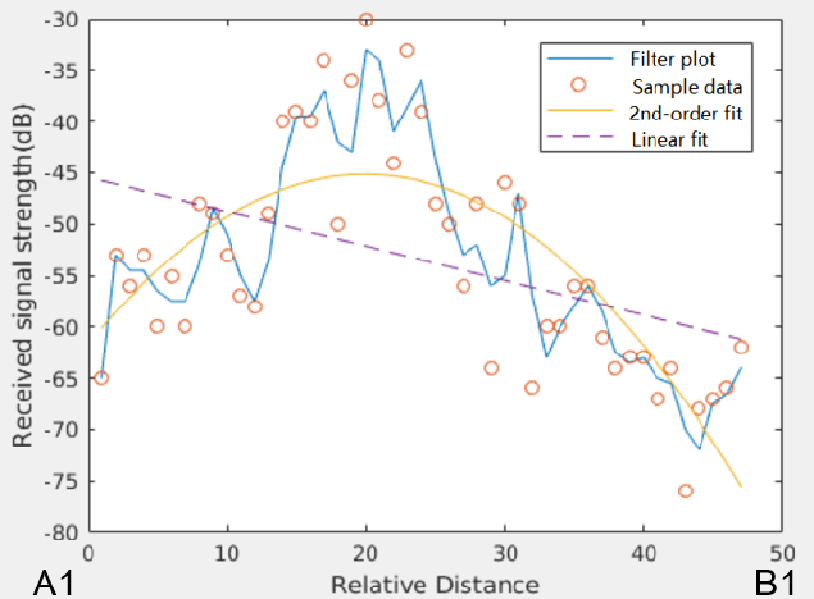

Figure 4.3 1-D Plot of Signal Strength vs. Distance for from A1 to B1.

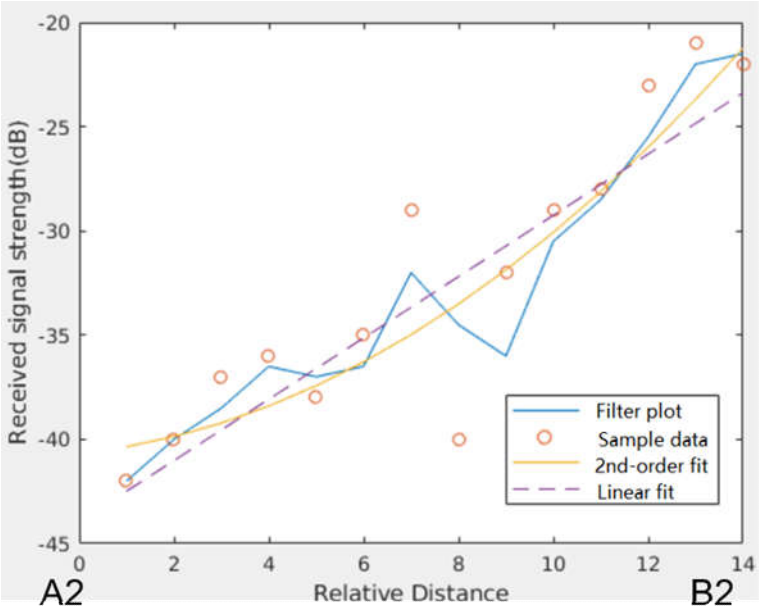

Figure 4.4 1-D Plot of Signal Strength vs. Distance for from A2 to B2.

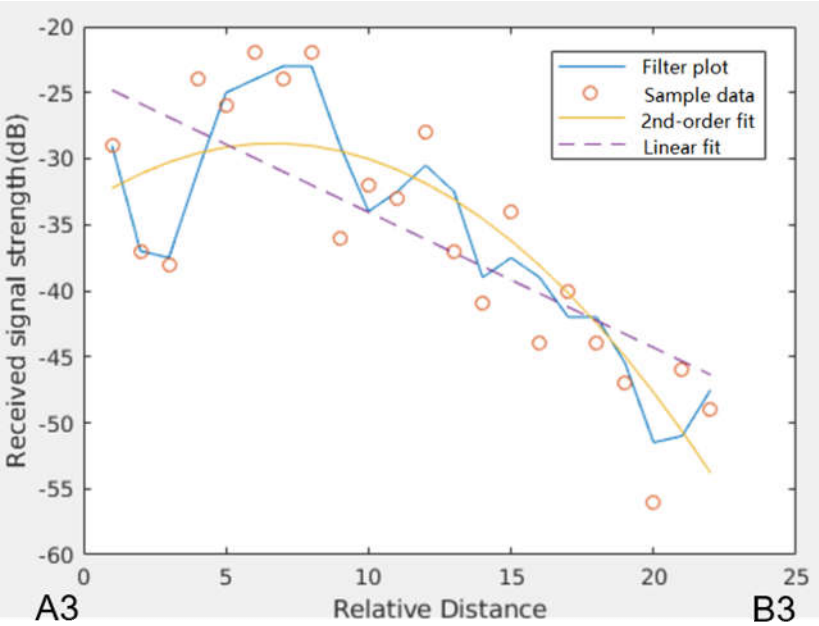

Figure 4.5 1-D Plot of Signal Strength vs. Distance for from A3 to B3.

Table 4.2 Small Range Data Sampling.

Label	Signal strength
D208	-31.8dB
D209	-28.0dB
D210	-33.6dB
D205	-22.8dB
D204	-31dB
D211	-31.4dB

Another comparison experiment is done as System Case II in Figure 4.6. The difference between the System Case I and System Case II is that after finishing floor determination, 2-D data is measured in the red area E200 in Figure 4.6 and the data processing result is in Figure 4.7 for System Case II. The signal strength in the yellow area is about -55dB. So, it is concluded that another searching is needed and the arrow in Figure 4.7 gives us an idea about the direction of the emitter.

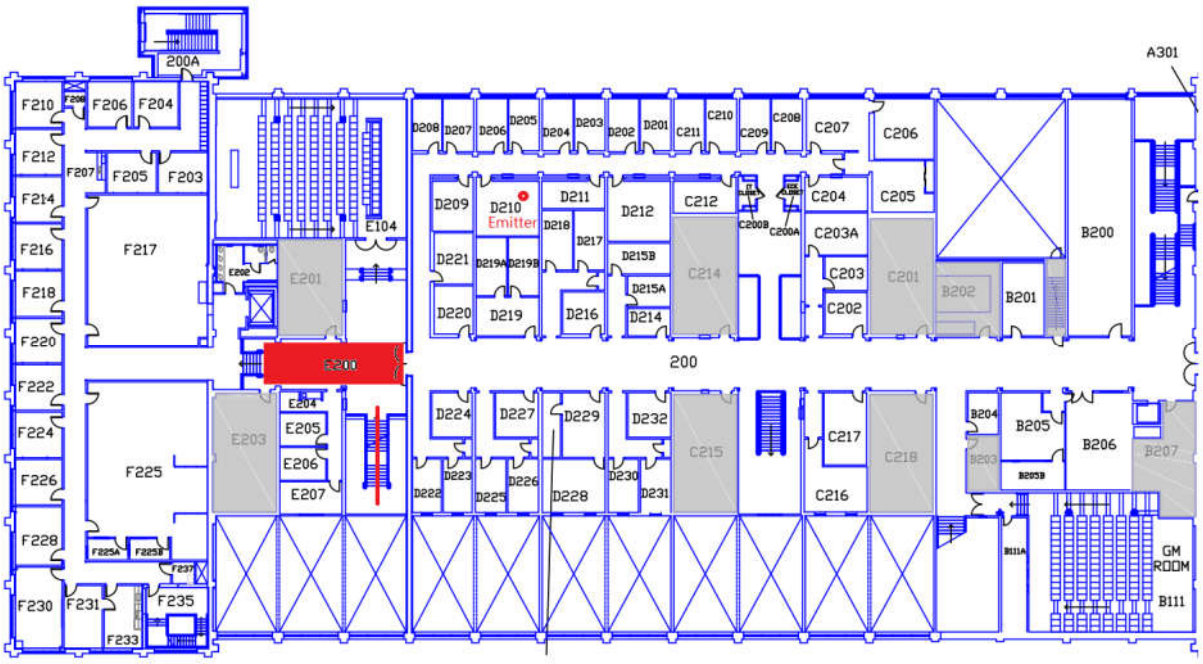

Figure 4.6 System Case II: System Evaluation Environment and Area E200.

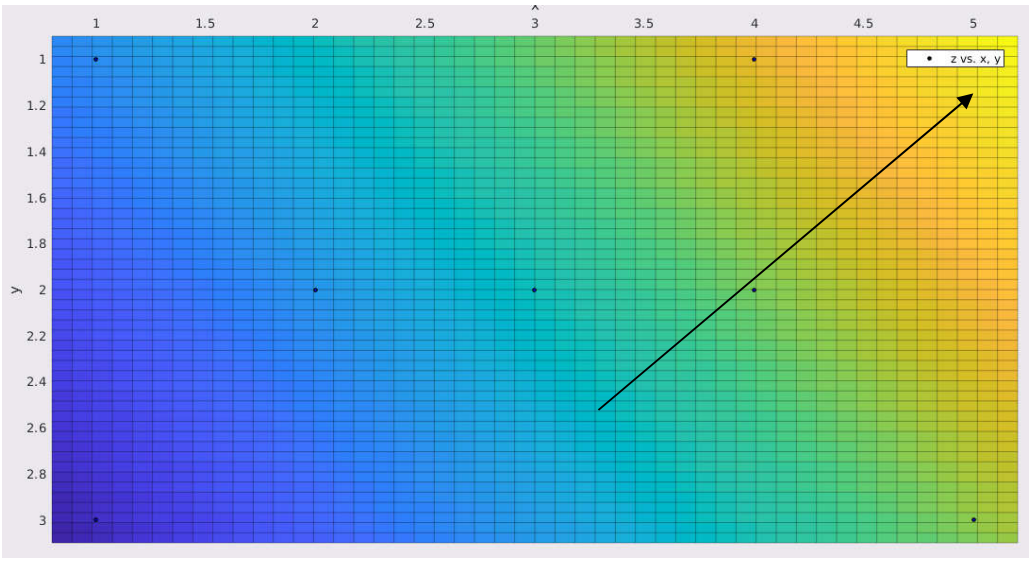

Figure 4.7 Top View of Fitting Surface for System Case II.

5. COMPARISON WITH THE CENTROID ALGORITHM

The goal of the thesis is to locate the emitter from some received signal strength measurements from different locations. Two kinds of signals are used. One is the 433MHz signal. The other is the 2.4GHz WLAN signal. The data include location information and received signal strength information. One of the most common localization algorithms is the centroid algorithm [31]. In this chapter, we will compare the localization accuracy of the proposed scheme with the centroid algorithm.

