TECHNISCHE UNIVERSITÄT CHEMNITZ

A Graph-SLAM Implementation with a Smartphone

Uğur Bolat

Date: 13. April 2018

Supervisors:
Daniel Froß
Steffen Weichold

Faculty of
Electrical Engineering
and
Information Technology

Contents

List of Figures

List of Tables

Abstract

The indoor positioning application with smartphones is a challenging problem because an average commercial smartphone has no specialized hardware solution yet. Therefore, one has to exploit the existing technologies; such as inertial sensors, signal strength measurements or camera. Since many of these technologies are not designed for the positioning purposes, hybrid systems are needed to compensate each other's drawback. One of the straightforward method is to build a radio map, composed of *Received Signal Strength Indications (RSSI)* that can be acquired from Wi-Fi or Bluetooth Low-Energy (BLE), where you create grid-based maps with the unique fingerprints. Downside of the fingerprinting is that it requires system owners to build the radio map. The easiest way to build this map is to store the signal strength measurements by standing at the reference positions. However, this solution does not scale to the big buildings. To make this radio mapping process efficient, we proposed *Graph-based Simultaneous Localization and Mapping (SLAM)* approach in this research paper. With SLAM, one can collect the measurement during walking. On the contrary, the problem gets more complicated as we have to track the user's walking path while mapping. To tackle this problem, we first lay out the general SLAM problem, which is well-known in robotics domain. Then, we transform the problem formulation to smartphone application since we don't have such rich sensing capabilities like robots in smartphones. By using this transformed SLAM algorithm, we compare Wi-Fi, BLE, and Magnetic Field sensors in the context of loop closure. As a consequence of this comparison, we find out that the Magnetic Field sensor is a valid candidate for place recognition by using the proposed simple dissimilarity function. Finally, we present the recovered walking path results.

1 Introduction

The very first proper positioning technology began with the Global Positioning System (GPS) in the mid-1960's. Over the last three decades, location-based technologies have started to find its place in commercial use cases. With the advancement and the prevalence of smartphone devices, it became more and more integrated into our life. Nowadays, we mostly handle our outdoor day-to-day navigation and positioning tasks with smartphones. Even though the GPS is the standard technology for the outdoor application, it fails at indoor. Meanwhile, researchers also have been trying to solve indoor positioning problem by utilizing many different technologies, e.g., ultrasonic systems, ultra-wideband systems or RFIDs but they all require substantial infrastructure setup.

Besides these technologies, the main interest in this paper is to solve the indoor localization problem with the existing technologies that is already available in a smartphone. In literature, three types of positioning systems exist. One of them is Geometric Wireless Positioning systems, where the received signal strength information is exploited to gather the distance information between receiver and sender. This technique suffers from Non-Line-of-Sight (NLOS) measurements since the wireless technologies are built for communication purposes but not the positioning. This makes the distance based applications limited and primitive. The second one is Fingerprinting, where the distinct blueprints are recorded in a database and are matched with the online measurements. One can record fingerprints using Wi-Fi, BLE and Magnetic Field information since they all have unique values at different locations based on the configuration of a building. This method provides 2-3 meters accuracy but this accuracy decreases as the radio map gets outdated over time. Therefore, one has to maintain the database by updating regularly. That means we need to find a way to save the fingerprint map in a labor efficient manner. The third and last technique is the Inertial Navigation System. With this system, one can track user's walking path by implementing *Pedestrian Dead Reckoning (PDR)* algorithm. However, the algorithm requires reliable *Inertial Measurement Unit (IMU)* sensors to operate properly since it is an accumulative method and can result in huge drifts in the absence of a calibration mechanism.

There have been enough studies that identify each method's drawbacks. The latest trend is to *fuse* them into a more complicated algorithm to get more maintainable positioning results. At this point, we choose SLAM because it stands out from rest of the sensor fusion algorithms due to its proven success in many difficult applications. In this paper, we are going to explore the SLAM and experiment with both simulation and real-world data to prove its capabilities.

2 Motivation

SLAM offers fascinating results in extreme positioning applications, ranging from exploring the surface of Mars to autonomous driving. Although the popularity of SLAM seems recent, the core idea is quite ancient if we think our ancestors in a situation where they find their way by looking at the star's position during the expeditions. We, humans, are naturally good at recording spatial memory to navigate ourselves in unknown environments. SLAM introduces many technical challenges, especially to the autonomous and mobile systems. This attracts many researchers' attention and I am one of them.

This attraction leads me not only to understand the SLAM problem but also to implement the SLAM in the rather different context. Therefore, the main goal is to convert the general SLAM problem that is used in robotics to the smartphone scenario. To do so, we explored the majority of the built-in sensors that are available in an average commercial smartphone and tried to answer the following research questions:

- Can we build reliable and consistent indoor maps with smartphone using SLAM?
- Can we recognize revisited places during mapping without sophisticated sensor capabilities?

Besides addressing these questions, we also lay out a recipe for a smartphone in the presence of limited smartphone's sensors.

3 PDR

Dead Reckoning Systems aim to track objects' relative motion by measuring the IMU that is carried with the object's body. The easiest way to track pedestrians' motion is by utilizing the smartphone's built-in Microelectromechanical Systems (MEMS) such as:

- **Accelerometer** contains a mass between two springs that are connected to the capacitive plates on each axis of te 3D Cartesian coordinate system. Thus, any kind of movement on the smartphone's body will cause to wobble mass between springs, which will then result in the capacitance changes. One should keep in mind that accelerometer is sensitive to any kind of movement. Thus, one should be able to distinguish specific motion behaviors from others, such as step events.

- **Gyroscope** senses angular velocity by means of *Coriolis effect*, which occurs the moment when the frame of reference is rotating while the mass is moving. To catch this effect, gyro has an oscillating mass connected to capacitive coms so that it can sense the rotating movements whenever the capacitance is changed. Overall, gyro provides accurate angular velocity in the short term, which then can be used to calculate the *relative* orientation. To get this information, we need to perform single integration the angular velocity to get the angle. Even though you get rid of the noise from the gyro data by integrating, you cause accumulative drifts on the orientation on the long term.

- **Magnetometer** detects changes in voltage or resonant frequency around the 3-axis. These changes are caused by a phenomenon called *Hall effect*, which magnetometer can measure electronically. Through the use of 3-axis, we can transform the locally measured magnetic field that is based on the smartphone's current body orientation to the global (world) coordinate system where z-axis is perpendicular to the ground. Then, one can calculate the orientation information relative to the earth. Yet, the issue with the magnetometer is that measurements will also include every possible magnetic field forces around the smartphone (e.g., ferromagnetic materials inside the building). This causes extra magnetic distortions on the measurement instantaneously. At the same time, the magnitude of 3-axis magnetic field force will be shifted towards the distortion, which will then result in a momentarily shift on the heading angle.

Luckily, each sensor has a different kind of error characteristic so that we can calibrate these errors by using sensor fusion techniques.

The idea of the PDR systems is to combine timestamps of the users' step, heading of the steps and step length information to track the target's path. If the starting location (x_0, y_0, θ_0) is known along with the aforementioned information, the walking path of a smartphone user can be built as follows:

$$x_t = x_{t-1} + l_t cos(\theta_t) \tag{1}$$

$$y_t = y_{t-1} + l_t sin(\theta_t) \tag{2}$$

where step length estimation is l_t, heading estimation is θ_t, coordinates of the current step and coordinates of the previous steps are (x_t, y_t) and (x_{t-1}, y_{t-1}), respectively.

