GCH1

FYT Progress Report

Indoor crowdsourced Wi-Fi fingerprinting with network embedding

by

ZHAO Ziqi

GCH1

Advised by

Prof. Gary Shueng-Han CHAN

April 29, 2020

Submitted in partial fulfillment
of the requirements for COMP 4981H
in the
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
2019-2020

1 Abstract

WiFi fingerprints are crucial in prevalent smart city applications, such as indoor localization and WiFi monitoring. Recent works on fingerprinting with crowdsourced data usually leverage inertial sensors, or radio propagation models to label collected signals. They may suffer from accumulative error of sensors, non-line-of-sight environments where models would fail. Other works use manifold alignment to estimate locations with sparse fingerprints, but may not be easily scalable due to the computation complexity. In this paper, we propose a highly scalable crowdsourced fingerprinting system based on pure WiFi signals without any assumption on signal propagation models. The system first generate an AP-observation network that captures the relationship between WiFi signals and APs, and then uses network embedding to infer dimension-reduced representations of WiFi signals. Finally the map matching system aligns these learned representations onto the map with sporadic location labels. Extensive results show that our proposed system can successfully construct the fingerprint database from crowdsourced WiFi signals, and the performance for localization is improved compared with the traditional method.

Contents

2 Introduction

2.1 Overview

Indoor WiFi fingerprint information is essential for various smart city applications, including localization services and WiFi heatmap monitoring. Each fingerprint[1] consists of location and corresponding Received Signal Strength Information (RSSI). Traditional approaches for fingerprinting (collection of fingerprints) is to hire professional surveyors with specially designed devices and collect WiFi signals at pre-designed locations, which is laborious and time consuming.

As such, fingerprinting based on crowdsensing has emerged as a promising technique due to its effectiveness in data collection. The task then becomes how to attach crowdsourced WiFi signals with location labels.

Recent works on crowdsensed fingerprinting usually take advantage of inertial motion units (IMUs) on mobile phones to predict user trajectories [1, 2], or leverage wireless signal propagation model [3] to estimate the distance between observation and Access Points (APs), thus label collected WiFi signals accordingly. There are also some works [4] which require only a small portion of fingerprints as references and use manifold alignment to directly match the WiFi signals to the physical locations based on the topological relationship between the learned WiFi signal manifold and the floor plan. Though efforts are made on crowdsensed fingerprinting, there are still challenges remaining:

- *Continuous recording of data*: IMU-based approaches require collected data to appear as continuous sequences, which pose restrictions on data collection thus may not be practical. In addition, mobile phone power drains fast in such applications.
- *Inaccurate distance estimation*: Wireless signal propagation model can be used to estimate physical distance under line-of-sight assumptions, but it is usually not the case indoors.
- *Scalability*: There are hundreds of APs on sites such as shopping malls, and a single spot is usually covered by much less APs. Manifold learning based approaches process WiFi data as uniform vectors whose length is the number of total APs, meaning that we need to fill in many missing values in those vectors. This greatly hinders scalability of such schemes.

[1] We refer to WiFi fingerprint as fingerprint unless otherwise stated.

To deal with these problems, this project proposes a novel indoor crowdsensed fingerprinting system that constructs fingerprint database without IMU support or assumption of signal propagation models, and is capable to evolve over time. The system first represent WiFi data with a bigraph that simulate real-world WiFi connections, then infers relative positions of WiFi signals in reduced signal space based on signal similarity, and finally uses few randomly collected location labels to pinpoint them on physical maps. This way, the crowdsensed WiFi signals are labelled and the fingerprint database is constructed.

We show an overview of our system in Figure 1. Our system first uses network embedding to learn latent representations (or embeddings) for WiFi data (or observations) such that more similar WiFi signals appear closer in embeddings. It then uses a semi-supervised map matching algorithm to match the embeddings into physical locations with a small portion of labelled data. Finally the estimated location labels with corresponding WiFi data are stored to fingerprint database. Our contributions can be summarized as follows:

- We represent WiFi data with a bigraph connecting WiFi signals and APs instead of a matrix. This data structure solves the RSSI missing value problem.
- To the best of our knowledge, this is the first work to use network embedding to estimate proximity relationships between WiFi patterns. We are able to construct fingerprint database without INS or wireless propagation models.
- We design a semi-supervised map matching algorithm to map observations from latent space to physical space. A multi-spring system is applied to obtain physical locations of observations given a small portion of labelled data.
- Extensive experiments have been conducted to evaluate the performance of our system. Results show that our proposed system can achieve location error of less than 1m with only less than 10 labelled data.

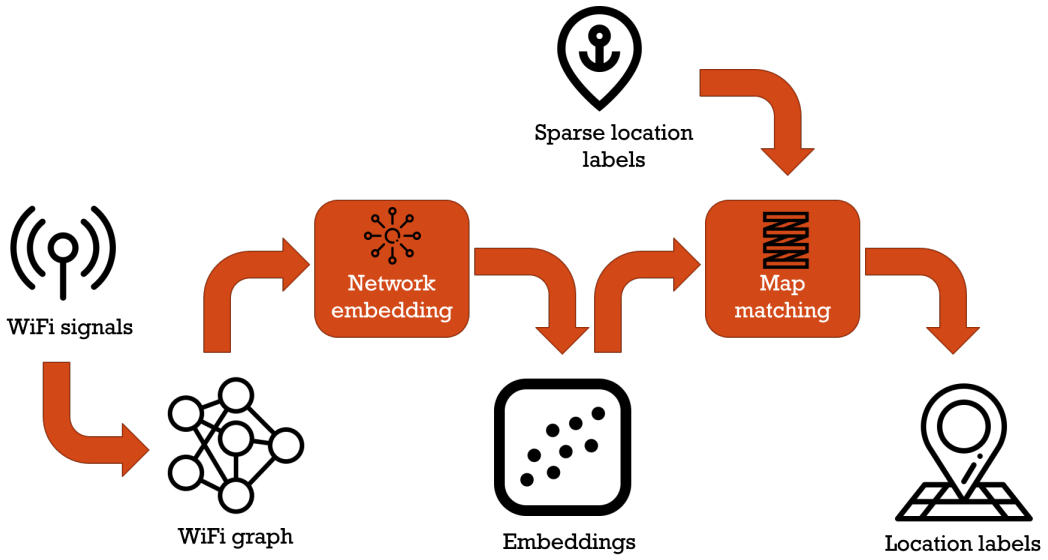


Figure 1: Overview of the system. WiFi signal vectors are processed with network embedding to get embedded coordinates, and then fed to map matching engine to obtain estimated labels.

2.2 Objectives

The goal of this project is to construct a Wi-Fi fingerprint database in indoor buildings from a few fingerprints and some crowdsourced data. To achieve this goal, the following objectives are going to be made:

1. Acquire Wi-Fi signal datasets from multiple buildings;
2. Build an algorithm that matches Wi-Fi signals to real-world coordinates;
3. Evaluate the performance of the proposed method in various aspects such as location accuracy and algorithmic complexity.

The first objective is accomplished by crowdsourced Wi-Fi signal collection and acquisition of prior fingerprints.

To reach the second objective, an AP-observation network will first be generated to model physical settings of APs and Wi-Fi signals. Then knowledge of network embedding is applied to predict relative distance between Wi-Fi signals as well as aligning them with physical coordinates.

Lastly, by visualizing cumulative distribution function (CDF) of location error of the algorithm and comparing it with state-of-the-art works, the third objective will be

achieved.

The biggest challenge of this project is the dependency of prior fingerprints. They can significantly reduce the error of fingerprint database, but require a lot of human labors at the same time. So a balance should be made carefully between the number of prior fingerprints and the location accuracy of the output fingerprint database.

2.3 Literature Survey

There are a variety of Wi-Fi fingerprinting methods under different categories, and we will discuss them below. Besides, since the proposed method uses graph learning, a short literature will also be described.

2.3.1 Wi-Fi fingerprinting

Wi-Fi fingerprinting has been broad studied, from traditional site surveys to more advanced ones. In the traditional site survey, professionals will record Received Signal Strength Indicator (RSSI) of all detected access points (APs) at each of reference points (RPs), which usually cover the place of interest in indoor environments. These RSSI vectors will be stored as a database in the server along with corresponding location labels. When a location query is received, the server will predict approximate location based on the similarity between the query RSSI vector and those in the database, according to K nearest neighbor mentioned in [5]. However, since site survey is time consuming and labor intensive, it's not easy for deployment in all the buildings worldwide.

Recently, there has been some advanced technology that can provide better solutions, either in improving the localization accuracy or reduce deployment difficulty. Some trending topics are shown as follows:

Crowdsourced Wi-Fi with inertial navigation system (INS) Since user's walking could provide an approximate distance information, and INS in mobile phones which contains an accelerometer and a gyroscope enable the estimation of user movement, there has been a wide range of researches that make use of it as an assistance to Wi-Fi localization, including Walkie-Markie[1], Moloc[6] etc. They will record IMU readings and estimate step count, step length, heading direction and finally try to recover the walking trajectory. However, due to accumulative error of accelerometer and gyroscope, the recovered trajectory could suffer from severe distortion, which causes much trouble for building fingerprint database. Besides, these methods require temporal information from INS and this information may not always be available. The proposed method in this work does not require users to collect continuous sequence of data. Instead, users can record data any time at any location, no matter the data is collected from continuous walk or from just random points in the site. Therefore it is more robust to

users' motion status.