5.1. The Centroid Algorithm for Emitter Localization

Given the set of measurement points $(x_i, y_i, RSSI_i)$, centroid algorithms will locate the emitter at the averaged position $(\sum_{i=1}^{N} w_i x_i, \sum_{i=1}^{N} w_i y_i)$. $w_i = \frac{1}{N}$.

One drawback of the centroid algorithm is that it is assuming the emitter is in the data sampling area. This assumption is not always true which results in inaccuracy of the centroid algorithm. However, our proposed algorithm in Section 2.4 and 2.5 doesn't make this assumption. After analyzing the signal strength data in the sampling area, our algorithm can decide whether the emitter is in this range or another round of searching is needed.

Even though the emitter is in the data sampling area indeed, warwalking measurements introduce strong sampling biases which result in inaccuracy of the centroid algorithm. Warwalking means the way to collect data by people walking.

To understand why our algorithm outperforms the central algorithm, the central algorithm is applied to the experiment in Chapter 4 as shown in Figure 5.1.

5.2. Accuracy Comparison

The experiment is designed in Figure 5.1. The emitter is in D210, and the signal strength data is collected on the red lines. The predicted position of the centroid algorithm and the predicted position of our proposed scheme are showed in Figure 5.1. It is obvious that our proposed scheme performs much better than the centroid algorithm in this case.

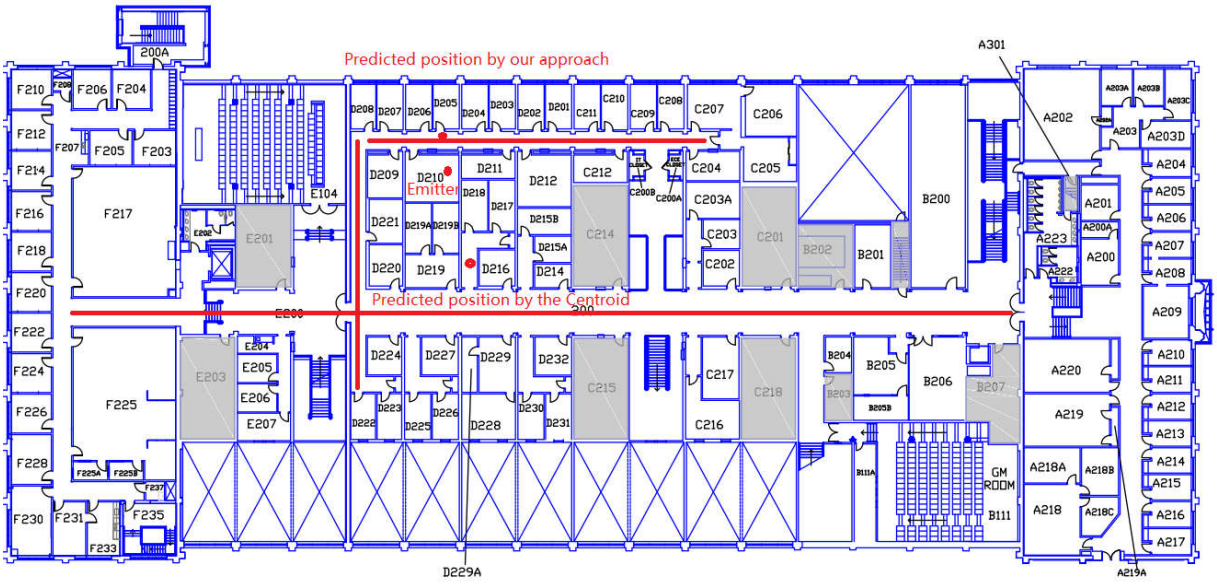

Figure 5.1 Localization Accuracy Comparison of the Centroid Algorithm and Our Proposed Scheme.

6. CONCLUSION AND FUTURE WORK

Our approach uses emitter signal strength to localize its position. Our search for the emitter starts with floor level determination and then the location of the emitter on one floor of a building.

The signal strength can be sampled in 1-D space or 2-D space. Those samples are processed with regression models to extract information on the direction or location of the emitter. The main contribution of this thesis is the idea of signal strength data collection and processing in 2-D space. The result of this data processing is a 2-D surface of signal strength changes over the area. This surface provides an excellent direction or guidance for further search through the gradient. Our approach has two advantages over the centroid algorithm, which is the most common localization algorithm. One is that our approach does not assume the emitter is inside the search area and our 2D surfaces will iteratively guide the search process. Our approach does not use sampling point average to decide the location of the emitter as the centroid algorithm does. Therefore, our approach will not be influenced by sampling location biases.

An entire emitter indoor localization system composed of the 433MHz wireless system and 2.4GHz WLAN wireless system has been designed and implemented. The hardware components for the 433MHz implementation consists of an emitter and a radio tester. Its software components consist of an Android APP and MATLAB code. The 2.4GHz WLAN implementation uses a smartphone as the emitter and another smartphone and its router as the radio tester. One advantage of 433MHz radiation is its long-distance propagation, especially with LoRa modulation technology. Its drawback is also its slow attenuation so that its energy

reduction cannot be used to distinguish the emitter in a short range. This is when the 2.4GHz WLAN can play an important role to cover short-range localization.

Future expansion of our work could be in the following three areas.

(1) To move 2-D data processing in the 433MHz wireless system from MATLAB to the Android APP. Currently, the Android APP which interfaces with the radio tester in the 433MHz wireless system can conduct 1-D data collection, 2-D data collection, and 1-D data regression analysis. However, it cannot do 2-D data regression analysis and gradient yet. So it would be nice to implement this function in the Android APP for convenience.

(2) To add more examples of our iterative and integrated approach. Iteration means to conduct several searches until the emitter is located. Each search can provide a direction for the next search. The search can be 1-D search, 2-D search or small range point search.

(3) To develop guidelines for switching between the 433MHz approach and 2.4GHz WLAN approach. The 433MHz approach is applied when the radio tester is quite far from the emitter while the 2.4GHz approach is applied when the radio tester is very close to the emitter. So, guidelines for switching between the 433MHz approach and 2.4GHz WLAN approach need to be developed so that the user can be quite clear about when to use the 433MHz approach and when to use the 2.4GHz WLAN approach.

LIST OF REFERENCES

[1] Huai-Jing Du and Jim Lee, “Radar Emitter Localization Using TDOA Measurements from UAVs and Shipborne/Land-Based Platforms,” Defence R&D Canada, 2002.

[2] Zafari, Faheem, Athanasios Gkelias, and Kin Leung, "A Survey of Indoor Localization Systems and Technologies," arXiv preprint arXiv:1709.01015 (2017).

[3] Gorji, Aliakbar A., and Brian DO Anderson, "Emitter Localization Using Received-strength-signal Data," Signal Processing 93.5 (2013): 996-1012.

[4] Liu, Hui, et al, "Survey of Wireless Indoor Positioning Techniques and Systems," IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 37.6 (2007): 1067-1080.

[5] Niculescu, Dragos, and Badri Nath. "Ad Hoc Positioning System (APS) using AOA." INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies. Vol. 3. Ieee, 2003.

[6] Macagnano, Davide, Giuseppe Destino, and Giuseppe Abreu, "Indoor Positioning: A Key Enabling Technology for IoT Applications." Internet of Things (WF-IoT), 2014 IEEE World Forum on. IEEE, 2014.

[7] Xia, Feng, et al, "Internet of Things," International Journal of Communication Systems 25.9 (2012): 1101.

[8] Cho, Yong K., Jong Hoon Youn, and Diego Martinez, "Error Modeling for an Untethered Ultra-wideband System for Construction Indoor Asset Tracking," Automation in Construction 19.1 (2010): 43-54.

[9] Shen, Shaojie, Nathan Michael, and Vijay Kumar, "Autonomous Multi-floor Indoor Navigation with a Computationally Constrained MAV," Robotics and automation (ICRA), 2011 IEEE international conference on. IEEE, 2011.

[10] Youssef, Moustafa, and Ashok Agrawala, "The Horus WLAN Location Determination System," Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services. ACM, 2005.

[11] Chen, Lina, et. al, "An Improved Algorithm to Generate a Wi-Fi Fingerprint Database for Indoor Positioning," Sensors 13.8 (2013): 11085-11096.

[12] B. B. Peterson, C. Kmiecik, R. Hartnett, P. M. Thompson, J. Mendoza, H. Nguyen, "Spread Spectrum Indoor Geolocation,", J. Inst. Navigat., vol. 45, no. 2, pp. 97-102, 1998.

[13] N. S. Correal, S. Kyperountas, Q. Shi, M. Welborn, "An Ultra Wideband Relative Location System," in Proc. IEEE Conf. Ultra Wideband Syst. Technol., pp. 394-397, Nov. 2003.

[14] Zhang, Cemin, et. al, "Accurate UWB Indoor Localization System Utilizing Time Difference of Arrival Approach," Radio and Wireless Symposium, 2006 IEEE.

[15] Wang, Xiaopu, Yan Xiong, and Wenchao Huang, "An Accurate Direction Finding Scheme Using Virtual Antenna Array via Smartphones," Sensors 16.11 (2016): 1811.

[16] https://en.wikipedia.org/wiki/Monopole_antenna.

[17] Khan, Niazul Islam, Anwarul Azim, and Shadli Islam, "Radiation Characteristics of a Quarter-wave Monopole Antenna above Virtual Ground," Journal of Clean Energy Technologies 2.4 (2014).

[18] Clayton A. Paul, Introduction to Electromagnetic Compatibility, 2nd Edition, John Wiley & Sons, Inc., 2006.

[19] Constantine A. Balanis, Antenna Theory: Analysis and Design, Third Edition, John Wiley & Sons, Inc., 2005.

[20] https://en.wikipedia.org/wiki/Poynting_vector.

[21] https://en.wikipedia.org/wiki/Linear_regression.

[22] https://en.wikipedia.org/wiki/Polynomial_regression.

[23] https://en.wikipedia.org/wiki/Taylor_series.

[24] https://www.link-labs.com/lora.

[25] SX1276 datasheet, Page87.

[26] https://en.wikipedia.org/wiki/Arduino.

[27] http://www.airspayce.com/mikem/arduino/RadioHead/.

[28] Adafruit RFM69HCW and RFM9X LoRa Packet Radio Breakouts Datasheet.

[29] https://www.amazon.in/CENTIoT-Bluetooth-wireless-Serial-Transceiver/dp/B01M14FKHV.

[30] Coefficient of Determination Definition, Statrek website.

[31] Han, Dongsu, et. al, "Access Point Localization Using Local Signal Strength Gradient," International Conference on Passive and Active Network Measurement. Springer, Berlin, Heidelberg, 2009.