3.1 Step Detection

PDR's walking path construction relies on how accurately we count steps of a smartphone user. To detect a step event, we record the accelerometer sensor data continuously and recognize the acceleration and deceleration pattern of the movement. We sample the accelerometer sensor at 50 Hz, which is enough to observe the step patterns if the average walking speed is considered as around 1.5 m/s. However, before we perform the step detection algorithm, two preprocessing operations are needed due to the raw accelerometer sensor's noisy data.

$$a_{eff,t} = \sqrt{a_{x,t}^2 + a_{y,t}^2 + a_{z,t}^2} \tag{3}$$

The obvious solution for step detection is to recognize the varying acceleration patterns along the vertical axis that is relative to the ground. However, an accelerometer can only measure based on the phone's body. Since the orientation information of the phone body produces errors, we perform the step detection algorithm on the magnitude of the acceleration. $a_{eff,t}$ represent the acceleration's magnitude and 'eff' corresponds to the acceleration caused by *gravity*, which we need to remove as well.

In the first preprocessing operation, simple high-filter is used to remove this gravity effect from acceleration. Recommended α value are $0.9 \geq \alpha \leq 1$ and α is taken 0.9 in our experiments.

$$g_{z,t} = \alpha g_{z,t-1} + (1 - \alpha) a_{eff,t} \tag{4}$$

Then, we can remove the gravity effect from the acceleration magnitude. This high-passed filtered a_{HPF} sensor data is also called linear acceleration.

$$a_{HPF} = a_{eff,t} - g_{z,t} \tag{5}$$

Second and last preprocessing operation for the acceleration is low-pass filtering. We choose a simple moving average filter for reducing the random noise in the acceleration data. m represent the window length of the moving average filter and it is taken 10.

$$a_{LPF}[i] = \frac{1}{m} \sum_{j=-(m-1)/2}^{(m-1)/2} a_{HPF}[i + j] \tag{6}$$

After preprocessing of the raw acceleration data, a step detection algorithm can be applied. The simplest approach would be zero-crossing method but this causes to count step events that are in fact not steps. Therefore, we apply *peak detection* algorithm to recognize the step events. The idea is to not to have only one threshold, which is the zero-crossing case but to have two thresholds, which are called *maxima* and *minima*. With the help of these two thresholds, whenever we recognized a pattern where the acceleration holds a value that is below minima threshold after exceeding the maxima threshold within the time

span tolerance, we call it as a step event. In Figure-3.1, blue circles and red circles represent the accepted maxima events and minima events, respectively. In our experiments, we take maxima 0.2 m/s^2 and minima -0.2 m/s^2

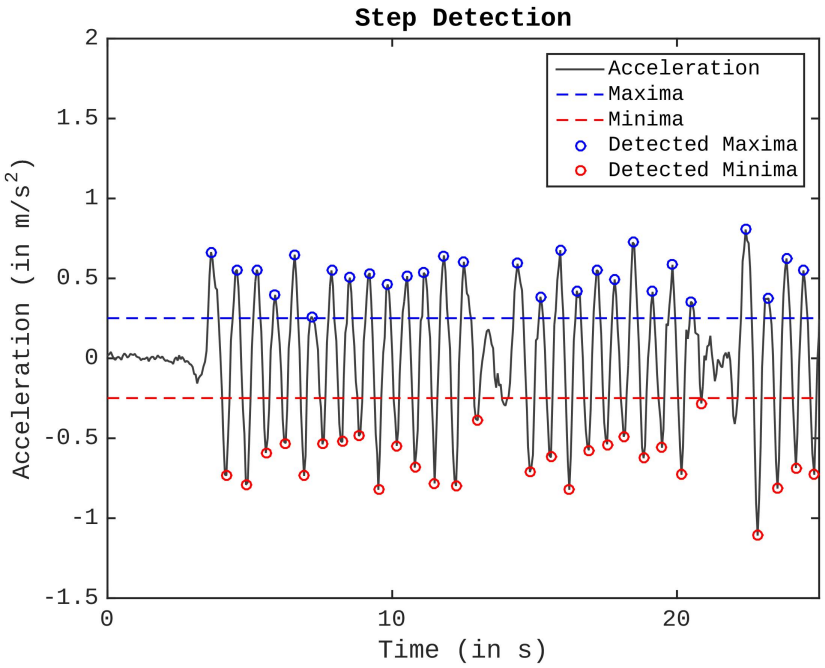


Figure 3.1: Peak Detection Algorithm

To put more simply, we first search for an acceleration value that is greater than maxima threshold and save the timestamp of the maxima event. Then, we expect to observe a value that is less than mimima threshold and if it occurs, we save the timestamp of the minima event too. At the final step, we check the timestamp difference between the maxima event and the minima event that is saved previously. If the difference is less than 1.5 second, we accept this event as a step. If not, we reject the event and the state goes back to search for another maxima event. For more details of the peak detection algorithm, I want to refer readers to the [**?**].

With the peak detection algorithm, one can utilize this technique to build the walking path in an online fashion, but we only record the step event timestamps for our offline SLAM algorithm, which we will be discussing in section-4.

3.2 Step Length

Since our goal is to construct an accurate walking path, we should avoid assigning static step lengths. Instead, we should adopt one of three dynamic step length estimation models[1]. Before selecting any of these models, we compared

[1]For comprehensive analysis for the step length estimation, I refer readers to [**?**]

them with the same preprocessed acceleration sensor data.

The first option is the *Scarlet* model, which takes into minimum, maximum and average magnitude of the acceleration during one step event. k is an empirical constant and is taken 0.8.

$$l = k \cdot \frac{\frac{\sum_{k=1}^{N} |a_{LPF,k}|}{N} - a_{LPF,valley}}{a_{LPF,peak} - a_{LPF,valley}} \tag{7}$$

The second option is the *Weinberg* model, which considers the minimum and maximum magnitude of the acceleration. In this case, k is taken 0.5.

$$l = k \cdot \sqrt[4]{a_{LPF,peak} - a_{LPF,valley}} \tag{8}$$

The third option is the *Kim* model, which only considers the average magnitude. k is taken 0.65 for the Kim model.

$$l = k \cdot \sqrt[3]{\frac{\sum_{k=1}^{N} |a_{LPF,k}|}{N}} \tag{9}$$

Choosing empiric constant k that is close to the real average step length can be achieved. However, real concern that is experienced with the Scarlet and Kim models are that they estimate largely different than one another, which ultimately cause to deviate from the step mean. Keep in mind that the user was asked to keep his steps as constant as possible throughout the walking path. We can observe these deviations in the Figure-3.2. For this reason, we use the Weinberg model for our step length estimation.

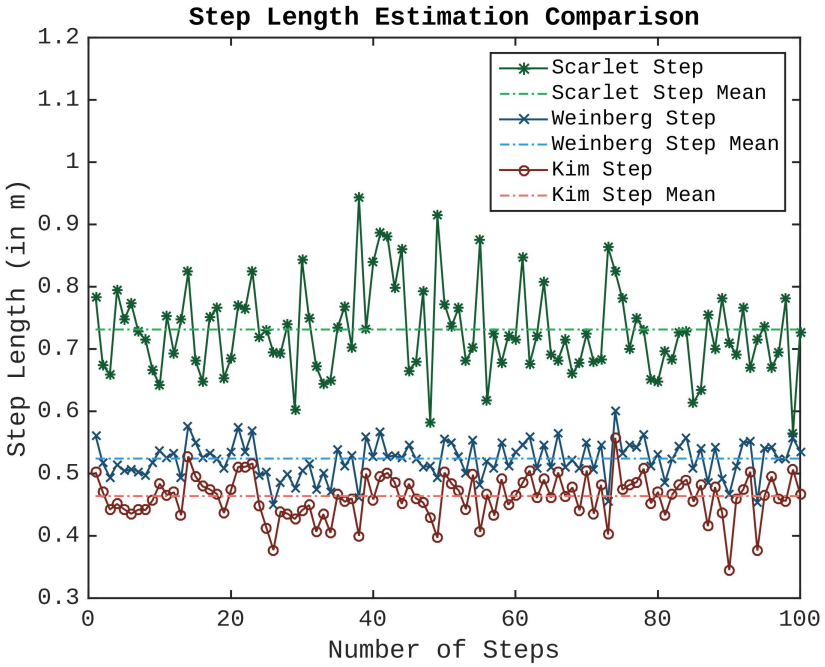


Figure 3.2: Comparison Between Scarlet, Kim and Weinberg

Note that there are still limitations with regards to step detection algorithm and step length estimation because the empirical constants are fixed to the specific user's walking characteristic, which might produce erroneous results if the sensor data is collected by another person.