Crowdsourced Wi-Fi with Wi-Fi signal propagation model Signal propagation model is a ideal estimation of signal strength with respect to distance from measurement to the signal source. One popular work is EZ [7], which use RSSI to constrain the relative position between signal point and APs. The general idea of this work is quite brilliant, since it does not need extra information and could achieve acceptable result, but one drawback is that it has an assumption on the Wi-Fi signal propagation model which is thought to be not reliable in real-world environment. The proposed method gets rid of the propagation model, and only depends on similarity between RSSI values hence is more convincing to use.

Crowdsourced Wi-Fi with prior fingerprints Researchers have been making efforts to reduce the labor of traditional site survey. The technique proposed in [8] utilizes compressive sensing to localize users. It consider the reference points as a sparse matrix and recover the more detailed signal map. But it still requires many fingerprints to work well. Another trending field is manifold alignment [4, 9, 10]. It first creates two manifolds in both the signal space and the physical space, and a transfer learning algorithm will match the local geometry from one manifold to another. By this learning, only a small portion of fingerprints are required to achieve similar performance as other works. But one drawback is that its parameters need tuning on different environments, and this limit the ability for scalability. The proposed method in this work also uses a small portion of fingerprints but makes an effort to improve the robustness of the parameters so that it can be extended into different environments.

2.3.2 Network embedding

Network embedding has stand out as a intuitive way to capture the distance and relationship between nodes. It represents nodes as a low-dimensional vector, and has a wide range of applications based on the representation vector, including link prediction and node clustering, as mentioned in [11]. For weighted undirected graph, there are some good unsupervised model like LINE [12], which preserve both first order and second order proximity between vertices in a graph. In this work, we will do some modifications to LINE and adapt to our cases for indoor localization.

Graph neural networks use deep learning method to solve multiple graph-related

problems, and it can also be used for network embedding problems. The model could be unsupervised, semi-supervised and supervised. [13] has conducted a comprehensive review on popular graph neural networks. In this work, we will also choose some models including graph auto encoder (GAE) [14] as the comparison schemes to learn the graph structure.

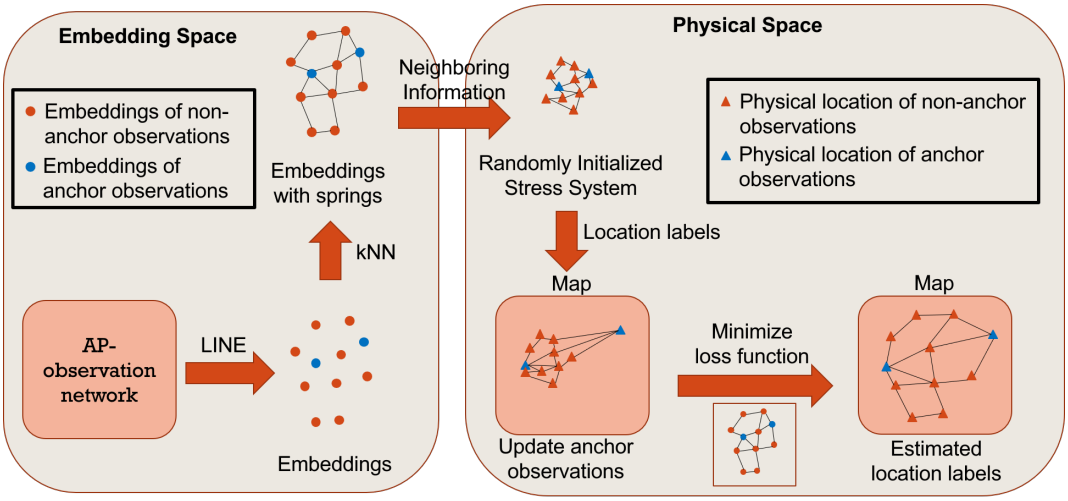


Figure 2: Flow diagram of the whole system.

3 Methodology

3.1 Design

This section will describe the formulation of the problem and briefly discuss the proposed method to the problem. To begin with, the formulation of this work was defined. Then the whole work will be divided into subtasks, namely building AP-observation network, network embedding and map matching system, and database construction. The whole workflow is shown in Fig.2

3.1.1 Dataset construction and problem formulation

The focus of this work is to use a small portion of fingerprints and crowdsourced Wi-Fi data to construct a fingerprint database. Before introducing the problem formula, some definitions and notations have to be made for clarity.

Definition Wi-Fi data are taken at random points in the indoor environment. These points are called *observations*. Each observation has two attributes: measurement and location label. The *measurement* of an observation is a collection of all detected AP's MAC address and corresponding RSSI values taken at this point. The coordinate of this observation is termed as its *location label*.

An illustration of collected Wi-Fi data are shown in Fig.3a. There are totally n observations and m APs in the experiment environment. The subscript of the RSSI is

	AP_1	AP_2	AP_3	...	AP_m
$O_1(O_{A\_1})$	$RSSI_{11}$	$RSSI_{12}$	$RSSI_{13}$	...	-
$O_2(O_{A\_2})$	$RSSI_{21}$	-	$RSSI_{23}$	...	$RSSI_{2m}$
...	...	...	...	...	...
$O_k(O_{A\_k})$	-	-	$RSSI_{k3}$	...	-
$O_{k+1}(O_{N\_1})$	-	-	-	...	$RSSI_{(k+1)m}$
...	...	...	...	...	...
$O_n(O_{N\_(n-k)})$	-	$RSSI_{n2}$	-	...	$RSSI_{nm}$

(a) Wi-Fi dataset

Observation	Location label
$O_1(O_{A\_1})$	$<x_1, y_1>$
$O_2(O_{A\_2})$	$<x_2, y_2>$
...	...
$O_k(O_{A\_k})$	$<x_k, y_k>$
$O_{k+1}(O_{N\_1})$	unknown
...	...
$O_n(O_{N\_(n-k)})$	unknown

(b) Location label set

Figure 3: Illustration of datasets. "O" stands for observations, "AP" for access points and "RSSI" for received signal strength indicator. "-" means that the AP is not detected hence no RSSI record.

defined as:

$$RSSI_{ij} = \text{RSSI value of } AP_j \text{ measured at } O_i.$$

For example, the measurement for O_2 in Fig.3a is a collection of APs' MAC addresses and RSSI values:

$$O_2^M = \{MAC_1 : RSSI_{21}, MAC_3 : RSSI_{23}, \cdots, MAC_m : RSSI_{2m}\}.$$

Note that since AP_2 is not detected by O_2, the entry for AP_2 is not included in O_2's measurement.

The location label of O_2 is simply its coordinate in the indoor environment:

$$O_2^L = \{x_2, y_2\}$$

Definition A *non-anchor observation* is an observation whose location label is unknown, while an *anchor observation* is an observation with known location label.

Observations are categorized into two sets based on the presence of location labels. $\mathcal{A}$ is the set of indices for anchor observations, and $\mathcal{N}$ is for non-anchor observations. Denote the number of anchor observations as k, then there are $n-k$ non-anchor observations in the dataset. In this work, there are only a small portion of anchor observations, so $k \ll n$.

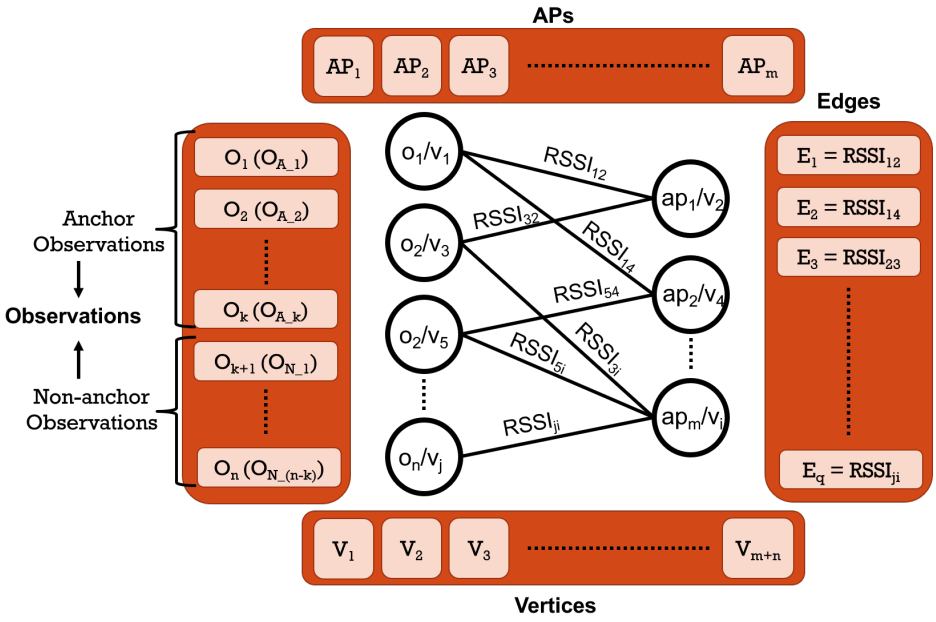


Figure 4: Illustration of AP-observation network. Four red boxes surrounding the network give explanations to notations and indices in the network.

Fig.3b shows the example of location label set. Since the first k entries are anchor observations, they have the location labels already, and the remaining don't have yet. Our problem is to use the Wi-Fi dataset and this location label set to estimate the labels for non-anchor observations. Once we have the labels, a Wi-Fi fingerprint database can be established for indoor localization.