APPENDICES

APPENDIX A. 433MHz Radio Tester and Emitter Programs on Arduino UNO Board

Radio Tester Code

```
#include <SPI.h>
#include <RH_RF95.h>
#define RFM95_CS 10
#define RFM95_RST 9
#define RFM95_INT 2
// Singleton instance of the radio driver
RH_RF95 rf95(RFM95_CS, RFM95_INT);
//string that stores the incoming message
String message;

// the setup function runs once when you press reset or power the board
void setup() {
  pinMode(RFM95_RST, OUTPUT);
  digitalWrite(RFM95_RST, HIGH);
  while (!Serial);
  Serial.begin(9600);
  delay(100);
  Serial.println("Arduino LoRa TX Test!");
  // manual reset
  digitalWrite(RFM95_RST, LOW);
  delay(10);
  digitalWrite(RFM95_RST, HIGH);
  delay(10);
  while (!rf95.init()) {
    Serial.println("LoRa radio init failed");
    while (1);
  }
  Serial.println("LoRa radio init OK!");
  // The initialization work for the radio module is done in rf95.init(). Defaults
after init are 434.0MHz, 13dBm, Bw = 125 kHz, Cr = 4/5, Sf = 128chips/symbol, CRC on.
The default transmitter power is 13dBm, using PA_BOOST.
```

```
//To increase the Tx power, set the Tx power to be 23dB.
  rf95.setTxPower(23, false);
}

int16_t packetnum = 0; // packet counter, we increment per mission
void loop() {
  while(Serial.available())
  {
   //while there is data available on the serial monitor
    message+=char(Serial.read());//store string from serial command
  }
  if(!Serial.available())
  {
    if(message!="")
    {
    //If data is available from the Android APP, send a message to the emitter
      char radiopacket[20] = "Hello World # ";
      itoa(packetnum++, radiopacket+13, 10);
      radiopacket[19] = 0;
      delay(10);
      rf95.send((uint8_t *)radiopacket, 20);
      delay(10);
      rf95.waitPacketSent();
      // Now wait for a reply from the emitter
      uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
      uint8_t len = sizeof(buf);
      delay(10);
      if (rf95.waitAvailableTimeout(1000))
      {
        // Should be a reply message for us now
        if (rf95.recv(buf, &len))
        {
          //Receive reply from the emitter successfully. Then send RSSI and SNR data
back to the Android APP
          uint8_t test[4];
          test[0] = 1;
          test[1] = rf95.lastRssi();
          test[2] = rf95.getSNR();
          test[3] = 255;
```

```
          Serial.write(test, sizeof(test));
        }
        else
        {
            uint8_t test[4];
            test[0] = 0;
            test[1] = 1;
            test[2] = 1;
            test[3] = 255;
            Serial.write(test, sizeof(test));
        }
      }
      else
      {
        //No reply from the emitter within the expected time. (Timeout)
        uint8_t test[4];
        test[0] = 0;
        test[1] = 2;
        test[2] = 2;
        test[3] = 255;
        Serial.write(test, sizeof(test));
      }
      message=""; //clear the data
    }
  }
}
```

Emitter Code

```
#include <SPI.h>
#include <RH_RF95.h>

#define RFM95_CS 10
#define RFM95_RST 9
#define RFM95_INT 2
#define RFM95_INT3 3

// Set carrier frequency to be 434MHz
#define RF95_FREQ 434
```

```
// Singleton instance of the radio driver
RH_RF95 rf95(RFM95_CS, RFM95_INT);
// Blinky on receipt for debugging
#define LED 13

void setup() {
  pinMode(LED, OUTPUT);
  pinMode(RFM95_RST, OUTPUT);
  digitalWrite(RFM95_RST, HIGH);
  while (!Serial);
  Serial.begin(9600);
  delay(100);
  Serial.println("Arduino LoRa RX Test!");
  // manual reset
  digitalWrite(RFM95_RST, LOW);
  delay(10);
  digitalWrite(RFM95_RST, HIGH);
  delay(10);
  while (!rf95.init()) {
    Serial.println("LoRa radio init failed");
    while (1);
  }
  Serial.println("LoRa radio init OK!");
  // Defaults after init are 434.0MHz, modulation GFSK_Rb250Fd250, +13dbM
  if (!rf95.setFrequency(RF95_FREQ)) {
    Serial.println("setFrequency failed");
    while (1);
  }
  Serial.print("Set Freq to: ");
  Serial.println(RF95_FREQ);
  // Defaults after init are 434.0MHz, 13dBm, Bw = 125 kHz, Cr = 4/5, Sf =
128chips/symbol, CRC on. The default transmitter power is 13dBm, using PA_BOOST.
  //To increase Tx power, set it to be 23dB
  rf95.setTxPower(23, false);
}

void loop() {
  if (rf95.available())
  {
    // Should be a message for us now
```

```
    uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
    uint8_t len = sizeof(buf);
    if (rf95.recv(buf, &len))
    {
      //A packet is received from the radio tester.
      digitalWrite(LED, HIGH);
      RH_RF95::printBuffer("Received: ", buf, len);
      Serial.print("Got: ");
      Serial.println((char*)buf);
      Serial.print("RSSI: ");
      int8_t Rssi = rf95.lastRssi();
      Serial.println(rf95.lastRssi(), DEC);
      rf95.printSNR();
      //Send a reply to the radio tester
      uint8_t data[] = "And hello back to you";
      rf95.send(data, sizeof(data));
      rf95.waitPacketSent();
      Serial.println("Sent a reply");
      digitalWrite(LED, LOW);
    }else
    {
      Serial.println("Receive failed");
    }
  }
}
```

APPENDIX B. Android APP for Interfacing with the 433MHz Tester

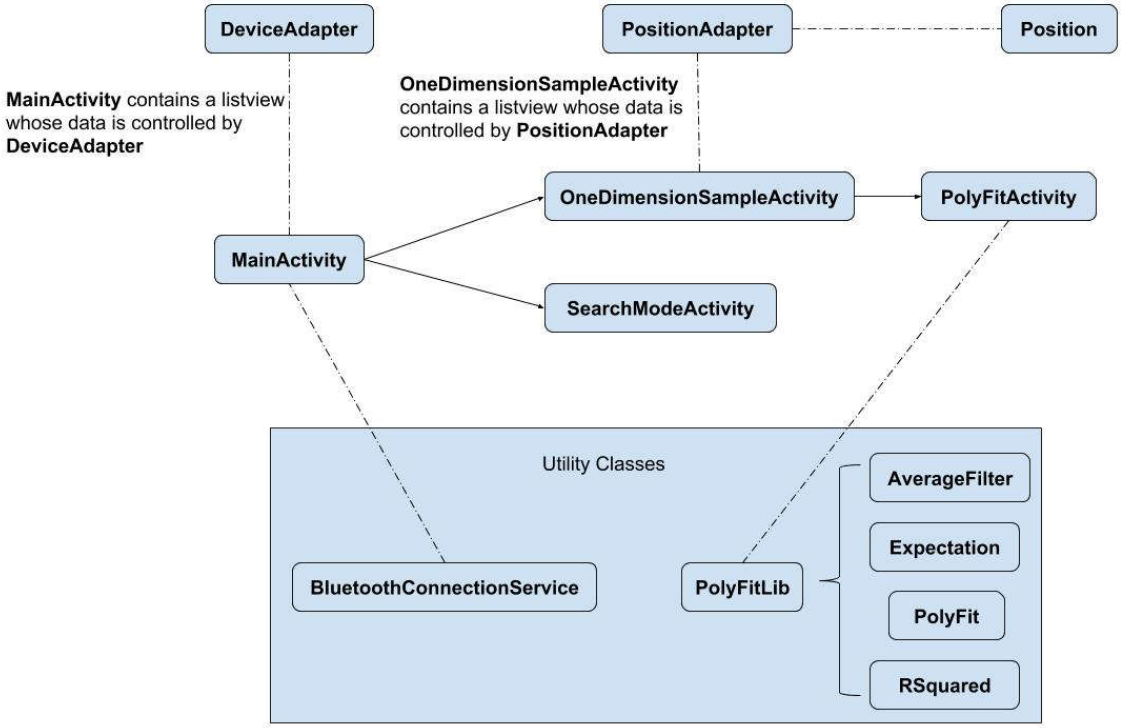

Figure 7.1 Diagram for Android APP Interfacing with Radio Tester in 433MHz Wireless System.

Table 7.1 Class Description for Figure 7.1.

Class	**Description**
MainActivity	In charge of making Bluetooth pair and connection with the radio tester
OneDimensionSampleActivity	In charge of 1-D data sampling
SearchModeActivity	In charge of 2-D data sampling and small range point data sampling
PolyFitActivity	In charge of 1-D data processing which is regression analysis