3.3 Heading

Acquiring accurate heading information is the most challenging part of the PDR algorithm for indoor applications. As we explained at the beginning of this section, each IMU sensor has a different attribute and each has advantages and disadvantages. For example, gyroscope provides instantaneous angular velocity, which we can extract the $\Delta\theta_i$ *relative angle* with integration but we need a starting (fixed) angle θ_0 if the absolute angle is needed. Additionally, we need to correct drifts, caused by integration, regularly with another *absolute* sensor type. For the magnetometer, there are random erroneous heading results due to the distortion in the indoor environment as we discussed earlier. Therefore, magnetometer also needs another sensor for compensating these errors. On the other hand, accelerometer can provide orientation information by measurement displacement on certain axis, but there is again drift issue due to the double integration.

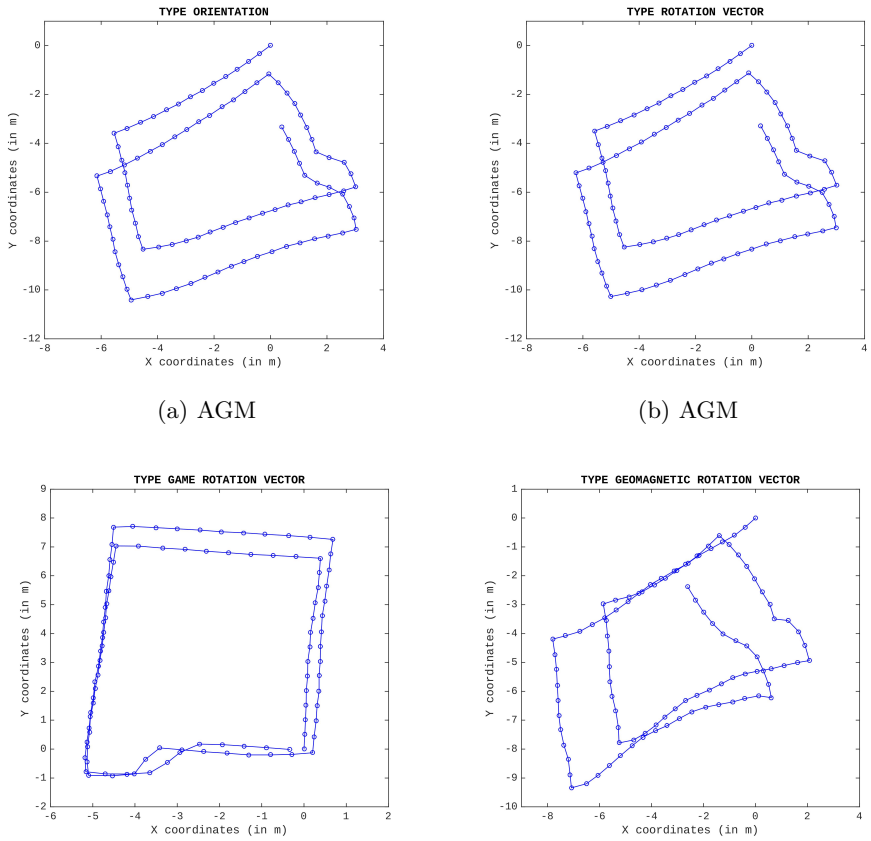
Android offers different kind of sensor fusion algorithm using different sensor combinations:

Table 1: Android Libraries for Orientation

Library Name	Description	Method	Angle Type
ORIENTATION (AGM)	accelerometer, gyroscope, magnetometer	Complementary Filter	Euler
ROTATION VECTOR (AGM)	accelerometer, gyroscope, magnetometer	Kalman Filter	Quaternion
GAME ROTATION VECTOR (AG)	accelerometer, gyroscope	Kalman Filter	Quaternion
GEOMAGNETIC ROTATION VECTOR (AM)	accelerometer, magnetometer	Kalman Filter	Quaternion

To show the differences of each method, we implemented PDR algorithm by using a small portion of the recorded walking path. Figure-3.3a and As we can see, Figure-3.3b produces almost identical heading results. However, ORIENTATION method uses Euler angle, which has a well-known issue called gimbal lock when the pitch angle is 90 degree. Therefore, this method is deprecated and only used in old Android smartphone models. Figure-3.3c shows the how gyroscope and accelerometer perform highly accurate for short time period; however, it doesn't have an absolute heading. Therefore, the resulting

walking path isn't fixed to the global coordinate system. Lastly, we can observe the indoor related distortions in Magnetometer in Figure-3.3d when the user is walking straight line. Notice the bending effect of the straight line.



(a) AGM (b) AGM

(c) AG (d) AM

Figure 3.3: Heading Comparison

In our experiments, we used only two of these libraries; i.e., 'TYPE_ROTATION_VECTOR' and 'TYPE_GAME_ROTATION_VECTOR'. The former provides the most accurate absolute heading results in terms of the topology of the walking path. Even though it contains the magnetometer's random heading errors, it doesn't deform the path's shape, which is important for our SLAM algorithm. The latter doesn't have absolute heading information but it is sufficiently accurate for building the ground truth path.

4 SLAM

Simultaneous localization and mapping (SLAM) algorithms offer solutions to a problem where robots acquire map information of its environment while simultaneously localizing themselves. All the available information for estimating its poses $x_{1:t}$ and map M of the environment are *control actions* $u_{1:t}$ and *environment perception measurements* $z_{1:t}$. Typically, a mobile robot receives or produces motion decisions to change its current position to move around while measuring its position changes through a method called *odometry*. However, the motion sensors that are used in this process produce noisy and drifty data, which can lead to a wrong estimation of the positions. To compensate these error, the robot constantly performs series of perception measurements in order to recognize previously visited places. In this way, the robot has a great chance to correct its odometry based errors whenever the robot recognizes the *revisits* during mapping.

In a Graph SLAM[2] system, abstraction is needed to deal with the complexity. One of the widely-accepted SLAM frameworks is the *front-end* and *back-end* style modeling, as it is shown in the Figure-4.1. In the front-end, raw sensor data is stored in the memory to be preprocessed. In this preprocessing stage, several filtering techniques are applied to remove the noise or proper data structure is formed for the further processing. Then, data association is performed with the preprocessed sensor data to recognize revisited places (loop closures). At the last stage of front-end, graph construction is executed constantly. With the provided graph, pose state optimization is done to eliminate accumulated position drifts from the system. Finally, the map is generated.