Definition A *fingerprint database* is a collection of observations with either actual location labels (anchor observations' case) or estimated ones (non-anchor observations' case).

The above definitions give fundamental concepts on which the whole problem are built. The problem statement is therefore made as follows:

Problem statement Given a set of n observations $\{O_1, O_2, \cdots, O_n\}$ with their measurements $\{O_1^M, O_2^M, \cdots, O_n^M\}$, among which $k (k \ll n)$ of them are anchor observations with location labels $\{O_1^L, O_2^L, \cdots, O_k^L\}$, estimate the location labels for non-anchor observations $\{\hat{O}_{k+1}^L, \hat{O}_{k+2}^L, \cdots, \hat{O}_n^L\}$.

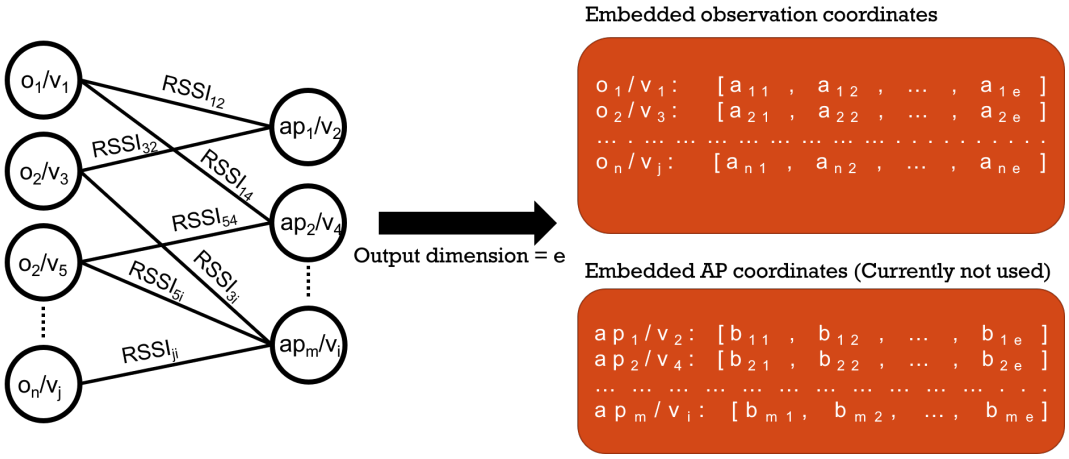


Figure 5: Illustration of LINE. o, ap, v are the notations for observations, APs and vertices in the network.

3.1.2 Building an AP-observation network

The first step after we get the crowdsourced Wi-Fi data is to construct the AP-observation network, which is an intuitive simulation of physical environment. In this network, both APs and observations become vertices of the network, and once there exists $RSSI_{ij}$, there is an edge from O_i to AP_j, with the weight of $RSSI_{ij} + c$, where c is an offset to make all weights nonnegative. Fig.4 shows the structure of the AP-observation network.

The reasons for using AP-observation network instead of a matrix to store the WiFi signals are twofold: first, this network solves the problem of missing RSSI values, since we do not need to fill up missing values as what we do for the matrix. Second, this network is easy to extend when new observations come in. Matrix form of WiFi data encounters the problem of extending vector lengths when new APs are introduced by incoming observations. AP-observation network can simply solve this problem by adding new observation nodes and AP nodes. It is also convenient to remove some observations or APs if they are useless at some stages.

3.1.3 Network embedding

The generated AP-observation network will be fed into the network embedding algorithm to learn lower-dimensional representations of both observations and APs. It studies the local structure of the AP-observation network based on two types of proximity, then preserve these local structures in the embedded space. There is no

location labels required in the network embedding, so this step is unsupervised. To make the algorithm work best, the output dimension of vertices are usually higher than 2-D and cannot be directly used as location labels.

There are many network embedding algorithms already in the publications, and they focus on different applications, including link prediction and node classification, etc. In this work we chose LINE [12], because it can better capture the local proximity of the network and is able to handle large datasets, which enables the scalability of this work. In addition, its running complexity is quite low compared to state-of-the-art works.

LINE mainly deals with local proximity relation between vertices in the network. That means if two observations have more similar measurements than others, they should be closer in the embedded space. An illustration of LINE is shown in Fig.5. Denote vertices as v_i, where i is the index of vertex. There are two types of proximity in LINE, first order and second order proximity. First order proximity defines the joint probability of v_i and v_j as:

$$p_1(v_i, v_j) = \frac{1}{1 + \exp(-\vec{u}_i^T \cdot \vec{u}_j)},$$

where $\vec{u}_i, \vec{u}_j$ are the embeddings for v_i, v_j. Its empirical probability is defined as:

$$\hat{p}_1(v_i, v_j) = \frac{w_{ij}}{\sum_{(i,j)\in E} w_{ij}},$$

where w_{ij} is the weight of edge $(i,\ j)$, and E is the set of all edges in the network. KL-divergence is used as the objective function:

$$O_1 = KL(\hat{p}_1 || p_1) = -\sum_{(i,j)\in E} w_{ij} \log p_1(v_i, v_j).$$

Second order proximity considers neighbor relationship and tries to preserve transition probability. To be more specific, it first defines the probability of "context" v_j given vertex v_i as:

$$p_2(v_j|v_i) = \frac{\exp(\vec{u}_j'^T \cdot \vec{u}_i)}{\sum_{k=1}^{|V|} \exp(\vec{u}_k'^T \cdot \vec{u}_i)},$$

where $\vec{u}_j', \vec{u}_k'$ are the embedding for v_j and v_k when they are treated as "context". Then the empirical probability of "context" is defined as:

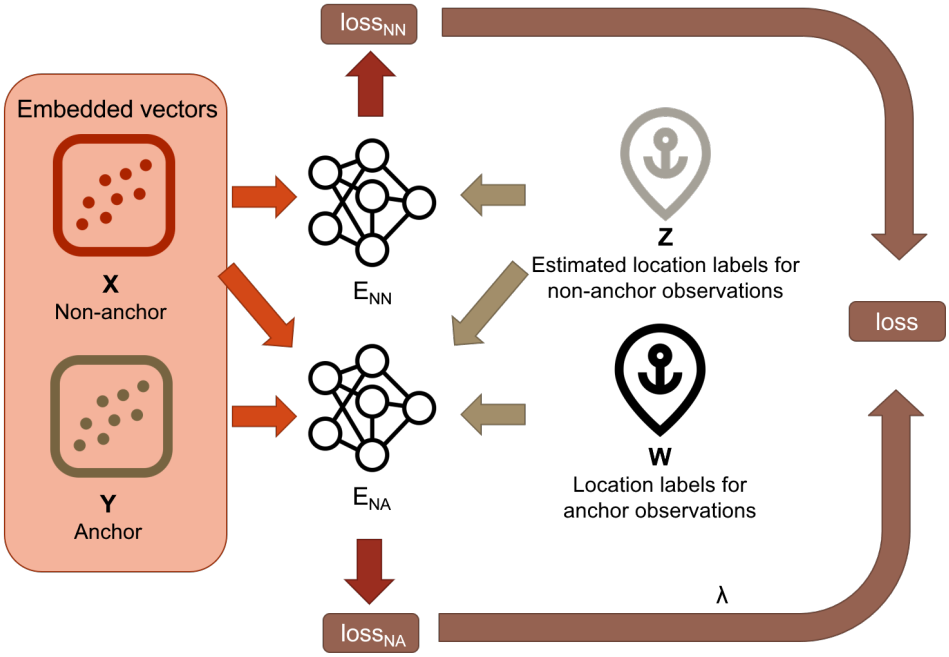


Figure 6: Illustration of map matching algorithm.

$$\hat{p_2}(v_j|v_i) = \frac{w_{ij}}{\sum_{k \in N(v_i)} w_{ik}},$$

.

where w_{ij} is the weight of edge (i, j), and $N(v_i)$ is the set of all neighbors of v_i. Dissimilarity of the two probabilities is measured by KL-divergence, which serves as the objective function that should be minimized:

$$O_2 = KL(\hat{p_2}||p_2) = -\sum_{(i,j) \in E} w_{ij} \log p_2(v_j|v_i).$$

By using negative sampling technique, LINE can learn the embeddings for large datasets very fast. After processing the AP-observation network, LINE will generate embedded coordinates for observations and APs. Currently APs' embeddings are not included in this work, so only embeddings for observations will be kept for map matching.

3.1.4 Map matching system

The map matching system takes the embeddings of observations and location labels of anchor observations, and estimates the location labels for non-anchor observations.

The idea is to use nearest neighbors as local constraints and use location labels as global constraints. The flow diagram of this system in shown in Fig.6.

To begin with, the embeddings will be divided into non-anchor type and anchor type. Then we have two sets, X for embeddings of non-anchor observations and Y for anchor ones. We also have the set of location labels for anchor observations W and Z for the set of estimated location labels of non-anchor observations.

In embedding space, to capture the local structure, k-nearest-neighbor (kNN) will be used on the embeddings both within non-anchor observations and between non-anchor and non-anchor observations to extract neighbor information. For each embedded vector, the algorithm finds the nearest k embedded vectors, and the edges between the vector and its nearest neighbors will be selected as a neighbor information. Denote the chosen vertex pairs as E_{NN} and E_{NA} correspondingly.