BluetoothConnectionService.java

```
package com.example.hang.bluetoothdatatest;

public class BluetoothConnectionService {

    private static final String TAG = "BluetoothConnectionServ";

    private static final String appName = "MYAPP";

    private static final UUID MY_UUID_INSECURE = UUID.fromString("8ce255c0-200a-11e0-
ac64-0800200c9a66");


    private AcceptThread mInsecureAcceptThread;

    private ConnectThread mConnectThread;

    private ConnectedThread mConnectedThread;


    private BluetoothDevice mmDevice;

    private UUID deviceUUID;

    ProgressDialog mProgressDialog;

    private final BluetoothAdapter mBluetoothAdapter;

    Context mContext;


    private BluetoothConnectionService(Context context) {

        mContext = context;

        mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

    }


    private static BluetoothConnectionService instance;

    public static BluetoothConnectionService getInstance(Context context) {

        if (instance == null) {

            instance = new BluetoothConnectionService(context);

        }

        return instance;

    }


    public static BluetoothConnectionService getInstance() {

        return instance;

    }


    /**

     * This thread runs while listening for incoming connections. It behaves

     * like a server-side client. It runs until a connection is accepted

     * (or until cancelled).
```

```
     */
    private class AcceptThread extends Thread {
        private final BluetoothServerSocket mmServerSocket;
        public AcceptThread() {
            BluetoothServerSocket tmp = null;
            try {
                tmp =
mBluetoothAdapter.listenUsingInsecureRfcommWithServiceRecord(appName,
MY_UUID_INSECURE);
                Log.d(TAG, "AcceptThread:Setting up Server using" + MY_UUID_INSECURE);
            } catch (IOException e) {
                e.printStackTrace();
            }
            mmServerSocket = tmp;
        }

        public void run() {
            Log.d(TAG, "run: AcceptThread Running.");
            BluetoothSocket socket = null;

            try {
                Log.d(TAG, "run: RFCOM server socket start.....");
                socket = mmServerSocket.accept();
                Log.d(TAG, "run: RFCOM server socket accept connection..");
            } catch (IOException e) {
                e.printStackTrace();
            }

            if (socket != null) {
                connected(socket, mmDevice);
            }
            Log.i(TAG, "End mAcceptThread");
        }

        public void cancel() {
            Log.d(TAG, "cancel: Cancelling AcceptThread");

            try {
                mmServerSocket.close();
            } catch (IOException e) {
                e.printStackTrace();
```

```
            }
        }
    }

    /**
     * This thread runs while attempting to make an outgoing connection
     * with a device. It runs straight through; the connection either
     * succeeds or fails.
     */
    private class ConnectThread extends Thread {
        private BluetoothSocket mmSocket;
        public ConnectThread(BluetoothDevice device, UUID uuid) {
            Log.d(TAG, "ConnectThread: started");
            mmDevice = device;
            deviceUUID = uuid;
        }

        public void run() {
            BluetoothSocket tmp = null;
            Log.i(TAG, "RUN mConnectThread");

            //Get a BluetoothSocket for a connection with the given BluetoothDevice
            try{
                tmp = mmDevice.createRfcommSocketToServiceRecord(deviceUUID);
            } catch (IOException e) {
                e.printStackTrace();
            }
            mmSocket = tmp;
            //always cancel discovery because it will slow down a connection
            mBluetoothAdapter.cancelDiscovery();

            try {
                //This is a blocking call and will only return on a successful
connection or an exception
                mmSocket.connect();
                Log.d(TAG, "run: ConnectThread connected");
                connected(mmSocket, mmDevice);
            } catch (IOException e) {
                //close the socket
                try {
```

```
                    mmSocket.close();
                    Log.d(TAG, "Closed socket.");
                } catch (IOException e1) {
                    Log.e(TAG, "mConnectThread: run: could not close connection in
socket " + e1.getMessage());
                }
                Log.d(TAG, "run: mConnectThread: Could not connect to UUID: " +
MY_UUID_INSECURE);
            }
        }
        public void cancel() {
            try {
                Log.d(TAG, "cancel: Closing Client socket.");
                mmSocket.close();
            } catch (IOException e) {
               Log.e(TAG, "cancel: close() of mmSocket in ConnectThread failed. " +
e.getMessage());
            }
        }
    }



    /**
     * Start the chat service. Specifically start AcceptThread to begin a
     * session in listening (server) mode. Called by the Activity onResume()
     */
    public synchronized void start() {
        Log.d(TAG, "start");

        // Cancel any thread attempting to make a connection
        if (mConnectThread != null) {
            mConnectThread.cancel();
            mConnectThread = null;
        }

        if (mInsecureAcceptThread == null) {
            mInsecureAcceptThread = new AcceptThread();
            mInsecureAcceptThread.start();
        }
    }
```

```
    //AcceptThread starts adn sits waiting for a connection
    //Then ConnectThread starts and attempts to make a connection with the other
devices AcceptThread


    //I will call this method.
    public void startClient(BluetoothDevice device, UUID uuid) {
        Log.d(TAG, "startClient: started");


        //Initprogress dialog
        mProgressDialog = ProgressDialog.show(mContext, "Connecting Bluetooth",
"Please Wait...", true);
        mConnectThread = new ConnectThread(device, uuid);
        mConnectThread.start();
    }


    /**
     * This thread runs during a connection with a remote device.
     * It handles all incoming and outgoing transmissions.
     */
    private class ConnectedThread extends Thread {
        private final BluetoothSocket mmSocket;
        private final InputStream mmInStream;
        private final OutputStream mmOutStream;

        public ConnectedThread(BluetoothSocket socket) {
            Log.d(TAG, "ConnectedThread: starting.");

            mmSocket = socket;
            InputStream tmpIn = null;
            OutputStream tmpOut = null;

            //dismiss the progressdialog when connection is established
            mProgressDialog.dismiss();
//            Toast.makeText(mContext, "It is connected", Toast.LENGTH_LONG).show();

            try {
                tmpIn = mmSocket.getInputStream();
                tmpOut = mmSocket.getOutputStream();
            } catch (IOException e) {
                e.printStackTrace();
            }
```

```
            mmInStream = tmpIn;
            mmOutStream = tmpOut;
        }

        public void run() {
            byte[] buffer = new byte[10];
            int bytes;
            List<Byte> list = new ArrayList<>();

            while (true) {
                //Read from inputStream
                try {
                    bytes = mmInStream.read(buffer);
                    for (int i = 0; i < bytes; i++) {
                        list.add(buffer[i]);
                    }
                    if ((buffer[bytes-1] & 0xFF) == 0xFF) {
                        //end of this packet
                        int size = list.size();
                        byte[] res = new byte[size];
                        for (int i = 0; i < list.size(); i++) {
                            res[i] = list.get(i);
                        }
                        Intent incomingMessageIntent = new Intent("incomingMessage");
                        incomingMessageIntent.putExtra("MessageByteArray", res);
LocalBroadcastManager.getInstance(mContext).sendBroadcast(incomingMessageIntent);
                        list.clear();
                    }
                } catch (IOException e) {
                    Log.e(TAG, "write : Error reading inputStream.." + e.getMessage());
                    break;
                }
            }
        }

        //call this from the main activity to send data to the remote device
        public void write(byte[] bytes) {
            String text = new String(bytes, Charset.defaultCharset());
            Log.d(TAG, "write: Writing to outputStream: " + text);
```

```
            try {
                mmOutStream.write(bytes);
            } catch (IOException e) {
                Log.e(TAG, "write : Error wriring to outputstream." + e.getMessage());
            }
        }

        //call this from the main activity to shutdown the connection
        public void cancel() {
            try {
                mmSocket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }


    private void connected(BluetoothSocket mmSocket, BluetoothDevice mmDevice) {
        Log.d(TAG, "connected: starting");
        //Start the thread to manage the connection and perform transmissions
        mConnectedThread = new ConnectedThread(mmSocket);
        mConnectedThread.start();
    }

    public void write(byte[] out) {
        Log.d(TAG, "write: Write called.");
        //Perform the write
        mConnectedThread.write(out);
    }
}
```