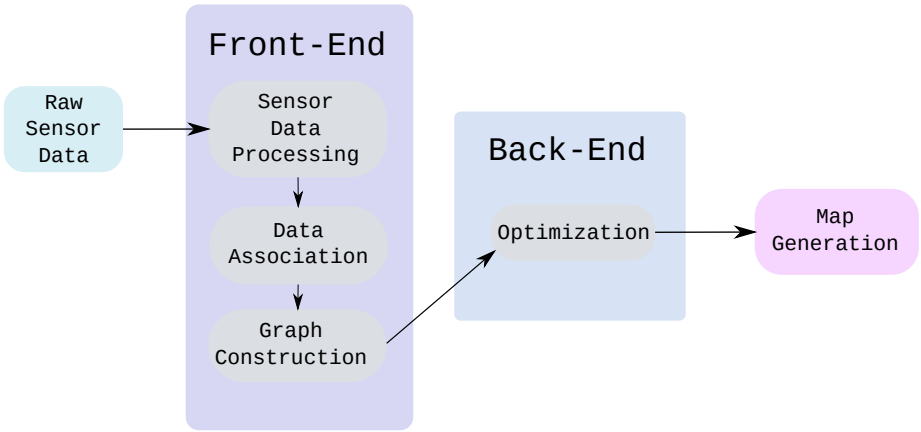


Figure 4.1: SLAM Framework

The problem that we try to tackle in this paper, in fact, does not involve robots, instead *pedestrians* in an indoor environment. This make the SLAM problem slightly different than mobile robotics problem. Thus, we need to make several distinctions:

[2]There are other SLAM systems in literature; however, we only deal with the Graph SLAM in this research project and whenever we call SLAM, we refer to Graph SLAM

- **M**apping: A robot has the ability to explore the environment in such a dense way that it can create the occupancy grid maps and save features like unique landmarks. In pedestrian case, this density is not possible due to the limitations of the sensors in an average smartphone. The maps created by smartphone using SLAM have less resolution and topographically simpler. However, this is still valuable information for smartphone user oriented positioning and navigation applications where 2-3 meters accuracy is enough. Our goal is to find a way to represent smartphone generated maps with the abstracted fashion. The candidate we choose for this purpose is *pose graphs*, which we will be explaining in section-4.1.

- **L**ocalization: In terms of finding the accurate position, odometry data is much more precise in robots because the smartphone lacks such quality IMUs and encoders so the drifts caused by motion sensors are worse in the smartphone. The other and critical issue is that we don't have the capability to recognize revisited places through feature extraction of landmarks. Robots are equipped with the sensors (e.g., RGB-D cameras, laser scanners or radars) that can provide high precision (2D or 3D) range and/or bearing information of the landmarks. This does not only provide place recognition capability but also produces location information relative to the landmarks. If the landmark position is accurate enough, this information can be converted to an absolute location. With the smartphone, you have the range only observations that one can exploit by collecting RSSI, which is a 1D measurement. To get the 2D or 3D coordinate position, triangulation calculation is needed by measuring RSSI from at least 2 or 3 access points. Still, this RSSI triangulation methods suffer from multi-path or shadowing effect, which is widely known issue in RSSI based localization technologies. The other possibility is to use the mono-camera in smartphones but we will not be discussing in this paper. Since getting location information through landmark observations is trivial, we need to express these revisited place in the SLAM concept. We achieve this need with *fingerprint* method. With this method, we constantly compare the current observation with the fingerprint history, except we produce an error function that informs the optimizer how drifted we are at the current position. This fingerprint information can be gathered from RSSI or magnetic field measurements. We will discuss this (loop closure) error functions in the following part of this section.

- **S**imultaneous: The difficulty of the SLAM problem lies in this 'S' part, which is to generate the map during the localization. Two types of concurrency form are offered in the literature; i.e., online SLAM in which the interest is in probability $p(x_t, m|z_{1:t}, u_{1:t})$ of the current pose and full SLAM in which interest is in probability $p(x_{1:t}, m|z_{1:t}, u_{1:t})$ of the all the poses. In our case, we will be proceeding with the full SLAM. For further information about general SLAM problem, I refer readers to [**?**].

4.1 Graph SLAM

SLAM can be modeled with the graphs in which nodes represent poses and edges present dependencies or constraints between nodes. *Dynamic Bayesian Graphs (DBG)* are suitable for comprehensive representation and better understanding.

In DBG, there exist two kinds of property; i.e., *observable* nodes and *unobservable* nodes. Observable nodes contain the estimation of the pose state and of the landmark observation. However, one can never know the real positions of the pose and of the landmarks, which are unobservable nodes. In Figure-4.2, dark nodes are the observable node and white nodes are the unobservable nodes:

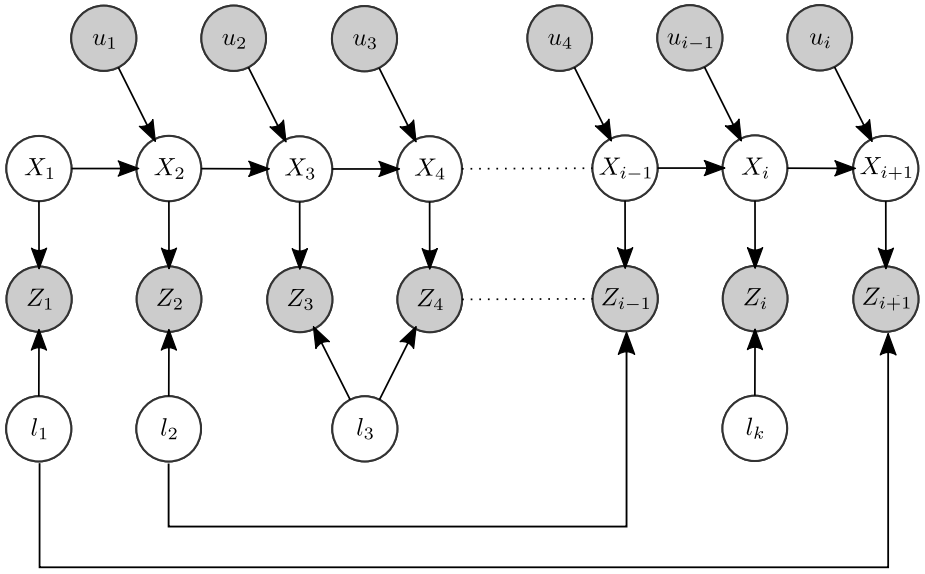


Figure 4.2: Dynamic Bayesian Graph

IMPROVE below!

- True pose states: $X_{1:i+1}$[3]
- Estimated pose states and landmark states: $Z_{1:i+1}$[4]
- True landmark states: $l_{1:k}$
- Estimated control actions: $u_{1:i}$[5]

Notice in Figure-4.2 that l_1 is observed both in X_1 and X_{i+1} pose states. This means that those two poses are the revisited place. Same rule applies for the l_2 and l_3.

Another graph representation of the SLAM is called *Factor Graph*, which is a compressed version of DBG where pose states have relations via undirected edges in a much simpler way. This type of graph is also called bipartite undirected graphs. Since we don't distinguish true and estimated pose in Factor Graph, we will refer as $X_{1:i+1}$ pose nodes from now on. In Figure-4.3, pose nodes are connected two types of edges; i.e., red edges, representing the odometry

[3]True pose state can mean real world positions (x_i, y_i, z_i), real wold heading θ_i or both. Similarly, true landmark state can have similar properties, (l_i^x, l_i^y, l_i^z) and l_i^θ.

[4]Based on the model, we can estimate positions and heading $(\widehat{x_i}, \widehat{y_i}, \widehat{z_i}, \widehat{\theta_i})$, and landmark positions $(\widehat{l_i^x}, \widehat{l_i^y}, \widehat{l_i^z}, \widehat{l_i^\theta})$ or both.

[5]Estimated control actions can be a displacement vector $(\Delta x, \Delta y, \Delta\theta)$.

constraints and blue edges, representing the landmark constraints, which are also called *loop closures* constraints.