Then in physical space, neighbor sets E_{NN} and E_{NA} are used to build a new network in which each vertex represents the estimated location label of the observation. Each observation corresponds to an embedded vector in the embedding space and a location label in the physical space. Therefore, the network in the physical space contains the estimated location labels for observations. They are first initialized with random location. Then we set up our objective function based on the constraints between non-anchor observation and anchor observations. The objective function consists of three parts, namely the loss within non-anchor observations $loss_{NN}$, and the loss between non-anchor observations and anchor observations $loss_{NA}$:

$$loss_{NN} = \sum_{(i,j)\in E_{NN}} (\|\vec{x}_i - \vec{x}_j\| - \|\vec{z}_i - \vec{z}_j\|)^2,$$

$$loss_{NA} = \sum_{(i,k)\in E_{NA}} (\|\vec{x}_i - \vec{y}_k\| - \|\vec{z}_i - \vec{w}_k\|)^2.$$

The reason for not having loss within anchor observations is that both their embedded vectors Y and their physical locations W are fixed, so the loss cannot be further optimized.

So the objective function is:

$$\min_X \quad loss_{NN} + \mu \cdot loss_{NA},$$

where μ is the weight parameter that controls the focus of optimization. The higher

the μ, the closer the non-anchor observations will approach neighboring anchor observations, and the more the non-anchor observations will push away from each other.

Then the derivatives for two loss functions are calculated as:

$$\frac{\partial loss_{NN}}{\partial \vec{x}_i} = 2 \sum_{\vec{x}_j \in knn(\vec{x}_i, X)} \frac{(\|\vec{z}_i - \vec{z}_j\| - \|\vec{x}_i - \vec{x}_j\|)}{\|\vec{z}_i - \vec{z}_j\|} \cdot (\vec{z}_i - \vec{z}_j),$$

$$\frac{\partial loss_{NA}}{\partial \vec{x}_i} = 2 \sum_{\vec{w}_k \in knn(\vec{x}_i, Y)} \frac{(\|\vec{z}_i - \vec{w}_k\| - \|\vec{x}_i - \vec{y}_k\|)}{\|\vec{z}_i - \vec{w}_k\|} \cdot (\vec{z}_i - \vec{w}_k).$$

The loss function will be trained by Adam optimizer for fast convergence. After training, the set of estimated location labels Z is obtained. Measurements and location labels for all observations are then passed to the construction of fingerprint database.

3.1.5 Fingerprint database construction

Once we get the estimated location labels for non-anchor observations, we can combine them to be fingerprints. Since crowdsourced data are taken at random points and some of them may be very close to each other, a fingerprint simplification is used to eliminate unnecessary fingerprints which are too close. Specifically, when we have the location label set, the algorithm will choose a random point and remove other points within a given distance threshold. Then this process will continue until all the points in location label set are either chosen or removed. The chosen points will form the final fingerprints. After fingerprint simplification, the remaining will be stored as fingerprint database, and this can be used for practical localization services.

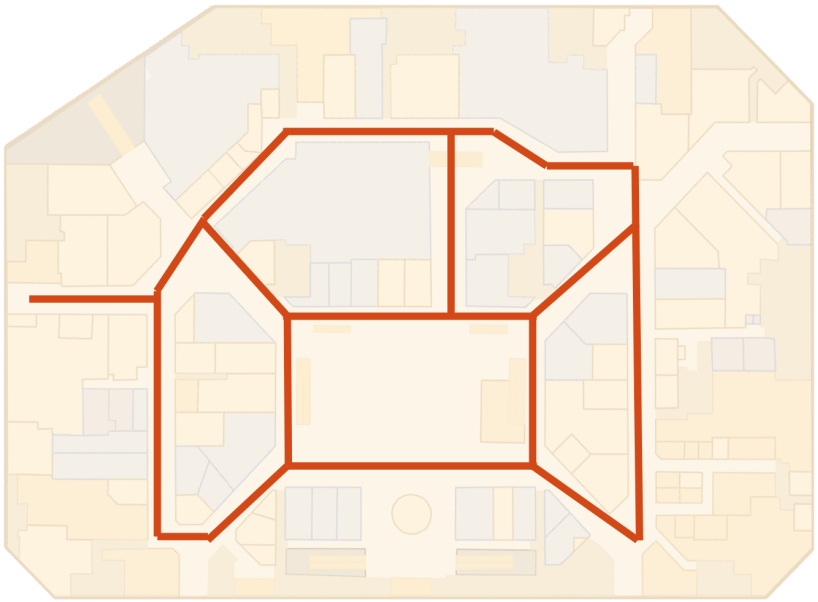


Figure 7: Hanghau shopping mall

site	Hanghau shopping mall
area	$180 \times 132 = 23760\ m^2$
training data	1510
testing data	1560
turning points	17

Table 1: Information about the experiment sites

3.2 Implementation

Based on the design section above, we have the following implementations in this work:

3.2.1 Collection of the dataset

We collected crowdsourced Wi-Fi data in various indoor environments including the school campus and the shopping mall. They represent two typical indoor conditions: in school campus there are many narrow corridors, whereas in shopping malls the corridors are much wider and there are also open squares. The maps of them are shown in Fig. 7.

The data are collected through a set of pre-defined paths so that the actual location labels of WiFi signals can be easily recorded. All the collected data have their location

labels, and they are only used for validation. Table 1 shows the size of the dataset and the area of the experiment sites.

3.2.2 Setup the running environment on a Linux server

All the codes are written in Python. Because the program is large and requires matrix computations, it is good to run it on a server. Therefore, we setup the environment on the server in the lab. Anaconda is an integrated software for managing Python packages conveniently, so we install it on the server. To write code directly on the server, Jupyter notebook and vim is carefully configured. Some other common packages are also installed, including scikit-learn, numpy, matplotlib, tensorflow, etc.

3.2.3 Implement the data loader program

Data loader is a program that loads the collected data and convert them to an AP-observation network. It will first number both the observations and APs, and then generate an edge list for further use. The data loader also allows incremental update of the edge list, so it is easy to extend the dataset when new WiFi signals are collected.

3.2.4 Implement the anchor selection program

In this project, the anchors are selected based on some criteria. For example, the anchors are chosen uniformly throughout the whole site. Other criteria are also possible, including sampling anchors near critical points (e.g., turning points), or just sampling randomly in the site. For testing, different parameters (e.g., sampling density) can be applied to the criteria so that the number of anchor observations is controllable.

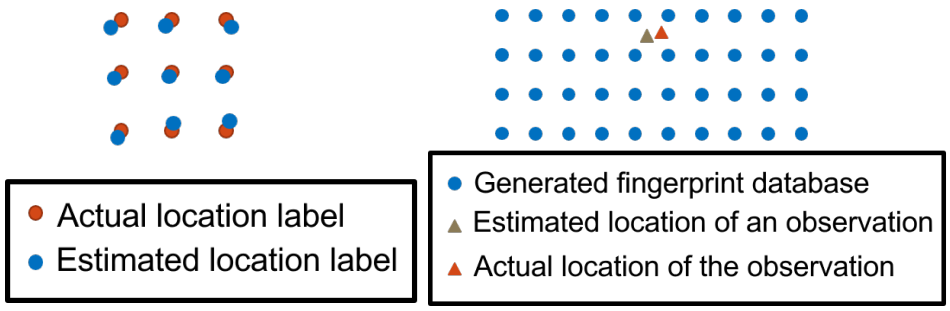
3.2.5 Implement the evaluation codes

To show how our proposed method works, a set of evaluations must be made. So we wrote a series of functions that can analyze the data in various aspects. For a set of data, the evaluation codes can calculate the basic statistics such as median, IQR (interquartile range) and RMSE (root mean square error). It can also shows the CDF (cumulative distribution function) of a given dataset. For two sets of data, it can analyze the correlation between them by both plots and calculation of correlation coefficient. If we want to compare multiple experiments at the same time, a box plot function can handle this situation.

3.2.6 Implementation of the main program codes

Based on the design, the program code is implemented step by step. At the beginning, we wrote a small demo code to validate that our method is usable for the collected data. A lot of software testing procedures were conducted to fix the bugs for the program. When the testing were done, we put them into Python files, and added a module that can automatically conduct a sequence of experiments. In addition, we also implemented a mode selection module so that all the experiments and the testing process can be done by running the same program with different modes. These modules allow us to run the program more conveniently.

Initially, the demo code is written in Jupyter notebook for better interaction. When most of the codes are implemented and tested, they are migrated to a series of Python files. For the existing algorithms such as kNN, we directly use the API provided by scikit-learn. After trying on several existing network embedding algorithms, we finally adopt the opensource repository on GitHub to conduct the LINE algorithm. As for the map matching part, we uses Tensorflow to build up the model and defines the objective function.



(a) Database error (b) Positioning error

Figure 8: Illustration of different errors. Database error shows the ability to correctly construct the fingerprint database, while positioning error shows the ability to correctly locate a person.

3.3 Testing

The performance of the proposed work should be comprehensively examined, including the accuracy of database reconstruction, location error, complexity, and other factors that may affect them.