MainActivity.java

```
package com.example.hang.bluetoothdatatest;

public class MainActivity extends AppCompatActivity implements
AdapterView.OnItemClickListener {
    private static final String TAG = "MainActivity";


    // If you are connecting to a Bluetooth serial board then try using the well-known
SPP UUID 00001101-0000-1000-8000-00805F9B34FB.
```

```
    private static final UUID MY_UUID_INSECURE = UUID.fromString("00001101-0000-1000-
8000-00805F9B34FB");
    BluetoothAdapter mBluetoothAdapter;
    BluetoothConnectionService mBluetoothConnection;
    Button btnStartConnection;
    Button btnSend2;
    Button btn_OneDimension;
    TextView incomingMessage;



    Button btnSearchMode;


    private ArrayList<BluetoothDevice> mBTDevices;
    BluetoothDevice mBTDevice;
    ArrayAdapter<BluetoothDevice> adapter;
    ListView lvNewDevices;
    StringBuilder messages;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mBTDevices = new ArrayList<>();
        adapter = new DeviceAdapter(this, mBTDevices);


        Button btnONOFF = (Button) findViewById(R.id.btnONOFF);
        btnSend2 =  (Button) findViewById(R.id.btnSend2);
        btnStartConnection =  (Button) findViewById(R.id.btnStartConnection);
        btnSearchMode = (Button) findViewById(R.id.btnSearchMode);
        btn_OneDimension = (Button) findViewById(R.id.btn_OneDimension);
        incomingMessage = (TextView) findViewById(R.id.incomingMessage);
        messages = new StringBuilder();


        LocalBroadcastManager.getInstance(this).registerReceiver(mReceiver, new
IntentFilter("incomingMessage"));


        lvNewDevices = (ListView) findViewById(R.id.lvNewDevices);
        lvNewDevices.setAdapter(adapter);


        mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
```

```
        IntentFilter filter = new
IntentFilter(BluetoothDevice.ACTION_BOND_STATE_CHANGED);
        registerReceiver(mBroadcastReceiver4, filter);
        lvNewDevices.setOnItemClickListener(MainActivity.this);


        btnONOFF.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                enableDisableBluetooth();
            }
        });

        btnStartConnection.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                startConnection();
            }
        });



        btnSend2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                byte[] bytes = "2".getBytes(Charset.defaultCharset());
                mBluetoothConnection.write(bytes);
            }
        });



        btnSearchMode.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(MainActivity.this,
SearchModeActivity.class);
                startActivity(intent);
            }
        });

        btn_OneDimension.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
```

```
                Intent intent = new Intent(MainActivity.this,
OneDimensionSampleActivity.class);
                startActivity(intent);
            }
        });
    }

    public void startConnection() {
        startBTConnection(mBTDevice, MY_UUID_INSECURE);
    }

    public void startBTConnection(BluetoothDevice device, UUID uuid) {
        Log.d(TAG, "startBTConnection: Initializing RFCOM Bluetooth Connection");
        mBluetoothConnection.startClient(device, uuid);
    }

    BroadcastReceiver mReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            messages = new StringBuilder();
            byte[] bytes = intent.getByteArrayExtra("MessageByteArray");
            //messages.append(text + "\n");
            for (byte b : bytes) {
                int v2 = b & 0xFF; // v2 is 200 (0x000000C8)
                messages.append(v2 + " ");
            }
            //messages.append("\n");
            incomingMessage.setText(messages.toString());
        }
    };


    private final BroadcastReceiver mBroadcastReceiver1 = new BroadcastReceiver() {
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();
            if (action.equals(mBluetoothAdapter.ACTION_STATE_CHANGED)) {
                final int state = intent.getIntExtra(BluetoothAdapter.EXTRA_STATE,
mBluetoothAdapter.ERROR);
                switch (state) {
                    case BluetoothAdapter.STATE_OFF:
                        Log.d(TAG, "OnReceive: STATE_OFF");
```

```
                        break;
                    case BluetoothAdapter.STATE_ON:
                        Log.d(TAG, "OnReceive: STATE_ON");
                        break;
                    case BluetoothAdapter.STATE_TURNING_OFF:
                        Log.d(TAG, "OnReceive: STATE_TURNING_OFF");
                        break;
                    case BluetoothAdapter.STATE_TURNING_ON:
                        Log.d(TAG, "OnReceive: STATE_TURNING_ON");
                        break;
                }

            }
        }
    };

    private final BroadcastReceiver mBroadcastReceiver2 = new BroadcastReceiver() {
        public void onReceive(Context context, Intent intent) {
            final String action = intent.getAction();
            if (action.equals(BluetoothAdapter.ACTION_SCAN_MODE_CHANGED)) {
                int mode = intent.getIntExtra(BluetoothAdapter.EXTRA_SCAN_MODE,
BluetoothAdapter.ERROR);
                switch (mode) {
                    case BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE:
                        Log.d(TAG, "mBroadcastReceiver2:
SCAN_MODE_CONNECTABLE_DISCOVERABLE");
                        break;
                    case BluetoothAdapter.SCAN_MODE_CONNECTABLE:
                        Log.d(TAG, "mBroadcastReceiver2: SCAN_MODE_CONNECTABLE");
                        break;
                    case BluetoothAdapter.SCAN_MODE_NONE:
                        Log.d(TAG, "mBroadcastReceiver2: SCAN_MODE_NONE");
                        break;
                    case BluetoothAdapter.STATE_CONNECTING:
                        Log.d(TAG, "mBroadcastReceiver2: STATE_CONNECTING");
                        break;
                    case BluetoothAdapter.STATE_CONNECTED:
                        Log.d(TAG, "mBroadcastReceiver2: STATE_CONNECTED ");
                        break;
                }
```

```
            }
        }
    };

    private final BroadcastReceiver mBroadcastReceiver3 = new BroadcastReceiver() {
        public void onReceive(Context context, Intent intent) {
            final String action = intent.getAction();
            if (action.equals(BluetoothDevice.ACTION_FOUND)) {
                BluetoothDevice device =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                adapter.add(device);
                Log.d(TAG, "adapter add device name" + device.getName());
            }
        }
    };



    private final BroadcastReceiver mBroadcastReceiver4 = new BroadcastReceiver() {
        public void onReceive(Context context, Intent intent) {
            final String action = intent.getAction();
            if (action.equals(BluetoothDevice.ACTION_BOND_STATE_CHANGED)) {
                BluetoothDevice mDevice =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);


                //3 cases
                //case1: bonded already
                if (mDevice.getBondState() == BluetoothDevice.BOND_BONDED) {
                    Log.d(TAG, "mBroadcastReceiver4: BOND_BONDED");
                    Toast.makeText(MainActivity.this, "mBroadcastReceiver4:
BOND_BONDED",Toast.LENGTH_SHORT).show();
                    mBTDevice = mDevice;
                } else if (mDevice.getBondState() == BluetoothDevice.BOND_BONDING) {
                    Log.d(TAG, "mBroadcastReceiver4: BOND_BONDING");
                } else if (mDevice.getBondState() == BluetoothDevice.BOND_NONE) {
                    Log.d(TAG, "mBroadcastReceiver4: BOND_NONE");
                }
            }
        }
    };

    public void enableDisableBluetooth() {
```

```
        if (mBluetoothAdapter == null) {
            Log.d(TAG, "No bluetooth available");
            return;
        }

        if (!mBluetoothAdapter.isEnabled()) {
            Intent enableBTIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivity(enableBTIntent);
            //Filter intent by action
            IntentFilter BTIntent = new
IntentFilter(BluetoothAdapter.ACTION_STATE_CHANGED);
            registerReceiver(mBroadcastReceiver1, BTIntent);
        } else {
            mBluetoothAdapter.disable();
            IntentFilter BTIntent = new
IntentFilter(BluetoothAdapter.ACTION_STATE_CHANGED);
            registerReceiver(mBroadcastReceiver1, BTIntent);
        }
    }

    public void btnEnableDisable_Discoverable(View view) {
        Log.d(TAG, "btnEnableDisable_Discoverable: Making device discoverable for 300
seconds");
        Intent discoverableIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
        discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION,  300);
        startActivity(discoverableIntent);

        IntentFilter intentFilter = new
IntentFilter(mBluetoothAdapter.ACTION_SCAN_MODE_CHANGED);
        registerReceiver(mBroadcastReceiver2, intentFilter);
    }

    public void btn_Discover(View view) {
        Log.d(TAG, "btnDiscover: Looking for unpaired devices.");
        adapter.clear();
        if (mBluetoothAdapter.isDiscovering()) {
            mBluetoothAdapter.cancelDiscovery();
            Log.d(TAG, "btnDiscover: Canceling discovery");
            checkBTPermissions();
            mBluetoothAdapter.startDiscovery();
        } else {
```

```
            checkBTPermissions();
            mBluetoothAdapter.startDiscovery();
        }

        IntentFilter discoveryDevicesIntent = new
IntentFilter(BluetoothDevice.ACTION_FOUND);
        registerReceiver(mBroadcastReceiver3, discoveryDevicesIntent);
    }

    /*
    This method is required for all devices running API23+
    Android must programmatically check the permissions for bluetooth. Putting the
proper permissions in the manifest is not enough.
     */
    private void checkBTPermissions() {
        if (Build.VERSION.SDK_INT > Build.VERSION_CODES.LOLLIPOP) {
            int permissionCheck =
this.checkSelfPermission(Manifest.permission.ACCESS_FINE_LOCATION);
            permissionCheck +=
this.checkSelfPermission(Manifest.permission.ACCESS_COARSE_LOCATION);
            if (permissionCheck != 0) {
                this.requestPermissions(new
String[]{Manifest.permission.ACCESS_FINE_LOCATION,
Manifest.permission.ACCESS_COARSE_LOCATION}, 1001);
            }
        }
        else {
            Log.d(TAG, "checkBTPermissions: No need to check permission. SDK version <
LOLLIPOP");
        }
    }

    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
        mBluetoothAdapter.cancelDiscovery();
        Log.d(TAG, "onItemClick: You click on a device");
        String deviceName = mBTDevices.get(i).getName();
        String deviceAddress = mBTDevices.get(i).getAddress();

        Log.d(TAG, "Device name is " + deviceName);
        Log.d(TAG, "Device address is " + deviceAddress);
```

```
        if (Build.VERSION.SDK_INT > Build.VERSION_CODES.JELLY_BEAN_MR2) {
            Log.d(TAG, "Trying to pair with " + deviceName);
            mBTDevice = mBTDevices.get(i);
            if (mBluetoothAdapter.getBondedDevices().contains(mBTDevice)) {
                Toast.makeText(this, "This device is already
paired",Toast.LENGTH_SHORT).show();
            } else {
                boolean flag = mBTDevices.get(i).createBond();
                if (flag) {
                    Log.d(TAG, "Bonding begin!");
                } else {
                    Log.d(TAG, "Some immediate error happen");
                }
            }
            mBluetoothConnection =
BluetoothConnectionService.getInstance(MainActivity.this);
        }
    }

    @Override
    protected void onDestroy() {
        Log.d(TAG, "onDestroy is called.");
        super.onDestroy();
        unregisterReceiver(mBroadcastReceiver1);
        unregisterReceiver(mBroadcastReceiver2);
        unregisterReceiver(mBroadcastReceiver3);
        unregisterReceiver(mBroadcastReceiver4);
        LocalBroadcastManager.getInstance(this).unregisterReceiver(mReceiver);
    }
}
```

DeviceAdapter.java

```
package com.example.hang.bluetoothdatatest;
public class DeviceAdapter extends ArrayAdapter<BluetoothDevice> {
    public DeviceAdapter(Context context, ArrayList<BluetoothDevice> devices) {
        super(context, 0, devices);
    }
```

```
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        // Get the data item for this position
        BluetoothDevice device = getItem(position);
        // Check if an existing view is being reused, otherwise inflate the view
        if (convertView == null) {
            convertView =
LayoutInflater.from(getContext()).inflate(R.layout.device_item, parent, false);
        }

        // Lookup view for data population
        TextView tvName = (TextView) convertView.findViewById(R.id.tvName);
        TextView tvAddress = (TextView) convertView.findViewById(R.id.tvAddress);
        // Populate the data into the template view using the data object
        tvName.setText(device.getName());
        tvAddress.setText(device.getAddress());
        // Return the completed view to render on screen
        return convertView;
    }
}
```

Position.java

```
public class Position {
    private String label;
    private double RSSI;
    public Position(String label, double RSSI) {
        this.label = label;
        this.RSSI = RSSI;
    }

    public String getLabel() {
        return label;
    }

    public double getRSSI() {
        return RSSI;
    }
}
```

PositionAdapter.java

```
package com.example.hang.bluetoothdatatest;
public class PositionAdapter extends ArrayAdapter<Position> {
    public PositionAdapter(Context context, ArrayList<Position> positions) {
        super(context, 0, positions);
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        // Get the data item for this position
        Position curPos = getItem(position);
        // Check if an existing view is being reused, otherwise inflate the view
        if (convertView == null) {
            convertView =
LayoutInflater.from(getContext()).inflate(R.layout.position_item, parent, false);
        }

        // Lookup view for data population
        TextView tv_label = (TextView) convertView.findViewById(R.id.tv_label);
        TextView tv_RSSI = (TextView) convertView.findViewById(R.id.tv_RSSI);
        // Populate the data into the template view using the data object
        tv_label.setText(curPos.getLabel());
        tv_RSSI.setText(String.valueOf(curPos.getRSSI()));
        // Return the completed view to render on screen
        return convertView;
    }
}
```