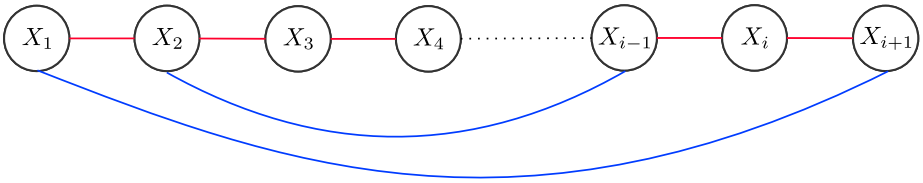


Figure 4.3: SLAM Factor Graph

In the case of a smartphone, pose nodes relates to step events. In addition, red edges connect two successive steps via PDR information and blue edges connect two revisited pose nodes that are proposed by RSSI or magnetic field observation. Further details about how we create the blue edges will be discussed in section-4.5.

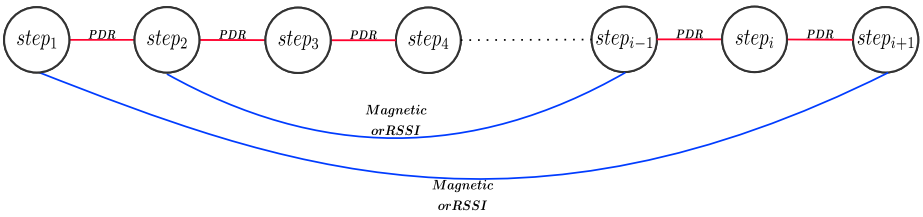


Figure 4.4: SLAM Factor Graph (Indoor Scenario)

4.2 Simulation Environment for Graph SLAM

The graph representation of the SLAM problem provides a sufficient modeling; however, the intuition gets much clearer when we combine with the walking path visualization. For this reason, we create a simulation environment in Matlab for our smartphone SLAM purposes. Initially, a ground truth walking path with the 125 steps and 5 access points[6] as landmarks are created. The starting position for the walking is $(x_0, y_0) = (0m, 0m)$ and the ending position is $(x_{125}, y_{125}) = (25m, 0m)$ as it can be see in Figure-4.5. We also assume that at least one RSSI measurement from each access point is available and is stored at each step.

[6]In the simulation environment, we gathered landmark constraints only from RSSI measurements. We did not implement magnetic field fingerprint method at this stage, since we only used this simulation environment for testing the robustness of SLAM algorithm in terms of smartphone application.

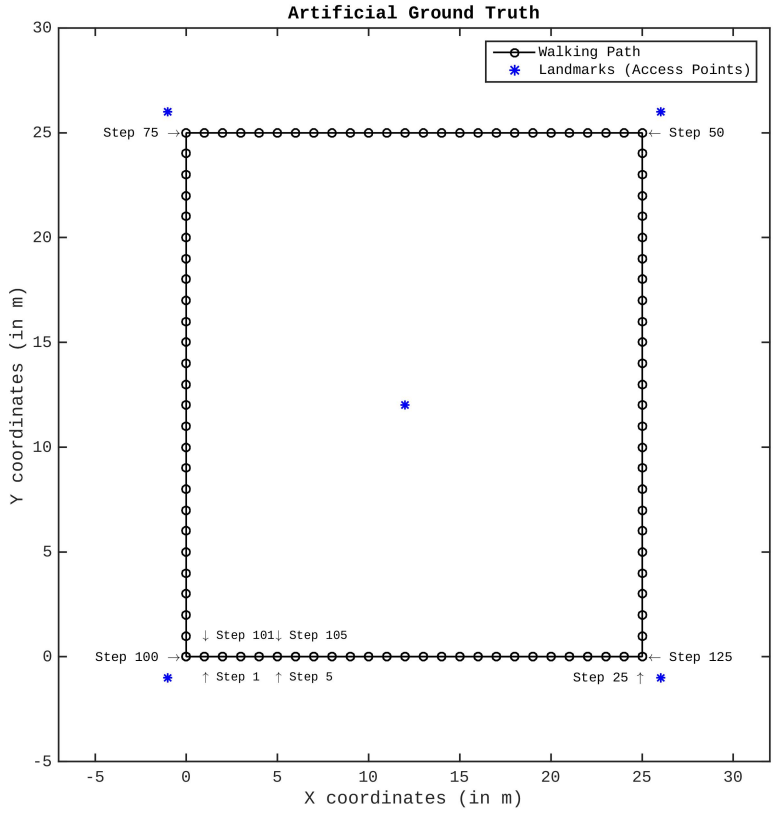


Figure 4.5: Ground Truth in the Simulation Environment

Then, noise is introduced to the system to represent the *drifting effect* of the IMU sensors. In Figure-4.6, the walking path starts from the same position but it gets drifted by noisy step length and step heading measurement and it ends at $(x_{125}, y_{125}) = (31.6m, 6.1m)$, which $8.9m$ away from its real position. The biggest motivation behind the SLAM is not only to correct the current localization error but to correct the whole walking path by keeping the topology of the route as close as possible to the ground truth. Note that, we still hold RSSI measurements from each access point in every step. Thus, we can build the *loop closures* for the revisit locations.

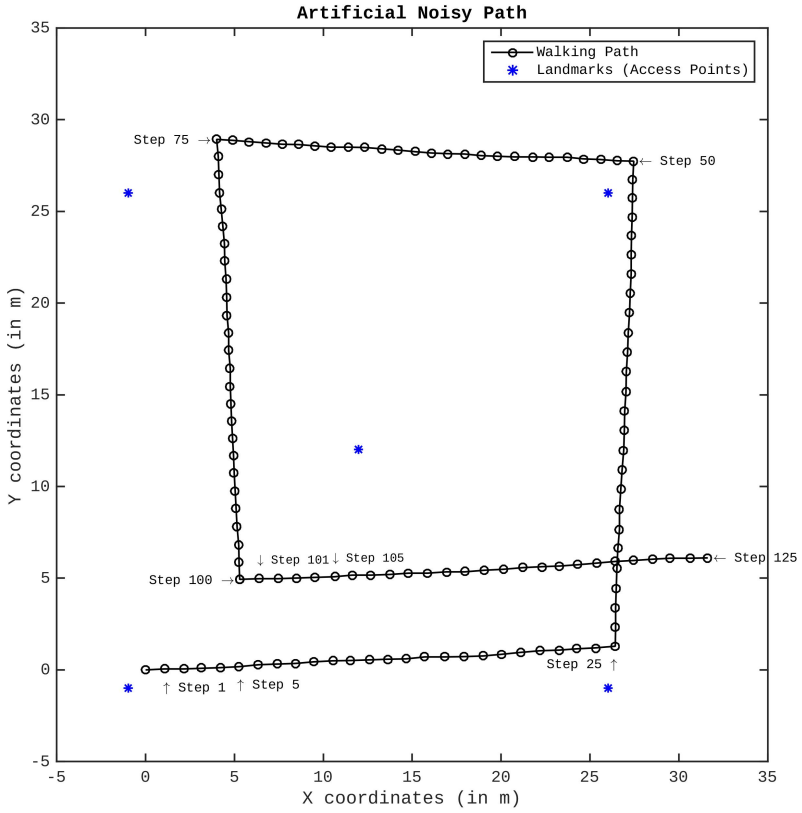


Figure 4.6: Noisy Path in the Simulation Environment

As we can see in the ground truth path, the starting point is revisited at the 100^{th} step after walking a square shaped path. After this moment each step starts to relate to its own revisited location since it is the same walking route after 100^{th} step. In Figure-4.7, the loop closures are represented with the red lines.