3.3.1 Testing the accuracy of database reconstruction

Database error measures the ability for the algorithm to correctly estimate the location labels for non-anchor observations. It is defined as the distance between actual location labels and estimated ones. The concept is shown in Fig.8a. Specifically, denote Z^e as the set of all estimated location labels, and Z^t as the set of all true location labels, then the database error for one fingerprint is:

$$e_i^{db} = \left\| z_i^e - z_i^t \right\|_2 .$$

Since most of the location labels will have small error and only a small number of location labels may have large error, we use the median and IQR(interquartile range) of all the error to evaluate the performance instead of using mean and standard deviation. Besides, RMSE (root mean square error) will also be used for evaluating the error.

3.3.2 Testing the accuracy of locating users

Positioning error measures the ability for the algorithm to correctly estimate the location of a user given one measurement. It is defined as the distance between actual

position of incoming observations and estimated ones using fingerprint database. The concept is shown in Fig.8b. Specifically, denote Z^L as the set of all location labels of observations, Z^M as the set of all WiFi measurements for observations, Q^M as the set of all query WiFi signals, and Q^L as the set of location labels of all query WiFi signals, then the database error for one fingerprint is:

$$e_i^{pos} = \left\| loc\_algo(q_i^M, Z^M, Z^L) - q_i^L \right\|_2,$$

where loc_algo is a localization algorithm that can use WiFi signals and fingerprints to estimate the location labels for incoming queries. Because positioning error is similar to database error, we can still evaluate it using median, IQR and RMSE of the error of all queries.

3.3.3 Testing the positioning error via different localization algorithms

When the fingerprints are generated from the algorithm, there are several ways to use them for location estimation. One simple approach is knn. When a new query WiFi signal $\vec{q}^M$ comes in, it will be compared to all fingerprints in Z^M and the WiFi cosine similarity will be calculated as follows:

$$similarity(\vec{q}^M, \vec{z}_i^M) = \frac{\vec{q}^M \cdot \vec{z}_i^M}{\|\vec{q}^M\| \cdot \|\vec{z}_i^M\|},$$

where $\vec{q}^M$ and $\vec{z}_i^M$ are RSSI vectors of the same length. To deal with the missing RSSI values, this method usually fill the missing value with a minimum value of -120.

After comparing the query WiFi signal with all fingerprints, the most k similar ones will be used to interpolate the location of the query:

$$knn(q^M, Z^M, Z^L, k) = \sum_{\vec{z}_i^M \in top_k(similarity(\vec{q}^M, \vec{z}_i^M))} similarity(\vec{q}^M, \vec{z}_i^M) \cdot \vec{z}_i^L.$$

Besides, our proposed method can also be used as an offline localization algorithm. To estimate the location of a query WiFi signal, the algorithm will add it to the existing AP-observation network, and continue to use network embedding to train the network. After that, we obtain its embedded vector, and establish the neighboring information with fingerprints' embedded vectors. Then the map matching model will update its physical location based on the objective function. When the map matching system

finishes, the learned location label will be the estimated location of the query WiFi signal.

However, since this process is much more complex than knn method and its running time is also much higher, our proposed algorithm is only used as a comparison to the knn method here. But for fingerprinting process where real-time estimation is not required, our proposed algorithm is capable to handle.

3.3.4 Testing the relationship between database error and positioning error

There is a strong connection between database error and positioning error. Experiments will be conducted to reveal how they relate to each other. It is expected that the database error is positively related to the positioning error. A scatter plot will be generated to show the general relationship between two types of error, and the spearman coefficient, a measure of correlation between two variables, will be calculated to quantify this relationship. The spearman coefficient is defined as the Pearson correlation coefficient between the ranked variables. For two sets of data X, Y, their values are first converted into the rank set X', Y', and the Pearson correlation will be calculated:

$$\rho_{X,Y}^{spearman} = \rho_{X',Y'}^{pearson} = \frac{\sum_{i=1}^{N}(x_i' - \bar{x}')(y_i' - \bar{y}')}{\sqrt{\sum_{i=1}^{N}(x_i' - \bar{x}')^2 \sum_{i=1}^{N}(y_i' - \bar{y}')^2}}$$

3.3.5 Testing the relationship between anchor observations and database error

Anchor observations play an essential part of the proposed algorithm. To check the generality and robustness of the algorithm, anchor observations will be generated in different settings in terms of number of anchor observations, choice of anchor observations, etc. To be specific, the following cases will be considered:

- Choose anchors uniformly in the environment, with a predefined sampling density;
- Choose anchors randomly in the environment, with a predefined sampling density;
- Choose anchors that are close to critical landmarks, e.g., turning points.

3.3.6 Testing the relationship between parameter choices and database error

There are several hyperparameters in both LINE and the map matching system. Experiments will test different settings of the parameters and obtain the database error as metrics. If the algorithm works equally well with several sets of parameter values, then it achieves a good robustness in different conditions. On the other hand, if the algorithm is sensitive to the hyperparameters, then we can find the optimal values for each of them.

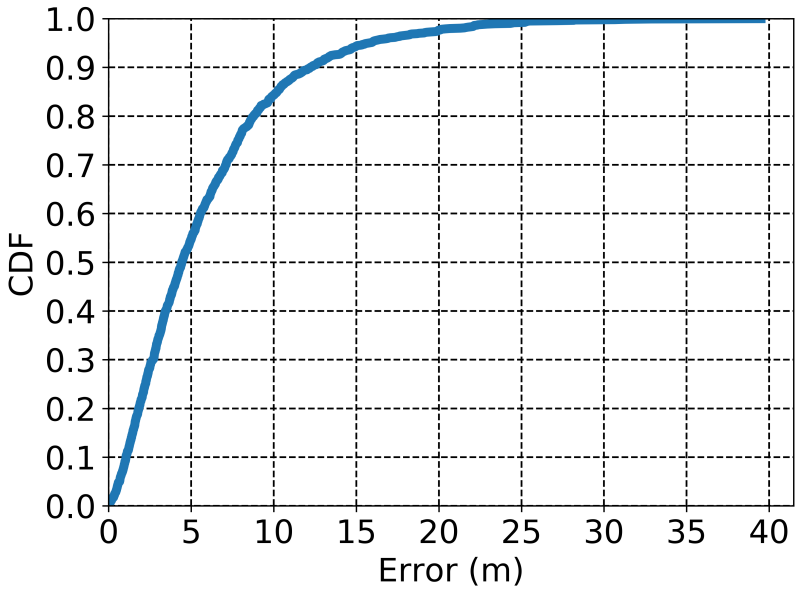


Figure 9: A CDF for the database error

Name	Method
RFK	RP + FP + kNN
RK	RP + kNN
RFP	RP + FP + proposed method

Table 2: Notation and explanation of different localization methods. They can all be used to calculate positioning error

3.4 Evaluation

To begin with the evaluation, some notations should be carefully defined. First of all, the term "RP" or "Reference Point" is used to denote the initial fingerprints, and "FP" or "fingerprint" is the generated one. Then based on the standard proposed in Section 3.3, we evaluate the performance of the method by the following results:

3.4.1 Database error

Database error can be measured in different aspects, such as median, iqr, rmse, and CDF. We take an experiment from Hanghau shopping mall as an example to show the database error. According to Table 3, the median, iqr and rmse of database error is 4.44m, 5.59m and 7.69m respectively. Fig. 9 shows the CDF for this experiment. From

type	median (m)	iqr(m)	rmse(m)
Database	4.44	5.59	7.69
RFK	23.26	37.27	41.41
RK	27.12	34.92	44.29
RFP	12.53	24.39	25.67

Table 3: Statistics of an experiment in Hanghau shopping mall. "Database" stands for database error, "RFK", "RK" and "RFP" stand for the positioning error using localization algorithms in Table.2 respectively.

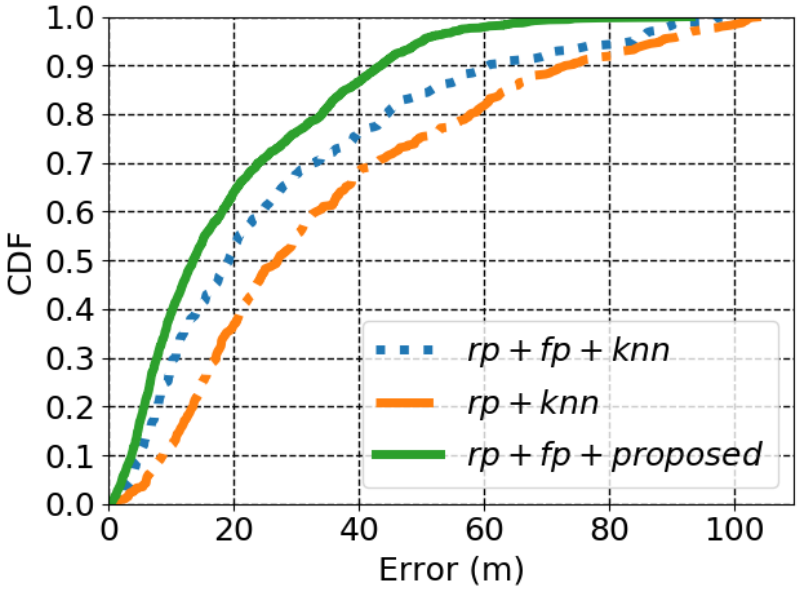


Figure 10: A CDF for the positioning error using different localization algorithms