SearchModeActivity.java

```
package com.example.hang.bluetoothdatatest;
public class SearchModeActivity extends AppCompatActivity {
    BluetoothConnectionService bluetoothConnectionService;
    final static String TAG= "SearchModeActiviry";
    private Button btn_addLabel;
    private Button btn_stop;
    private TextView textView;
    private Handler handler;
    private boolean sendAndreceiveData = false;
```

```
private ListView listview_label;
private List<Integer> tempRSSIlist;
private List<Integer> tempSNRlist;
private List<Integer> revisedRSSIlist;
private PositionAdapter adapter;
private ArrayList<Position> positionList;
Set<String> labels;
String curLabel = null;
int count = 0;

@Override
protected void onDestroy() {
    LocalBroadcastManager.getInstance(this).unregisterReceiver(mReceiver);
    super.onDestroy();
}

BroadcastReceiver mReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        count++;
        StringBuilder messages = new StringBuilder();
        messages.append("index = " + count + "\n");
        byte[] bytes = intent.getByteArrayExtra("MessageByteArray");
        //messages.append(text + "\n");
        int RSSI = 0;
        int SNR = 0;
        for (int i = 0; i < bytes.length; i++) {
            byte b = bytes[i];
            int v2;
            if (i == 1) {
                //RSSI
                v2 = b & 0xFF;
                v2 = (256-v2)*(-1);
                RSSI = v2;
            }  else if (i == 2) {
                //SNR
                v2 = b & 0xFF;
                if (v2 > 127) {
                    v2 = (256-v2)*(-1);
                }
```

```
                    SNR = v2;
                } else {
                    v2 = b & 0xFF;
                }
                messages.append(v2 + " ");
            }
            if (sendAndreceiveData) {
                textView.setText(messages.toString());
                Log.d(TAG, "get RSSI = " + messages.toString());
                if ((bytes[0] & 0xFF) == 0x01) {
                    tempRSSIlist.add(RSSI);
                    tempSNRlist.add(SNR);

                    if (SNR > 0) {
                        revisedRSSIlist.add(RSSI);
                    } else {
                        revisedRSSIlist.add((int)(RSSI + 0.25 * SNR));
                    }
                }
            }
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_search_mode_activiry);
        this.setTitle("SearchModeActivity");
        LocalBroadcastManager.getInstance(this).registerReceiver(mReceiver, new
IntentFilter("incomingMessage"));
        handler = new Handler(Looper.getMainLooper()) {
            @Override
            public void handleMessage(Message inputMessage) {
                switch (inputMessage.what) {
                    case 1:
                        //send command 2
                        byte[] bytes = "2".getBytes(Charset.defaultCharset());
                        bluetoothConnectionService.write(bytes);
                        handler.sendEmptyMessageDelayed(1, 2000);
//                          if (sendAndreceiveData) {
```

```
//                            handler.sendEmptyMessageDelayed(1, 4000);
//                        }
                        break;
                    default:
                        break;
                }
            }
        };

        tempRSSIlist = new ArrayList<>();
        tempSNRlist = new ArrayList<>();
        revisedRSSIlist = new ArrayList<>();
        positionList = new ArrayList<>();
        labels = new HashSet<>();
        adapter = new PositionAdapter(this, positionList);

        bluetoothConnectionService = BluetoothConnectionService.getInstance();
        btn_addLabel = (Button) findViewById(R.id.btn_addLabel);
        btn_stop = (Button) findViewById(R.id.btn_stop);
        textView = (TextView) findViewById(R.id.textView);
        listview_label = (ListView) findViewById(R.id.lv_label);
        listview_label.setAdapter(adapter);

        final View view = getLayoutInflater().inflate(R.layout.dialog_add_label, null);
        final AlertDialog alertDialog = new AlertDialog.Builder(this).create();
        alertDialog.setTitle("Add a label");
        alertDialog.setCancelable(false);
        final EditText et_addLabel = (EditText) view.findViewById(R.id.etLabel);
        alertDialog.setButton(AlertDialog.BUTTON_POSITIVE, "OK", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                curLabel = et_addLabel.getText().toString();
                et_addLabel.setText("");
                if (labels.contains(curLabel)) {
                    Toast.makeText(SearchModeActivity.this, "This label is already
included!", Toast.LENGTH_SHORT).show();
                    return;
                }

                Log.d(TAG, "A new label is added as " + curLabel);
```

```
                sendAndreceiveData = true;
                //sent command 2 every 4 second.
//                Message msg = Message.obtain();
//                msg.what = 1;
//                handler.sendMessage(msg);
                handler.sendEmptyMessage(1);
                count = 0;
            }
        });

        alertDialog.setButton(AlertDialog.BUTTON_NEGATIVE, "Cancel", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                alertDialog.dismiss();
            }
        });
        alertDialog.setView(view);
        btn_addLabel.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                alertDialog.show();
            }
        });

        btn_stop.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                sendAndreceiveData = false;
//                Log.d(TAG, "Stop button is clicked");
                handler.removeMessages(1);
                double aveRSSI = getAverage();

                StringBuilder s = new StringBuilder();
                s.append("RSSI:");
                for (int i = 0; i < tempRSSIlist.size(); i++) {
                    s.append(tempRSSIlist.get(i) + ",");
                }
                s.append("\n");
                s.append("SNR:");
```

```
                for (int i = 0; i < tempSNRlist.size(); i++) {
                    s.append(tempSNRlist.get(i) + ",");
                }

                s.append("\n");
                s.append("revised RSSI:");
                for (int i = 0; i < revisedRSSIlist.size(); i++) {
                    s.append(revisedRSSIlist.get(i) + ",");
                }
                textView.setText(s);
                tempRSSIlist.clear();
                tempSNRlist.clear();
                revisedRSSIlist.clear();

                labels.add(curLabel);

                Position pos = new Position(curLabel, aveRSSI);
                adapter.add(pos);
            }
        });
    }

    private double getAverage() {
        if (revisedRSSIlist.size() == 0) {
            return -1;
        }
        double sum = 0;
        for (int i = 0; i < revisedRSSIlist.size(); i++) {
            sum += revisedRSSIlist.get(i);
        }
        return sum / revisedRSSIlist.size();
    }
}
```

OneDimensionSampleActivity.java

```
package com.example.hang.bluetoothdatatest;
public class OneDimensionSampleActivity extends AppCompatActivity {
    private BroadcastReceiver mReceiver = new BroadcastReceiver() {
        @Override
```

```
        public void onReceive(Context context, Intent intent) {
            byte[] bytes = intent.getByteArrayExtra("MessageByteArray");
            //messages.append(text + "\n");
            int RSSI = 0;
            int SNR = 0;
            for (int i = 0; i < bytes.length; i++) {
                byte b = bytes[i];
                int v2;
                if (i == 1) {
                    //RSSI
                    v2 = b & 0xFF;
                    v2 = (256-v2)*(-1);
                    RSSI = v2;
                }  else if (i == 2) {
                    //SNR
                    v2 = b & 0xFF;
                    if (v2 > 127) {
                        v2 = (256-v2)*(-1);
                    }
                    SNR = v2;
                } else {
                    v2 = b & 0xFF;
                }
            }
            double revisedRSSI = RSSI;
            if (SNR < 0) {
                revisedRSSI = RSSI + 0.25 * SNR;
            }
            strengthList.add(revisedRSSI);
            textView.setText("RSSI="+RSSI+";SNR="+SNR+";revisedRSSI="+revisedRSSI);
        }
    };
    private BluetoothConnectionService bluetoothConnectionService;
    private Button btn_StartCollect;
    private Button btn_Stop;
    private Button btn_Process;
    private TextView textView;
    private final ScheduledExecutorService scheduler =
Executors.newScheduledThreadPool(1);
    private final Runnable sendCommandRunnable = new Runnable() {
```

```
        @Override
        public void run() {
            byte[] bytes = "2".getBytes(Charset.defaultCharset());
            bluetoothConnectionService.write(bytes);
        }
    };
    private ScheduledFuture<?> SendCommandHandler;
    private ArrayList<Double> strengthList;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_one_dimension_sample);
        btn_StartCollect = (Button) findViewById(R.id.btn_StartCollect);
        btn_Stop = (Button) findViewById(R.id.btn_StopCollect);
        btn_Process = (Button) findViewById(R.id.btn_Process);
        textView = (TextView) findViewById(R.id.textView1);
        LocalBroadcastManager.getInstance(this).registerReceiver(mReceiver, new
IntentFilter("incomingMessage"));
        bluetoothConnectionService = BluetoothConnectionService.getInstance();
        strengthList = new ArrayList<>();

        btn_StartCollect.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                strengthList.clear();
                SendCommandHandler = scheduler.scheduleAtFixedRate(sendCommandRunnable,
0, 2, TimeUnit.SECONDS);
            }
        });

        btn_Stop.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                scheduler.schedule(new Runnable() {
                    @Override
                    public void run() {
                        SendCommandHandler.cancel(true);
                    }
                }, 0, TimeUnit.SECONDS);
                String str = listToStr();
```

```
                textView.setText("revisedRSSI list : " + str);
            }
        });

        btn_Process.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(OneDimensionSampleActivity.this,
PolyFitActivity.class);
                intent.putExtra("array", strengthList);
                startActivity(intent);
            }
        });
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        LocalBroadcastManager.getInstance(this).unregisterReceiver(mReceiver);
    }

    private String listToStr() {
        StringBuilder builder = new StringBuilder();
        for (double strength : strengthList) {
            builder.append(strength + ";");
        }
        return builder.toString();
    }
}
```