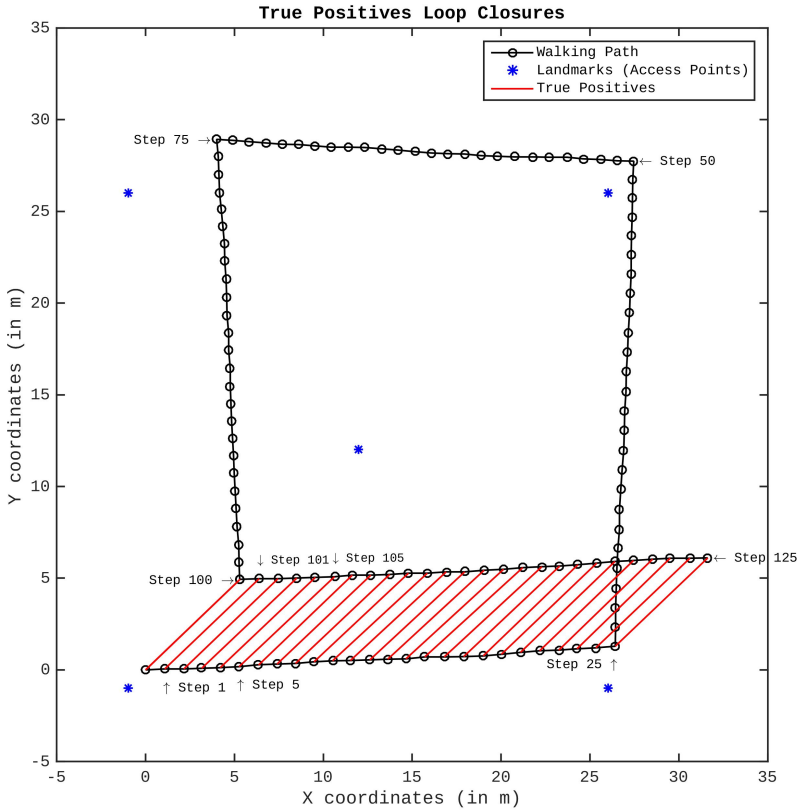


Figure 4.7: True Positives Loop Closures in the Simulation Environment

4.3 Graph SLAM Formulation

For our mapping purposes, it is sufficient that we have 2D cartesian plane and represent this plane with the state vector $\boldsymbol{x} = (x, y)^\intercal$. The motion model that we are going to build requires an assumption, which will help us to leverage efficient statistical methods. The assumption is that successive poses and landmark poses are *normally distributed random variables* with variance Σ_i and Λij, respectively. x_{i+1} represent the next pose after x_i.

$$x_{i+1} \sim \mathcal{N}(f(x_i, u_i), \Sigma_i) \tag{10}$$

x_j represents the current pose that has loop closure relationship with x_i pose that is visited previously. u_i and u_{ij} are the motion control actions as we also discussed earlier.

$$x_j \sim \mathcal{N}(f(x_i, u_{ij}), \Lambda ij) \tag{11}$$

These motion models tell us that there exist pose states that are normally distributed. Therefore, we aim to find such a state configuration of the poses

that we get the most likely, in technical words, *maximum a posteriori* and the probabilistic representation is as follows:

$P(X|U)$ is the conditional probability, which represents the probability distribution of the pose states X given the control actions U. That being said, we can now use the likelihood function $P(X|U)$ to calculate the most likely state configuration X^* as map information.

$$X^* = \arg\max_X P(X|U) \tag{12}$$

Then, we elaborate the probability distribution by splitting into two using motion models, i.e., odometry and loop closures. In addition to splitting, we can define the joint probability of each pose by the product rule so that the estimation for the state of each pose contributes the overall likelihood.

$$P(X|U) \propto \underbrace{\prod_i P(x_{i+1}|x_i, u_i)}_{\text{Odometry Constraints}} \cdot \underbrace{\prod_{ij} P(x_j|x_i, u_{ij})}_{\text{Loop Closure Constraints}} \tag{13}$$

Now, we transform our problem to make it more computational friendly since calculating products is inefficient. However, derivation from equation 13 to 14 is not provided in this paper and I want to refer readers to [**?**]. The author explained the derivation in detail in his Ph.D. thesis. To calculate the maximum of the joint probability of our problem, we perform the *error minimization* approach so that we don't have to predict the state randomly. The idea is to minimize the difference between model estimations $\widehat{x_i}$ and state observations x_i. With this approach, one can *constrain* the state estimation from less likely ones. Then, maximum likelihood can be reached with an iterative error minimizing method, e.g., Least Squares method, which we will be discussing further in section-4.4.

As we have two motion models, we have two parts in equation 14 as well. For the odometry, the estimation for the next pose $\widehat{x_{i+1}}$ will be calculated by the model function $f(x_i, u_i)$. Same logic applies for the loop closure, except the loop closures might not necessarily represent the consecutive steps. That is the reason we have (i, j) pose pairs indexing.

$$X^* = \arg\min_X = \underbrace{\sum_i \|f(x_i, u_i) - x_{i+1}\|^2_{\Sigma_i}}_{\text{Odometry Constraints}} + \underbrace{\sum_{ij} \|f(x_i, u_{ij}) - x_j\|^2_{\Lambda ij}}_{\text{Loop Closure Constraints}} \tag{14}$$

Next step is to minimize the error function that we described using Least Squares optimization method.

4.4 Least Squares

As we discussed in the previous, error minimization operation is required to get the maximum likelihood. To do so, we search the most likely state configuration as close as possible to its ground truth. Before plugging the least square part into our SLAM problem, we explain how this method works.

Least squares problem is a non-linear optimization problem. The goal is to find interesting points, such as local/global maximum or local/global minimum,

on the *objective function*. Due to the non-linearity, one can only approximate by generating a quadratic model of the objective function and iterate through the interest points using *Newton's* methods. For example, the interest point in Figure-4.8 is at the local minimum that is highlighted as a red point cloud.

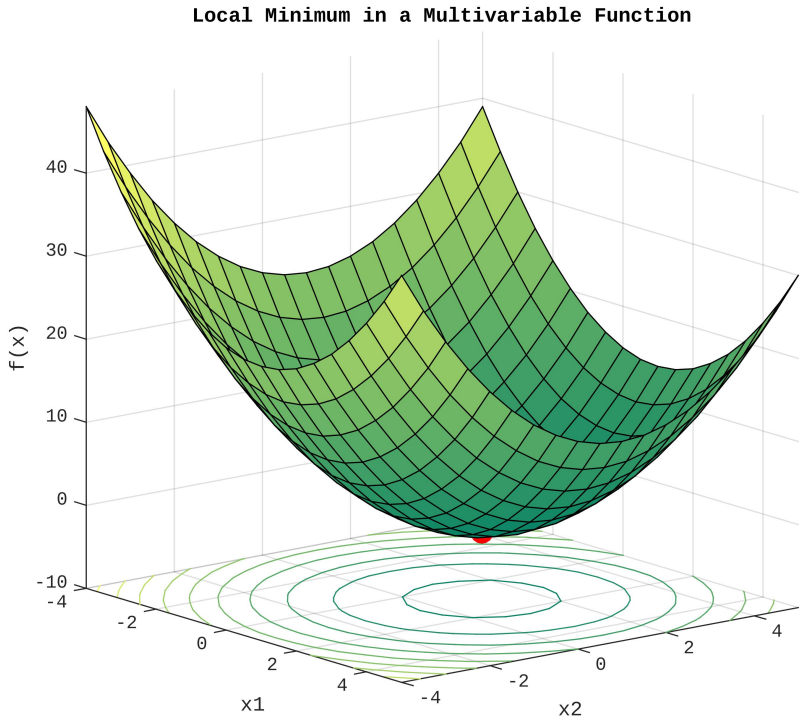


Figure 4.8: Local Minimum at a Quadratic Function

In literature, there are many versions of Newton's method and they all try to find the maximum/minimum points in the most efficient way. Levenberg-Marquardt (LM) method, which solves this optimization problem with the gradient descent approach with the help of the approximated Hessian function, is one of them and we will be using it in our SLAM algorithm.