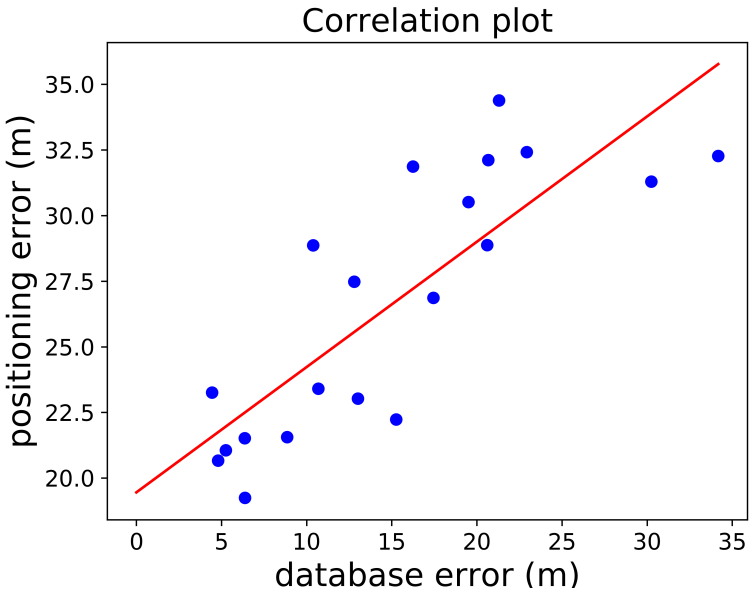


Figure 11: A scatter plot showing the relationship between positioning error and database error

the plot, it is easy to find that the location estimations for more than half of the WiFi signals have the error less than 5m. And over 90% of them have the error within 15m. There are still a small portion of data who has extremely high error. This may result from the map matching part where distant points are mistaken as close.

3.4.2 Positioning error

Positioning error is an approximation of how well the localization algorithm can perform using either reference points or fingerprints or both. As proposed in the testing section, we design three localization algorithms, and their names and methods are shown in 2. For RFK, both reference points and fingerprints are used as a database, then the location estimation can be performed by kNN. In RK, fingerprints are removed from the database, and the rest is the same as RFK. As for RFP, the kNN module is replaced by our proposed localization method as discussed in the testing part.

When we do not have fingerprints, we can only use RK for localization. But after we generate fingerprints from our proposed method, we can use it to improve the location accuracy. So RFK and RFP are proposed as a better way to estimate locations. Table 3 shows the positioning error for all three methods. As shown in this table, RK performs the worst, with a median error of 27.12m. By introducing the generated fingerprints,

RFK achieves a smaller error of 23.26m. However, the iqr of RFK is 2m larger than RK. This shows that although RFK has a smaller overall error, but its variation is higher than RK. RFP has a significant improvement in all statistics than RK and RFK, with a median error of 12.53m and iqr error of 24.39m. This result is reasonable because by using our proposed localization method, the missing RSSI value problem will be avoided. Besides, the map matching system provides higher constraints than simply considering k most similar WiFi signals in kNN. Fig.10 shows the CDF of positioning error for all three localization algorithms. It can also visually show that RK has the worst performance, RFK has a better performance and RFP is the best among the three. With RFP, over 60% of points have the error of less than 20m, but for RFK and RK only 50% and 40% of points can achieve this level. Notice that there is still a long tail in RFP, and this may also be the problem of misconnecting distant points in the map matching.

3.4.3 Relationship between database error and positioning error

Database error and positioning error are quite similar, since the positioning relies on the database and a more accurate database can perform better in positioning. So we study the relationship between the database error and the positioning error using RFK. Fig.11 shows the scatter plot of two types of error. Each blue point consists of the median of both database error and positioning error in one experiment. The red line is the linear regression of all the blue points. As shown in the graph, the blue points generally distribute in a linear model, meaning that the less the database error, the less the positioning error. Spearman coefficient also support this interpretation, since the two types of error have a correlation of 85.7%, showing that they have a high correlation.

There is another issue in this graph. The slope of the line is far from 1, meaning that database error and positioning error are not equal in general. The reason for this may be related to kNN, since it conducts the weighted average location of k most similar ones. If we reduce the number of k until $k = 1$, then the slope may be close to 1.

3.4.4 Relationship between reference point density and database error

Reference points play an critical part in the whole method, so it is of great value to study how the number of reference points affect the database error. We sample

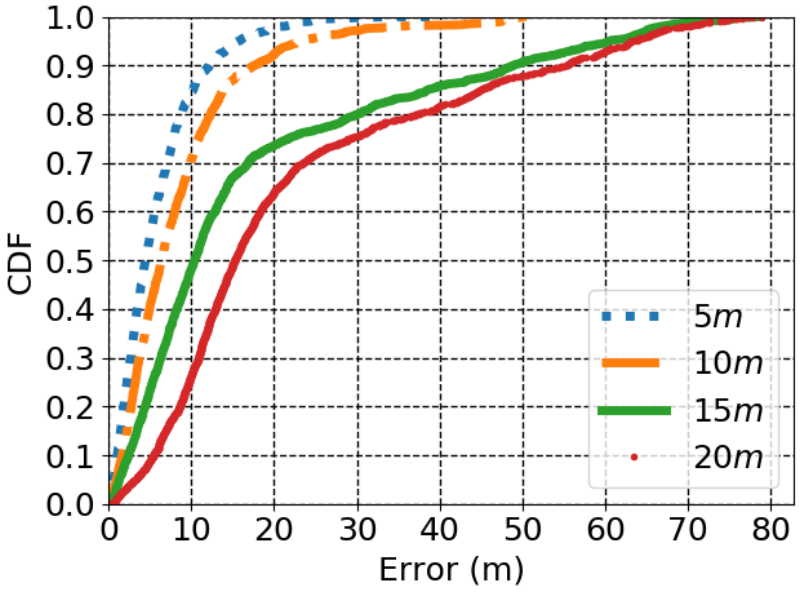


Figure 12: A CDF of database error for several reference point sample frequencies.

the reference points in different densities, i.e., sampling them in 5m, 10m, 15m, and 20m. Then we run the algorithm and test the database error. The results are shown in Fig.12. As is clearly shown, the database error increases as the reference point density is reduced. Besides, the maximum error also increases when the reference point density is reduced. When the sample density is 15m, the maximum error can even reach 80m. This shows that our proposed method is sensitive to the number of reference points. In terms of the level of location accuracy, approximately 85% of the points' error are within 10m when the sample density is 5m, and this ratio becomes 70% with the density of 10m, 50% with the density 15m, and 25% with the density 20m. However, if we measure the performance by looking at how many points have error less than the sample density, then it is surprising to find that nearly 65% of the points reach this goal regardless of sample densities. This means that our method can make sure overall more than half of the WiFi signals are labeled with satisfying location labels. It is also illustrated in Fig.13 that most of the data have acceptable database error, with a low median error and a low upper quantile error. However, there are also some outliers whose database error is far larger than the sample density, and the error of some points can reach 80m. For those whose error is much larger than the sample density, they may suffer from incorrect neighboring information generated in the map matching system.

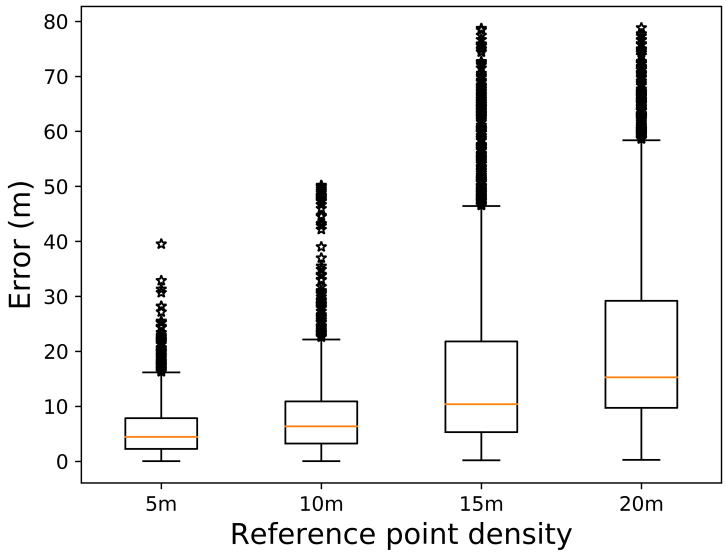


Figure 13: A box plot of database error for several reference point sample frequencies.