PolyFitActivity.java

```
package com.example.hang.bluetoothdatatest;
public class PolyFitActivity extends AppCompatActivity {
    private XYPlot plot;
    private TextView tv_result;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_poly_fit);


        plot = (XYPlot) findViewById(R.id.plot);
        tv_result = (TextView) findViewById(R.id.tv_result);


        Intent intent = getIntent();
        ArrayList<Double> list =
(ArrayList<Double>)intent.getSerializableExtra("array");
        double[] original = new double[list.size()];
        for (int i = 0; i < list.size(); i++) {
            original[i] = list.get(i);
        }
        double[] filtered = new AverageFilter(2, original).filter();
        int len = original.length;
        // create a couple arrays of y-values to plot:
        Number[] series1Numbers = getSeriesNumbers(original);
        // (Y_VALS_ONLY means use the element index as the x value)
        XYSeries series1 = new SimpleXYSeries(Arrays.asList(series1Numbers),
SimpleXYSeries.ArrayFormat.Y_VALS_ONLY, "Or");
        // create formatters to use for drawing a series using LineAndPointRenderer
        // and configure them from xml:
        LineAndPointFormatter series1Format = new LineAndPointFormatter(this,
R.xml.point_formatter);
        series1Format.setInterpolationParams(new CatmullRomInterpolator.Params(10,
CatmullRomInterpolator.Type.Centripetal));
        // add a new series' to the xyplot:
        plot.addSeries(series1, series1Format);


        Number[] series2Numbers = getSeriesNumbers(filtered);
        XYSeries series2 = new SimpleXYSeries(
                Arrays.asList(series2Numbers), SimpleXYSeries.ArrayFormat.Y_VALS_ONLY,
"F");
        LineAndPointFormatter series2Format = new LineAndPointFormatter(this,
R.xml.line_point_formatter_purple);
        series2Format.setInterpolationParams(new CatmullRomInterpolator.Params(10,
CatmullRomInterpolator.Type.Centripetal));
        plot.addSeries(series2, series2Format);


        PolyFit polyFit = new PolyFit(filtered);
        polyFit.init();
        double[] oneOrderFitRes = polyFit.oneOrderFit();
        LineAndPointFormatter series3Format = new LineAndPointFormatter(this,
R.xml.line_point_formatter_red);
```

```
        series3Format.setInterpolationParams(new CatmullRomInterpolator.Params(10,
CatmullRomInterpolator.Type.Centripetal));
        XYSeries series3 = generateSeries(0, len-1, len*10, 1, oneOrderFitRes[0],
oneOrderFitRes[1], 0);
        plot.addSeries(series3, series3Format);

        //get y_predicted
        double[] yOneOrder_predicted = new double[filtered.length];
        for (int i = 0; i < len; i++) {
            yOneOrder_predicted[i] = OneOrderfx(i, oneOrderFitRes[0],
oneOrderFitRes[1]);
        }
        RSquared rSuqred1 = new RSquared(filtered, yOneOrder_predicted, 2);
        double rsquared = rSuqred1.getRSquared();
        double adrsquared = rSuqred1.getAdjustedRSquared();
        double[] twoOrderFitRes = polyFit.twoOrderFit();
        LineAndPointFormatter series4Format = new LineAndPointFormatter(this,
R.xml.line_point_formatter_green);
        series4Format.setInterpolationParams(new CatmullRomInterpolator.Params(10,
CatmullRomInterpolator.Type.Centripetal));
        XYSeries series4 = generateSeries(0, len-1, len*10, 2, twoOrderFitRes[0],
twoOrderFitRes[1], twoOrderFitRes[2]);
        plot.addSeries(series4, series4Format);
        double[] yTwoOrder_predicted = new double[filtered.length];
        for (int i = 0;i < len; i++) {
            yTwoOrder_predicted[i] = TwoOrderfx(i, twoOrderFitRes[0],
twoOrderFitRes[1], twoOrderFitRes[2]);
        }
        RSquared rSuqred2 = new RSquared(filtered, yTwoOrder_predicted, 3);
        double rsquared2 = rSuqred2.getRSquared();
        double adjustedR2 = rSuqred2.getAdjustedRSquared();

        tv_result.setText("OneOrderFit Adjust_R2="+adrsquared+"\n"+"TwoOrderFit
Adjust_R2="+adjustedR2);
    }

    Number[] getSeriesNumbers(double[] array) {
        Number[] nums = new Number[array.length];
        for (int i = 0; i < nums.length; i++) {
            nums[i] = array[i];
        }
        return nums;
    }
```

```
    protected XYSeries generateSeries(double minX, double maxX, double resolution, int
order, double a, double b, double c) {
        //the value of order only has two choices: 1 or 2. Otherwise, throw exception
        final double range = maxX - minX;
        final double step = range / resolution;
        List<Number> xVals = new ArrayList<>();
        List<Number> yVals = new ArrayList<>();
        if (order == 1) {
            double x = minX;
            while (x <= maxX) {
                xVals.add(x);
                yVals.add(OneOrderfx(x, a, b));
                x +=step;
            }
            return new SimpleXYSeries(xVals, yVals, "One");
        } else {
            double x = minX;
            while (x <= maxX) {
                xVals.add(x);
                yVals.add(TwoOrderfx(x, a, b, c));
                x +=step;
            }
            return new SimpleXYSeries(xVals, yVals, "Two");
        }
    }

    protected double TwoOrderfx(double x, double a, double b, double c) {
        return a*Math.abs(x*x) + b*x + c;
    }

    protected double OneOrderfx(double x, double a, double b) {
        return a*x + b;
    }

    double[] getYValules(XYSeries series) {
        double[] yVals = new double[series.size()];
        for (int i = 0; i <  yVals.length; i++) {
            yVals[i] = series.getY(i).doubleValue();
        }
```

```
        return yVals;
    }
}
```

AverageFilter.java

```
package com.example.hang.bluetoothdatatest.PolyFitLib;
public class AverageFilter {
    int windowSize;
    double[] yArray;
    public AverageFilter(int size, double[] array) {
        windowSize = size;
        yArray = array;
    }

    public double[] filter() {
        double[] res = new double[yArray.length];
        for (int i = windowSize-1; i < res.length; i++) {
            //find res[i]
            double temp = 0;
            for (int j = i+1-windowSize; j <= i; j++) {
                temp += yArray[j];
            }
            res[i] = temp/windowSize;
        }

        for (int i = 0; i <  windowSize; i++) {
            res[i] = yArray[i];
        }
        return res;
    }
}
```

Expectation.java

```
package com.example.hang.bluetoothdatatest.PolyFitLib;
public class Expectation {
    double[] xArray;
    double[] yArray;
    public Expectation(double[] xArray, double[] yArray) {
```

```
    //the length of two arrays should be the same
    this.xArray = xArray;
    this.yArray= yArray;
}

double getExpectOfX() {
    double sum = 0;
    for (double temp : xArray) {
        sum += temp;
    }
    return sum/xArray.length;
}

double getExpectOfY() {
    double sum = 0;
    for (double temp : yArray) {
        sum += temp;
    }
    return sum/yArray.length;
}

double getExpectOfXY() {
    double sum = 0;
    for (int i = 0; i < xArray.length; i++) {
        double x = xArray[i];
        double y = yArray[i];
        sum += x*y;
    }
    return sum/yArray.length;
}

double getExpectOfX2() {
    double sum = 0;
    for (double temp : xArray) {
        sum += temp*temp;
    }
    return sum/xArray.length;
}

double getExpectOfX3() {
```

```
        double sum = 0;
        for (double temp : xArray) {
            sum += temp*temp*temp;
        }
        return sum/xArray.length;
    }

    double getExpectOfX4() {
        double sum = 0;
        for (double temp : xArray) {
            sum += temp*temp*temp*temp;
        }
        return sum/xArray.length;
    }

    double getExpectOfX2Y() {
        double sum = 0;
        for (int i = 0; i < xArray.length; i++) {
            double x = xArray[i];
            double y = yArray[i];
            sum += x*x*y;
        }
        return sum/yArray.length;
    }
}
```

PolyFit.java

```
package com.example.hang.bluetoothdatatest.PolyFitLib;
public class PolyFit {
    private double[] xArray;
    private double[] yArray;
    private Expectation expectation;

    private double expectofX;
    private double expectofY;
    private double expectofXY;
    private double expectofX2Y;
    private double expectofX2;
    private double expectofX3;
```

```
    private double expectofX4;

    public PolyFit(double[] yArray) {
        this.yArray = yArray;
        xArray = new double[yArray.length];
        for (int i = 0; i < yArray.length; i++) {
            xArray[i] = i+1;
        }
        expectation = new Expectation(xArray, this.yArray);
    }

    //must call init first before calling oneOrderFit and twoOrderFit functions
    public void init() {
        expectofX = expectation.getExpectOfX();
        expectofY = expectation.getExpectOfY();
        expectofXY = expectation.getExpectOfXY();
        expectofX2Y = expectation.getExpectOfX2Y();
        expectofX2 = expectation.getExpectOfX2();
        expectofX3 = expectation.getExpectOfX3();
        expectofX4 = expectation.getExpectOfX4();
    }

    public double[] oneOrderFit() {
        double[][] left = new double[][] { { expectofX, 1 }, {expectofX2,
expectofX } };
        double[] right = new double[] {expectofY, expectofXY};
        RealMatrix coefficients =new Array2DRowRealMatrix(left, false);
        DecompositionSolver solver = new LUDecomposition(coefficients).getSolver();
        RealVector constants = new ArrayRealVector(right, false);
        RealVector solution = solver.solve(constants);

        double a = solution.getEntry(0);
        double b = solution.getEntry(1);
        return new double[] {a, b};
    }

    public double[] twoOrderFit() {
        double[][] left = new double[][] { { expectofX2, expectofX, 1 }, { expectofX3,
expectofX2, expectofX }, { expectofX4, expectofX3, expectofX2 } };
        double[] right = new double[] {expectofY, expectofXY, expectofX2Y};
        RealMatrix coefficients =new Array2DRowRealMatrix(left, false);
```

```
        DecompositionSolver solver = new LUDecomposition(coefficients).getSolver();
        RealVector constants = new ArrayRealVector(right, false);
        RealVector solution = solver.solve(constants);

        double a = solution.getEntry(0);
        double b = solution.getEntry(1);
        double c = solution.getEntry(2);
        return new double[] {a,b,c};
    }
}
```

Rsquared.java

```
package com.example.hang.bluetoothdatatest.PolyFitLib;
public class RSquared {
    double[] yArray;
    double[] yPredict;
    private double average;
    private double SS_tot;
    private double SS_reg;
    private double SS_res;
    int p;
    public RSquared(double[] yArray, double[] yPredict, int p) {
        this.yArray = yArray;
        this.yPredict = yPredict;
        this.p = p;
        average = getAverageY();
        SS_tot = getSS_tot();
        SS_reg = getSS_reg();
        SS_res = getSS_res();
    }

    private double getAverageY() {
        double sum = 0;
        for (double temp : yArray) {
            sum += temp;
        }
        return sum/yArray.length;
    }
```

```
    private double getSS_tot() {
        double sum = 0;
        for (int i = 0; i < yArray.length; i++) {
            double temp = (yArray[i]-average)*(yArray[i]-average);
            sum += temp;
        }
        return sum;
    }

    private double getSS_reg() {
        double sum = 0;
        for (int i = 0; i < yArray.length; i++) {
            double temp = (yPredict[i]-average)*(yPredict[i]-average);
            sum += temp;
        }
        return sum;
    }

    private double getSS_res() {
        double sum = 0;
        for (int i = 0; i < yArray.length; i++) {
            double temp = (yPredict[i]-yArray[i])*(yPredict[i]-yArray[i]);
            sum += temp;
        }
        return sum;
    }

    public double getRSquared() {
        return 1-SS_res/SS_tot;
    }

    public double getAdjustedRSquared() {
        int n = yArray.length;
        return 1-SS_res/SS_tot*(n-1)/(n-p);
    }
}
```