Suppose that we have a model function $g(x; a)$. However, we don't know what the $x = (x_1, x_2)$ coefficients (also called *optimization parameters*) are and we can only plug a, which is the *independent* variable, into the *model* S to see how the output of the model changes given the independent variable.

$$\eta = g(x; \xi).$$

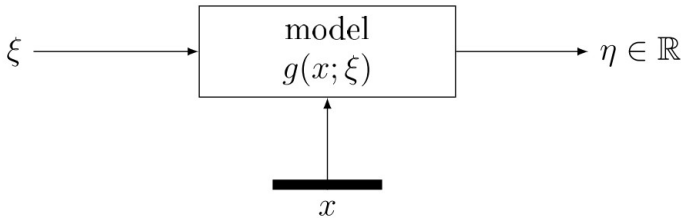


Figure 4.9: Least Square Model

$$\xi = a \text{ (independent variable)} \in \mathbb{R} \tag{15}$$
$$\eta = S \text{ (dependent variable)} \in \mathbb{R} \tag{16}$$

$$S = g(x; a) := x_1 + x_2 a \quad \text{where} \quad x = (x_1, x_2) \in \mathbb{R}^2 \tag{17}$$

To see this effect, we draw a graph that is shown in Figure-4.10.

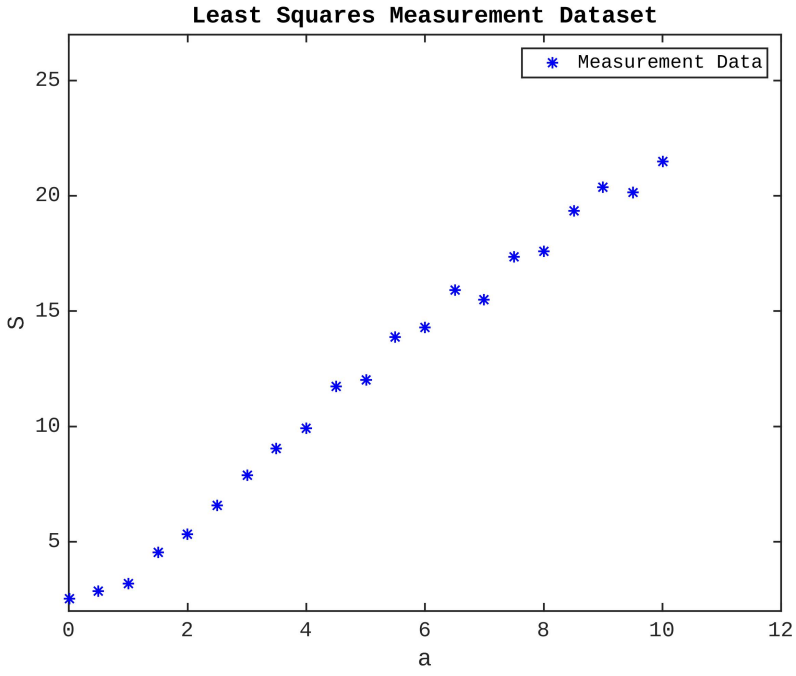


Figure 4.10: Least Squares Measurements

Our goal is now to find a linear function, which will fit this dataset. This is a typical least squares curve fitting problem, which we construct a *residuals*

function $r_i(x)$ by providing difference values between model estimation $g(x; a_i)$ and dependent variable S_i. In this case, the dependent variable represent the real world measurements and the residuals function represents the error between the estimated value and measurement value.

$$r_i(x) := g(x; a_i) - S_i \qquad \text{for } i = 1, \ldots, m. \tag{18}$$

Then, residuals function is squared to magnify larger error effect.

$$f(x) := \sum_{i=1}^{m} |r_i(x)|^2 = \sum_{i=1}^{m} |g(x; a_i) - S_i|^2 = \sum_{i=1}^{m} (g(x; a_i) - S_i)^2 \tag{19}$$

Now, we use the sum of squared error function to calculate the most likely configuration that can minimize the errors:

$$X^* = \arg\min_x F(x) = \arg\min_x \frac{1}{2} f(x)^\intercal f(x) = \sum_{i=1}^{m} (g(x; a_i) - S_i)^2, \quad x \in \mathbb{R}^2 \tag{20}$$

At this point, the objective function is ready to be handed over to LM method. The method will try to find the *optimal* solution X^* by minimizing the objective function.

$$\text{Minimize} \quad \sum_{i=1}^{m} (g(x; a_i) - S_i)^2, \quad x \in \mathbb{R}^2 \tag{21}$$

To help our understanding, the objective function is drawn in Figure-4.11. As we can see, the based on the given $x = (x_1, x_2)$ values, the objective function $F(x)$ behave as follows:

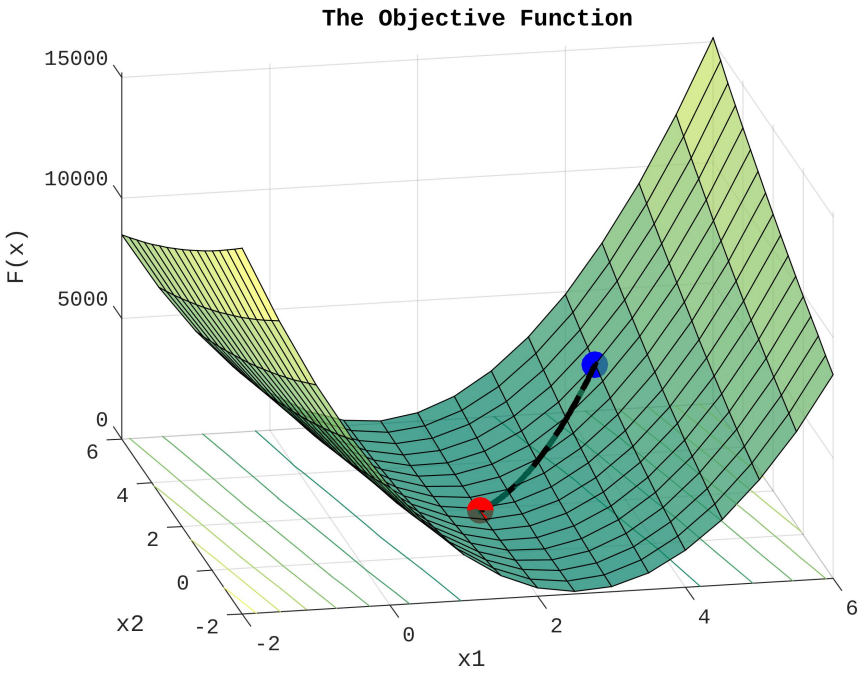


Figure 4.11: Local Minimum at Sum of Squared Error Function

The initial guess for the optimization parameters at $x = (4, 4)$, which is highlighted as a blue color point cloud and the local minimum point, which is the interest point of ours, is highlighted with the red cloud. LM method uses its gradient descent strategy to travel to the nearest local minimum. As we can see that the descent is successfully performed and the optimization operation results in the $x = (1.9, 2.1)$.

If we now place the optimization variables into our model function, we get the $g(x; a) = 1.9 + 2.1a$ and a fitted line based on the given dataset in Figure-4.12.