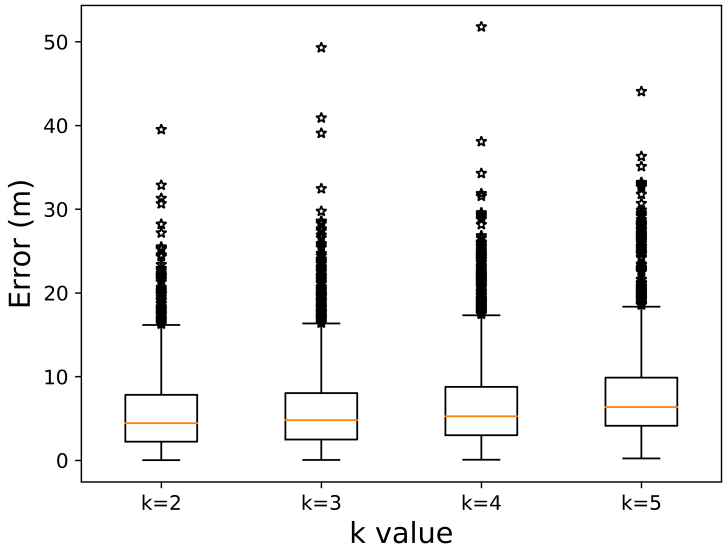


Figure 14: A box plot of database error for several k values

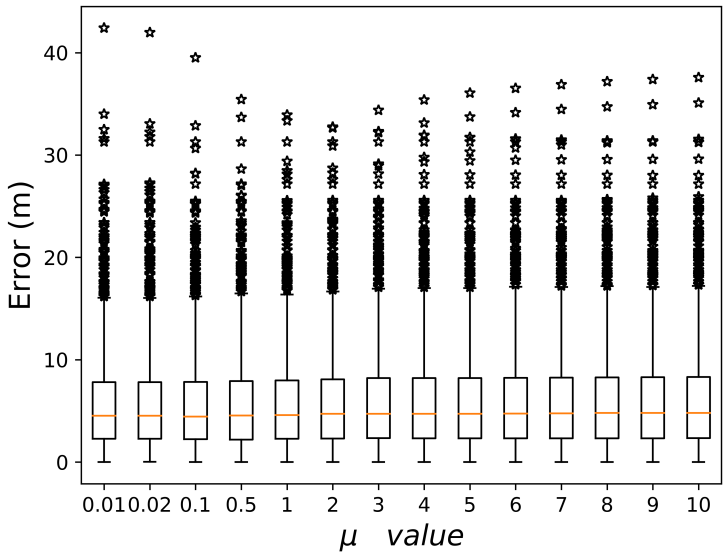


Figure 15: A box plot of database error for different μ choices

3.4.5 Relationship between k values and database error

Before the map matching system starts, neighboring information is captured by kNN. The larger the k value, the more neighbor will be connected to each point.If the relative distance between embedded vectors are the same as that between physical locations, then more neighboring information will definitely improve the accuracy and reduce the database error. However, since LINE can only give a rough local structure of observations, the value of k may be restricted to a small value. Fig.14 shows the effect of different k to the database error. As shown in the box plot, when k increases, the median of database error also increases, meaning that additional neighbors will be harmful to the performance.

3.4.6 Relationship between the weight parameter and database error

The weight parameter μ controls the level of constraints made by reference points. A larger μ will drag other points closer, but the constraints between non-anchor observations will be reduced. So there is a balance between the two types of constraints. Fig.15 shows how μ affects the database error. In terms of median and iqr error, the algorithm is not sensitive to μ's value, as they are almost of the same value. But its value affects the maximum error. When μ starts from 0.01 to 2, the maximum error drops

down for around 10m, and when it continues to increase from 2 to 10, the maximum error also increase. This implies that although μ does not help to reduce the median database error, it can be optimized to reduce the maximum error.

ID	AP1	AP2	AP3	AP4	AP5	AP6	AP7	AP8
1	-80	-90	-80	-	-	-	-	-
2	-	-	-	-	-	-70	-90	-80
3	-	-	-	-60	-60	-80	-	-100

Figure 16: An example of three RSSI vectors

4 Discussion

In this section, we will discuss the key issues of each part of the algorithm, and bring some outlook to the future work.

4.1 Missing RSSI value problem

Traditionally, the similarity between WiFi signals is measured by Euclidean distance between the vectors, or by cosine of the intersection angle of them. Both methods have to deal with the missing RSSI value problem. Take Fig.16 as an example. Intuitively, since the second and the third vector have overlapping APs, they are considered as close, while the second vector is further from the first one because there is no overlap of APs. But when we use Euclidean distance to measure their similarities, we have to fill in the missing RSSI values with a predefined constant, usually smaller than the minimum RSSI value among the vectors. In this example, this constant can be -120, -100 or -90, etc. However, for each of these values, the calculated Euclidean distance between the first and the second vector is smaller than that between the third and the second vector, which indicates a counterintuitive result that the second vector is more similar to the first one than the third one.

The AP-observation network can solve the missing RSSI value problem because it does not need to fill in the missing RSSI values. In this network, the measured RSSI values are converted to the edge between the observation and the AP. If one observation does not consist one AP's RSSI value, then there is no edge between the observation and that AP. But this network also has a drawback that the RSSI values are converted into non-negative edge weights by adding an predefined offset. This conversion is required by LINE to correctly capture the closeness between observations and APs. The effect of using different offsets are unknown up to now. Further work should either find a better way to get the offset or find a better network embedding model that can allow negative edge weights.

4.2 The hyperparameters in LINE

There are two main hyperparameters that affects the output of LINE, namely the order of proximity and the output dimension. The first one determines which proximity to use, and the second one determines how much latent information can be preserved. If the output dimension is low, then some hidden information of the AP-observation network may be lost; but if the output dimension is high, then it is harder to train and the computation will increase significantly.

Another problem is that we did not have a good metrics to evaluate how good LINE performs. For now the evaluation focus only on the database error and positioning error, which is the final result of the whole algorithm instead of an intermediate result from LINE. Further work should set up a evaluation method for the performance of LINE.

4.3 The hyperparameters in the map matching system

Currently the map matching system extracts the neighboring information of embedded vectors using kNN, which means that each embedded vector needs to connect the same number of vectors. However, in the crowdsensed WiFi signals, their locations may not be uniformly distributed in the site, and this leads to an uneven number of neighbors of each embedded vector. Therefore, further work should solve this problem by automatically choosing the number of nearest neighbors.

As for the weight parameter μ in the loss function, its value is also predefined beforehand. This value may vary in different sites. So it is necessary to study the relationship between μ and the dataset, and derive a formula for determining its value in different experiment settings.

4.4 The choice of reference points

For now the reference points (anchor observations) are chosen uniformly in the environment. This choice may not be optimal because it does not consider the special conditions in the environment. For example, the turning points can be treated as an important landmark that connects different corridors. If reference points are only sampled from these landmarks, then the number of them can be reduced without much loss of accuracy. Besides, it is easier to record the locations of these landmarks, and it can reduce the workload of collecting reference points. Further work should be finding these special locations in a given environment.

4.5 The number of reference points (anchor observations)

Reference points can provide a lot of physical information that can restrict the other points, but using too many of them is still time-consuming and labor-intensive. In Hanghau shopping mall, if the reference points are sampled per 5m, then there will be around 70 reference points, which is 4% of the training data. A better approach is to use less reference points and achieve the same performance, which is left for the future work.

4.6 Relative distance or absolute location

As mentioned above, the collection of reference points is inconvenient unless the number of reference points can be reduced to a small value. Besides, collecting reference points requires a map as prior knowledge, since it shows the absolute location of them in a map coordinate system. But in reality, the relative distance between two points are easier to get. PDR (pedestrian dead rocking) is a movement estimation method that can recover the user path based on IMU sensors, and it has been well studied in the past few years and can estimate the relative position with a moderate error. If this information is utilized, then the number of reference points can be further reduced.

4.7 Multi-floor and multi-building fingerprinting

The ultimate goal for fingerprinting is to build a complete fingerprint database not only in a single floor but also in multi-floor buildings and multi-building communities. When the fingerprinting can be well performed in the single floor, then the next step is to extend it to neighboring floors. We did some experiments on the separation of floors, and the results showed that it can well separate the floors who are not adjacent and the areas who are not adjacent, but the same regions in neighboring floors are hard to differentiate. Future work on multi-floor fingerprinting should analyze the signal difference in neighboring floors or model the signal transitions from one floor to its adjacent floor.

5 Conclusion

Indoor WiFi fingerprinting is essential and useful for various smart city applications, such as localization services and WiFi heatmap monitoring. Traditional fingerprinting method requires hiring professional surveyors to conduct site survey, which is laborious and time consuming. As an efficient alternative, crowdsensed WiFi fingerprinting methods emerged and their goal is to use additional information to reduce the workload of site survey. Recent works usually utilize IMUs to get user movements as constraints, or model the relationship between RSSI and distance as a constraint. However, they either require temporal information from users or use an inaccurate model. A third way is to use manifold alignment to match signals to the physical locations with sparse reference points. Their performance is excellent, but the model is hard to extend when new data comes in. Besides, it cannot handle the missing RSSI values very well. Our proposed method can solve this by creating an AP-observation network and use network embedding to generate embedded vectors for all WiFi signals. A map matching system then use sparse location labels and the neighboring information among the embedded vectors to match them to the physical space. Extensive experiments are conducted in one shopping mall, and various evaluation methods are applied to show the performance of the algorithm in different aspects. Due to limit of time, there are still some questions that remains unsolved and are left for future works.

6 References

[1] Guobin Shen, Zhuo Chen, Peichao Zhang, Thomas Moscibroda, and Yongguang Zhang. Walkie-Markie: Indoor Pathway Mapping Made Easy. In *NSDI: Symposium on Networked Systems Design and Implementation*, 2013.

[2] Huijie Chen, Fan Li, Xiaojun Hei, and Yu Wang. Crowdx: Enhancing automatic construction of indoor floorplan with opportunistic encounters. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(4):159, 2018.

[3] Sameh Sorour, Yves Lostanlen, Shahrokh Valaee, and Khaqan Majeed. Joint indoor localization and radio map construction with limited deployment load. *IEEE Transactions on Mobile Computing*, 14(5):1031–1043, 2014.