APPENDIX C. Android APP on the 2.4GHz Emitter

MainActivity.java

```
package com.example.hang.client;
public class MainActivity extends AppCompatActivity {
    Socket socket;
    String SOCKET_HOST = "192.168.3.50";
    DataOutputStream out;
    DataInputStream in;

    TextView tv_status;
    BlockingDeque<Integer> queue;
    private final ScheduledExecutorService scheduler =
Executors.newScheduledThreadPool(1);
    private ScheduledFuture<?> RSSIStrengthHandler;
    final Runnable fetchRSSI = new Runnable() {
        @Override
        public void run() {
            int strength = getSignalStrength();
            queue.add(strength);

            Message msg = Message.obtain();
            msg.what = 0;
            msg.obj = String.valueOf(strength);
            myHandler.sendMessage(msg);
        }
    };
    final Runnable stopFetchRSSI = new Runnable() {
        @Override
        public void run() {
            RSSIStrengthHandler.cancel(true);
            myHandler.sendEmptyMessage(1);
        }
    };

    Handler myHandler = new Handler(Looper.getMainLooper()) {
        @Override
        public void handleMessage(Message inputMessage) {
```

```
        // Gets the image task from the incoming Message object.
        if (inputMessage.what == 0) {
            String str = inputMessage.obj.toString();
            tv_status.setText("RSSI="+str);
        } else {
            tv_status.setText("");
        }
    }
};


@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    tv_status = (TextView) findViewById(R.id.tv_status);
    queue = new LinkedBlockingDeque<>();

    //create socket connection
    Runnable runnable = new Runnable() {
        @Override
        public void run() {
            try {
                socket = new Socket(SOCKET_HOST, 12345);
                out = new DataOutputStream(socket.getOutputStream());
                in = new DataInputStream(socket.getInputStream());
            } catch (UnknownHostException e) {
                System.out.println("Unknown host: " + SOCKET_HOST);
                System.exit(1);
            } catch (IOException e) {
                System.out.println("No I/O");
                System.exit(1);
            }
        }
    };
    Thread connectThread = new Thread(runnable);
    connectThread.start();
    try {
        connectThread.join();
    } catch (InterruptedException e) {
```

```
            e.printStackTrace();
        }

        Runnable readTask = new Runnable() {
            @Override
            public void run() {
                while (true) {
                    try {
                        String message = in.readUTF();
                        System.out.println("Get command from Server : " + message);
                        if (message.equals("1")) {
                            RSSIStrengthHandler =
scheduler.scheduleAtFixedRate(fetchRSSI, 0, 1, TimeUnit.SECONDS);
                        } else {
                            scheduler.schedule(stopFetchRSSI, 0, TimeUnit.SECONDS);
                        }
                    } catch (IOException e) {
                        System.out.println("Read failed");
                        System.exit(-1);
                    }
                }
            }
        };
        new Thread(readTask).start();


        Runnable writeTask = new Runnable() {
            @Override
            public void run() {
                while (true) {
                    Integer RSSI = null;
                    try {
                        RSSI = queue.poll(300, TimeUnit.MILLISECONDS);
                        if (RSSI != null) {
                            System.out.println("writeTask: strength = " + RSSI);
                            out.writeUTF(String.valueOf(RSSI));
                            out.flush();
                        }
                    } catch (InterruptedException e) {
                        e.printStackTrace();
```

```
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }

            }
        };
        new Thread(writeTask).start();
    }



    private int getSignalStrength() {
        WifiManager wifiManager = (WifiManager)
getApplicationContext().getSystemService(Context.WIFI_SERVICE);
        wifiManager.startScan();
        List<ScanResult> wifiList = wifiManager.getScanResults();
        if (wifiList.size() == 0) {
            return 100;
        }

        for (int i = 0; i < wifiList.size(); i++) {
            ScanResult scanResult = wifiList.get(i);
            String macAddress = scanResult.BSSID;
            if (macAddress.equals("d0:ff:98:81:46:f8")) {
                return scanResult.level;
            }
        }
        return 101;
    }
}
```

APPENDIX D. Android APP on the 2.4GHz Radio Tester

MainActivity.java

```
package com.example.hang.server;

public class MainActivity extends AppCompatActivity implements
LabelDialog.LabelDialogListener {

    ServerSocket server;
```

```
Socket client;
Worker worker;
Button btn_startCollect;
TextView tv_status;
private ListView listView;
String label = null;
List<Integer> strengthList;
private PositionAdapter positionAdapter;
private List<Position> list;

Handler myHandler = new Handler(Looper.getMainLooper()) {
    @Override
    public void handleMessage(Message inputMessage) {
        // Gets the image task from the incoming Message object.
        if (inputMessage.what == 0) {
            String str = inputMessage.obj.toString();
            tv_status.setText("RSSI = " + str);
            strengthList.add(Integer.valueOf(str));
        } else {
            tv_status.setText("");
        }
    }
};
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    btn_startCollect = (Button) findViewById(R.id.btn_startCollect);
    tv_status = (TextView) findViewById(R.id.tv_status);
    listView = (ListView) findViewById(R.id.lv_label);
    strengthList = new ArrayList<>();
    list = new ArrayList<>();
    positionAdapter = new PositionAdapter(this, list);
    listView.setAdapter(positionAdapter);

    try {
        server = new ServerSocket(12345);
    } catch (IOException e) {
        System.out.println("Could not listen on port 12345");
        System.exit(-1);
```

```
    }

    Runnable runnable = new Runnable() {
        @Override
        public void run() {
            try {
                client = server.accept();
                System.out.println("New client");
                worker = new Worker(client, myHandler);
            } catch (IOException e) {
                System.out.println("Accept failed: 12345");
                System.exit(-1);
            }
        }
    };
    Thread connectThread = new Thread(runnable);
    connectThread.start();


    btn_startCollect.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            String buttonText = btn_startCollect.getText().toString();
            if (buttonText.equals("Start to collect")) {
                openDialog();
            } else {
                worker.addCommand(0);
                btn_startCollect.setText("Start to collect");

                //process strengthList
                double ave = getAvearge();
                tv_status.setText("Label="+label+";     aveRSSI="+ave);
                System.out.println("Label="+label+";aveRSSI="+ave);
                positionAdapter.add(new Position(label, ave));
            }
        }
    });
}

public void openDialog() {
```

```
        LabelDialog labelDialog = new LabelDialog();
        labelDialog.show(getSupportFragmentManager(), "label dialog");
    }


    @Override
    public void applyText(String label) {
        this.label = label;
        tv_status.setText("label is " + label);
        strengthList.clear();
        worker.addCommand(1);
        btn_startCollect.setText("Stop");
    }


    private double getAvearge() {
        double res = 0;
        for (int val : strengthList) {
            res += val;
        }
        if (strengthList.size() == 0) {
            return 0;
        } else {
            return res/strengthList.size();
        }
    }
}
```

Worker.java

```
package com.example.hang.server;
public class Worker {
    DataInputStream in;
    DataOutputStream out;
    Socket client;
    BlockingDeque<Integer> queue;
    Handler handler;
    public Worker(Socket client, Handler myHandler) {
        this.client = client;
        handler = myHandler;
```

```
    queue = new LinkedBlockingDeque<>();
    try {
        in = new DataInputStream(client.getInputStream());
        out = new DataOutputStream(client.getOutputStream());

    } catch (IOException e) {
        System.out.println("Read failed");
        System.exit(-1);
    }

    new Thread(writeTask).start();
    new Thread(readTask).start();
}

Runnable writeTask = new Runnable() {
    @Override
    public void run() {
        while (true) {
            Integer command = null;
            try {
                command = queue.poll(300, TimeUnit.MILLISECONDS);
                if (command != null) {
                    System.out.println("command="+command);
                    out.writeUTF(String.valueOf(command));
                    out.flush();
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
};

Runnable readTask = new Runnable() {
    @Override
    public void run() {
        while (true) {
            try {
```

```
                    String message = in.readUTF();
                    System.out.println("From Client : " + message);
                    Message msg = Message.obtain();
                    msg.what = 0;
                    msg.obj = message;
                    handler.sendMessage(msg);

                } catch (IOException e) {
                    System.out.println("Read failed");
                    System.exit(-1);
                }
            }

        }
    };


    public void addCommand(int command) {
        queue.add(command);
    }
}
```

LabelDialog.java

```
package com.example.hang.server;
public class LabelDialog extends AppCompatDialogFragment {
    private EditText editLabel;
    private LabelDialogListener listener;

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);

        try {
            listener = (LabelDialogListener) context;
        } catch (ClassCastException e) {
            throw new ClassCastException(context.toString() +"must implement
LabelDialogListener");
        }
    }
```

```
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        LayoutInflater inflater = getActivity().getLayoutInflater();
        View view = inflater.inflate(R.layout.layout_dialog, null);

        builder.setView(view)
                .setTitle("Label")
                .setNegativeButton("cancel", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {

                    }
                })
                .setPositiveButton("ok", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        String label = editLabel.getText().toString();
                        listener.applyText(label);

                    }
                });

        editLabel = view.findViewById(R.id.edit_label);
        return builder.create();
    }

    public interface LabelDialogListener {
        void applyText(String label);
    }
}
```

Position.java

```
package com.example.hang.server;
public class Position {
    private String label;
    private double RSSI;
    public Position(String label, double RSSI) {
```

```
        this.label = label;
        this.RSSI = RSSI;
    }

    public String getLabel() {
        return label;
    }

    public double getRSSI() {
        return RSSI;
    }
}
```

PositionAdapter.java

```
package com.example.hang.server;
public class PositionAdapter extends ArrayAdapter<Position> {
    public PositionAdapter(Context context, List<Position> positions) {
        super(context, 0, positions);
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        // Get the data item for this position
        Position curPos = getItem(position);
        // Check if an existing view is being reused, otherwise inflate the view
        if (convertView == null) {
            convertView =
LayoutInflater.from(getContext()).inflate(R.layout.position_item, parent, false);
        }

        // Lookup view for data population
        TextView tv_label = (TextView) convertView.findViewById(R.id.tv_label);
        TextView tv_RSSI = (TextView) convertView.findViewById(R.id.tv_RSSI);
        // Populate the data into the template view using the data object
        tv_label.setText(curPos.getLabel());
        tv_RSSI.setText(String.valueOf(curPos.getRSSI()));
        // Return the completed view to render on screen
        return convertView;
    }
```

```
}
```

APPENDIX E. Regression Analysis in MATLAB

```
y=-1*[65,53,56,53,60,55];
x = 1:length(y);
windowSize = 2;
b = (1/windowSize)*ones(1,windowSize);
a = 1;
y_filter = filter(b, a, y);
for i=1:windowSize
    y_filter(i) = y(i);
end

figure
plot(y_filter);
xlabel("Relative Distance");
ylabel("Received signal strength(dB)");

hold on
rho = getRho(x,y)

p2 = polyfit(x, y_filter, 2)
y2 = polyval(p2,x);
error2 = y_filter - y2;
MSE2 = 0;
for i=1:length(x)
    temp = error2(i)*error2(i);
    MSE2 = MSE2 + temp;
end
MSE2

plot(x,y,'o')
hold on
plot(x,y2)

p1 = polyfit(x, y_filter, 1)
y1 = polyval(p1, x);
error1  = y_filter - y1;
MSE1 = 0;
```

```
for i=1:length(x)
    temp = error1(i) * error1(i);
    MSE1 = MSE1 + temp;
end
MSE1
hold on
plot(x, y1, '--')
legend("Filter plot", "Sample data", "2nd-order fit", "Linear fit");

function sigma = getSigma(A, mu)
    temp = 0;
    for i=1:length(A)
        temp = temp+(A(i)-mu)*(A(i)-mu);
    end
    sigma = sqrt(temp/(length(A)-1));
end

function rho = getRho(x, y)
    meanX = mean(x);
    meanY = mean(y);
    sigmaX = getSigma(x, meanX);
    sigmaY = getSigma(y, meanY);
    sum = 0;
    for i = 1:length(x)
        sum = sum + (x(i)-meanX)*(y(i)-meanY);
    end

    cov = sum/(length(x)-1);
    rho = cov/sigmaX/sigmaY;
end
```