[4] Khaqan Majeed, Sameh Sorour, Tareq Y. Al-Naffouri, and Shahrokh Valaee. Indoor localization and radio map estimation using unsupervised manifold alignment with geometry perturbation. *IEEE Transactions on Mobile Computing*, 15(11):2794–2808, 2016.

[5] Binghao Li, James Salter, Andrew G. Dempster, and Chris Rizos. Indoor positioning techniques based on wireless LAN. *Proceedings of the 1st IEEE International Conference on Wireless Broadband and Ultra Wideband Communications, AusWireless 2006*, pages 130–136, 2006.

[6] Wei Sun, Junliang Liu, Chenshu Wu, Zheng Yang, Xinglin Zhang, and Yunhao Liu. MoLoc: On distinguishing fingerprint twins. In *Proceedings - International Conference on Distributed Computing Systems*, 2013.

[7] Krishna Chintalapudi, Anand Padmanabha Iyer, and Venkata N. Padmanabhan. Indoor localization without the pain. In *Proceedings of the Annual International Conference on Mobile Computing and Networking, MOBICOM*, 2010.

[8] Chen Feng, Wain Sy Anthea Au, Shahrokh Valaee, and Zhenhui Tan. Received-signal-strength-based indoor positioning using compressive sensing. *IEEE Transactions on Mobile Computing*, 2012.

[9] Qiao Zhang, Mu Zhou, Zengshan Tian, and Yanmeng Wang. Indoor localization using semi-supervised manifold alignment with dimension expansion. *Applied Sciences (Switzerland)*, 6(11), 2016.

[10] Mu Zhou, Yunxia Tang, Zengshan Tian, Liangbo Xie, and Wei Nie. Robust Neighborhood Graphing for Semi-Supervised Indoor Localization with Light-Loaded Location Fingerprinting. *IEEE Internet of Things Journal*, 5(5):3378–3387, 2018.

[11] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. A Survey on Network Embedding. *IEEE Transactions on Knowledge and Data Engineering*, 31(5):833–852, may 2019.

[12] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: Large-scale Information Network Embedding. *Proceedings of the 24th International Conference on World Wide Web - WWW '15*, pages 1067–1077, mar 2015.

[13] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A Comprehensive Survey on Graph Neural Networks. pages 1–22, jan 2019.

[14] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

7 Appendix A: Division of work

The whole work can be divided into the following subtasks:

Done(D):

1. Find related works
2. Collect data on several sites
3. Design the algorithm
4. Proposal report
5. Implement the demo program
6. Implement the full program
7. Test the algorithm
8. Determine comparison schemes
9. Implement evaluation code
10. Progress report
11. Final report

Future(F):

1. Optimize proposed method
2. Conduct more experiments
3. Video
4. Thesis defense

8 Appendix B: GANTT Chart

According to the schedule from CSE department and the current progress of this work, the GANTT chart of this FYT is planned as follows (task number is the same as above):

	Aug	Sept	Oct	Nov	Dec	Jan	Feb	Mar	Apr
D1	✓	✓				✓	✓		
D2	✓	✓							
D3	✓	✓	✓	✓			✓	✓	
D4	✓	✓							
D5	✓	✓	✓						
D6				✓	✓	✓	✓	✓	
D7					✓	✓	✓	✓	✓
D8				✓	✓		✓		
D9							✓	✓	✓
D10						✓	✓		
D11								✓	✓

9 Appendix C: Required Hardware & Software

9.1 Hardware

Development and testing environment: a Linux server with at least one Nvidia GPU

Data collection environment: several Android phones with basic Wi-Fi functions

9.2 Software

For development and testing, we will make use of the following softwares:

Software	**Usage**
Python3	Programming language
Tensorflow	Machine learning platform
Jupyter notebook	For creating interactive result and visualization
Android Studio	For developing Android APP that can be used for data collection

For collecting data on smartphone, we need an APP that could conduct Wifi signal collection and pre-conducted site survey.

10 Appendix D: Monthly work summary

10.1 Monthly report of July 2019

Find and read paper related to Wi-Fi indoor localization I searched some papers related to Wi-Fi indoor localization and get a general picture of the field of it.

Set up programming environments on both laptop and server I installed required software and other useful packages on both my laptop and the server, and tested some simple code on them to make sure the installations are successful.

Brainstorm the goal of this work I met professor Chan several times to discuss about my work and get some advice. Between these meetings, I also had a close contact with Steve who is the PHD student of the professor. Steve and I discussed about a few possible directions of my work, and help me shape the roadmap of what I am going to work on.

10.2 Monthly report of August 2019

Design simulation for indoor environment I designed a simulation system which can produce data collection in the virtual environment. The initial purpose of this system is to validate if the proposed work is achievable, but later it was discarded since I got previously collected data from the campus and there is no need to do simulation.

Find and read paper related to network embedding After some discussions with professor Chan and Steve, I decided to try on network embedding method first and see the outcome. I searched for some highly-cited paper about this topic, and have a close look at it.

Search for opensource code from existing works After I read the papers, I looked for code of these works since most of them have opensource code. When I got the code, I tested them using corresponding example datasets and fix some bugs for them.

Preprocess the previous data into the format that can be fed into the code Since the input in the code I found has a specific format, I had to convert collected data

into that format. Besides, to make further operations easier, I designed some classes that contain useful information in subsequent tasks.

Search for algorithms that can visualize high-dimensional data The output of network embedding is a set of high-dimensional points which cannot be displayed directly on the screen. To visualize them, a method called manifold learning is used to reduce the dimension of data. I searched for several manifold learning algorithms and determined which one to use.

10.3 Monthly report of September 2019

Find and read paper related to graph neural network Learning of graph has been developing quickly these years. When graph learning is combined with deep neural network, a new method called graph neural network emerges. I found some related works in this field and check whether they can be used in my work.

Search for clustering algorithm To split the observation points without knowing the true position, clustering method was the solution. I searched for some popular clustering algorithms, analyze the limitations and strengths for them and choose the one which fulfill the requirement of this work.

Implement the basic algorithms that can map the observation into real-world coordinates I borrowed the concept of spring and potential energy from physics and use it to constrain the relative position between points. After I implement this algorithm, I tested it on a small dataset and see its performance.

10.4 Monthly report of October 2019

Test existing network embedding algorithm I found a famous network embedding algorithm, LINE, and its opensource code. I found some small datasets, and tested the performance of LINE using them. Graph neural network was discarded because LINE can run well enough.

Update the design and objective of the FYT Since the design went further, the scope of this work changed. A fusion framework is being considered as the next step

for this project. After the original work was done, other sensors' signals will be fused into the existing algorithm to improve the reliability.

Design a new network embedding algorithm There are still some limitations of LINE, since it cannot work well on the global structure. I started to design a new algorithm that based on LINE and works well in global view.

10.5 Monthly report of November 2019

Improve the network embedding algorithm A new network embedding algorithm was designed to improve the performance of LINE. It is expected to be more accurate in preserving the global structure of the network.

Discard the multi-story classification Since experiments worked bad for floor classification, this part was put aside for now. If time allows, the classification part should resume immediately.

Design the experiments for evaluation Various evaluation criteria were designed to measure the overall performance of the algorithm, including the positioning accuracy, the robustness of the algorithm, etc. The experiments are expected to be taken in December.

10.6 Monthly report of December 2019

Conduct experiments on the program Some experiments were done and evaluation showed that the performance was not acceptable. Therefore, the design was discarded, and a new design was required.

Update the design of network embedding algorithm Since the experiments showed that the design is problematic, it was updated with depth first search technique. More experiments were expected to finish in January.

Search for related works More related works were found and studied. Some inspirations were gained from these works. They are useful for further design.

10.7 Monthly report of January 2020

Optimize the program It was found that the speed of the original algorithm could be significantly slowed down when the data size became larger. As a result, the code was optimized using matrix multiplication instead of loops.

Conduct experiments on the program After the algorithm was optimized, some experiments were taken again to test its performance. The results showed that this algorithm still needs improving, but it is better than before.

10.8 Monthly report of February 2020

Redefine the problem statement After the work over a semester, I reorganize my thoughts as well as the problem formulation of my FYT. Now it's much clearer and more achievable.

Refine the evaluation criteria After the algorithm changed, the evaluation criteria also adapted to the new algorithm. It considers the environment conditions and is expected to give a better description about how good the algorithm is.

Search for the comparison schemes As the core development of the algorithm finished, the process to find some comparison schemes started. It is expected that there should be one or two related works as the comparison schemes.

10.9 Monthly report of March 2020

Implement the comparison methods The selected comparison methods are implemented in python, and they are tested to make sure they perform correctly. Experiment results of a small dataset is tested by the methods.

Implement the evaluation programs The full set of evaluation methods are programmed as a series of functions. To make the evaluation more convenient, another program is written to preprocess the experiment outputs. Then the evaluation codes will directly use the preprocessed data for plotting.

Conduct extensive experiments Since the experiment procedure is almost finalized, a full set of experiments started to run on the server. To speed up the experiments, the program is executed several times with different hyperparameter values.

10.10 Monthly report of April 2020

Conduct full evaluation The experiment results are obtained, and a full set of evaluations are conducted on the results. The evaluations can give the basic statistics of the error, and visualize them using CDF or box plot.

Prepare the final report At the same time, the final report is being drafted with more detailed explanation of the algorithm and the evaluation criteria. Evaluation results are comprehensively analyzed. Because there are still some questions that cannot be solved by the submission deadline, I put them in the discussion section for future works.