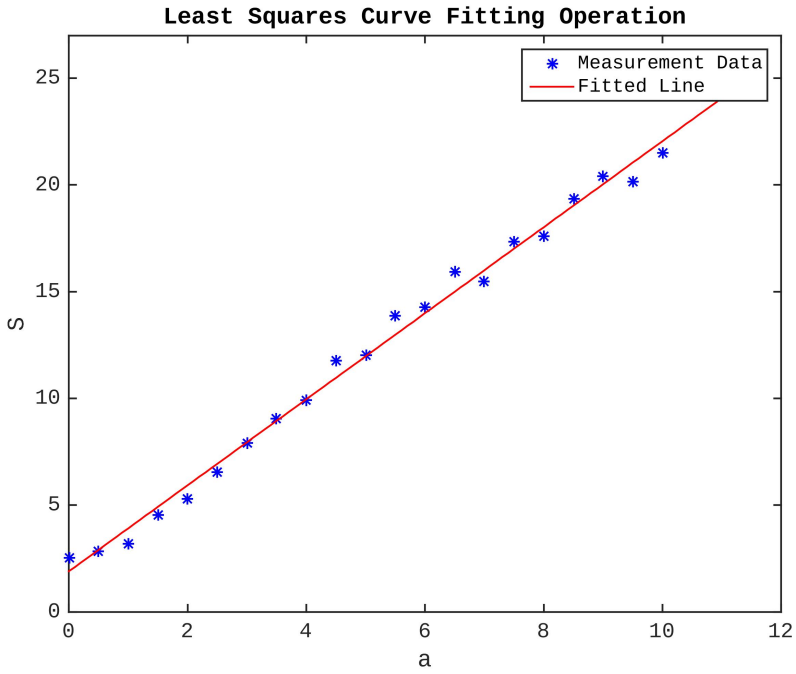


Figure 4.12: Least Squares Curve Fitting Operation

Keep mind that there are crucial factors whether LM will descent the local minimum to the interest point or not:

- *outliers* in the dataset,
- good *initial point guess*.

If these two criteria do not meet, LM might converge to the another local minimum or might not even converge to an optimal solution. It is important that we provide a good initial guess and minimum amount of outliers into any gradient descent optimization algorithm. Otherwise, we will not be able to converge to a local minimum that we are interested in since we will be stuck at another local minimum point. Not being able to find an optimal solution in SLAM context will result in an estimated state configuration is largely different from the ground truth.

4.5 Graph SLAM Setup for Smartphone

So far, we discussed how the general Graph SLAM problem is formulated. In this section, I am going to reformulate the Graph SLAM for the smartphone scenario. We already mentioned that odometry in robotics relates to step event in the smartphone. Therefore, we replace the odometry motion model with the PDR motion model. In this model, each step will have a state vector with $\boldsymbol{x} = (x, y)$, represents the 2D position coordinates. We don't include heading information into the objective function that is used for optimization of the walking path for our experiments.

The distinct separation for the smartphone SLAM is that landmark observations and its contribution to the optimization process. Usually, mobile robots are equipped with camera sensors and/or laser sensor. They provide 3D or 2D position results relative to the observed landmark, which easily be converted to the global coordinate system if the orientation of the robot is known. As we mentioned earlier, this is not possible with the RSSI based measurements as they only provide 1D measurement. Moreover, generating 2D position information with triangulation techniques does not provide accurate results. Therefore, we need to come up with a way to recognize the revisited places without using RSSI positioning capabilities. In the end, landmark observations are just loop closures constraints that help the optimization algorithm to converge to the minimum. If we find a way to create an error model function without using positioning algorithm, the optimization algorithm will still try to converge the minimum since the loop closure constraints behave like a *penalization factor*. For a now, we replace the $f(x_i, u_{ij}) - x_j$ with our error function model e_{ij}^{LC}, which we will explain in detail soon in this section.

By following previously mentioned changes, we transform the SLAM problem into equation-30.

$$\boldsymbol{X}^* = \arg\min_{\boldsymbol{X}} = \underbrace{\sum_i \|f(\boldsymbol{x}_i, u_i) - \boldsymbol{x}_{i+1}\|^2_{\Sigma_i}}_{\text{PDR Constraints}} + \underbrace{\sum_{ij} \|e_{LC}(\boldsymbol{x}_{ij})\|^2_{\Lambda ij}}_{\text{Loop Closure Constraints}} \tag{22}$$

To elaborate how we calculate the error function $e_{PDR}(x)$ based on PDR constraints, the following equation is provided.

$$f_{PDR}(\boldsymbol{x}) = e_{PDR}(\boldsymbol{x}) = \begin{pmatrix} f(\boldsymbol{x}_0, u_0) - \boldsymbol{z}_1 \\ f(\boldsymbol{x}_1, u_1) - \boldsymbol{z}_2 \\ \vdots \\ f(\boldsymbol{x}_{i-1}, u_{i-1}) - \boldsymbol{z}_i \end{pmatrix} \tag{23}$$

The error for each state is defined by a difference value between a model estimation $f(\boldsymbol{x}_0, u_0)$ and a measurement $\boldsymbol{z}$. The model estimation produces an estimated state vector $\boldsymbol{x} = (\Delta x_i, \Delta y_i)$. On the other hand, the PDR step detection algorithm produces a measured step length and a heading information where combine them into a state vector form $\boldsymbol{z} = (\Delta\widehat{x_n}, \Delta\widehat{y_n})$

Notice how we create loop closures in Figure-4.7. Due to the IMU caused drift issue, the walking path results in such a shape. Thanks to having one RSSI measurement from each access point in each step in the simulation environment, we are able to recognize revisited places, (e.g., step 0 and step 100, step 1 and step 101, so on) with the following dissimilarity equation:

$$dis(RSSI_i, RSSI_j) = \sqrt{\sum_{k=1}^{M} \frac{\|(RSSI_i^{(k)} - RSSI_j^{(k)})\|}{M}} \tag{24}$$

The dissimilarity function $dis(RSSI_k, RSSI_q)$ is simply a normalized euclidian distance that is calculated between two RSSI measurement vectors composed of surrounding access points. Note that one RSSI vector is measured at i^{th} step and the other one is at the j^{th} step.

Then, if the dissimilarity value is less the threshold that we empirically defined, we count it i and j steps as a loop closure. In the simulation environment, we take the dissimilarity threshold as 1 dBm. However, keep in mind that the threshold value will be different when we perform SLAM with real measurement.

$$f_{LC}(\boldsymbol{x}_{ij}) = e_{LC}(\boldsymbol{x}_{ij}) = e_{RSSI}(\boldsymbol{x}_{ij}) = d(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \quad (25)$$

Next step is to form an error model $e_{RSSI}(.)$ for the loop closures in a way that we can place into the objective function of our smartphone SLAM optimization problem. We again remind you that loop closures in Figure-4.7 are useful to generate this error model. The loop closure event says that step 0 and step 100 are in fact the same place due to the similarity of their RSSI measurement. By following this logic, we can also say that larger the euclidian distance between these two step, greater the error value should be. As a result, the loop closure will have more penalization effect during optimization operation. In the end, we expect from LM method to minimize the loop closure related error function. In addition to this loop closure related error function, we have the motion model error function in the same objective function as well. When these two error model functions are provided to the optimizer, LM will try to come up with such a state configuration that steps having loop closures constraints will be as close as possible. At the same time, the optimizer will have to keep the topological shape intact as we have PDR related constraints in the equation.

Finally, we present the objective function of our smartphone SLAM problem with the following:

$$F(\boldsymbol{x}) = \sum_{i=1}^{m} e_{PDR}^{\intercal}(x)\Sigma_i e_{PDR}(x) + \sum_{(i,j)\in C} \Lambda_{ij} e_{LC}(\boldsymbol{x}_{ij}) \quad (26)$$

The goal is to find a state configuration that minimizes the sum of squared errors:

$$\boldsymbol{X}^* = \arg\min_{\boldsymbol{x}} F(\boldsymbol{x}) \quad (27)$$

In our simulation experiments, we keep motion model variance Σ_i as an identity matrix and Λ as 1. The weighting of the each sensor measurement will be a future work.

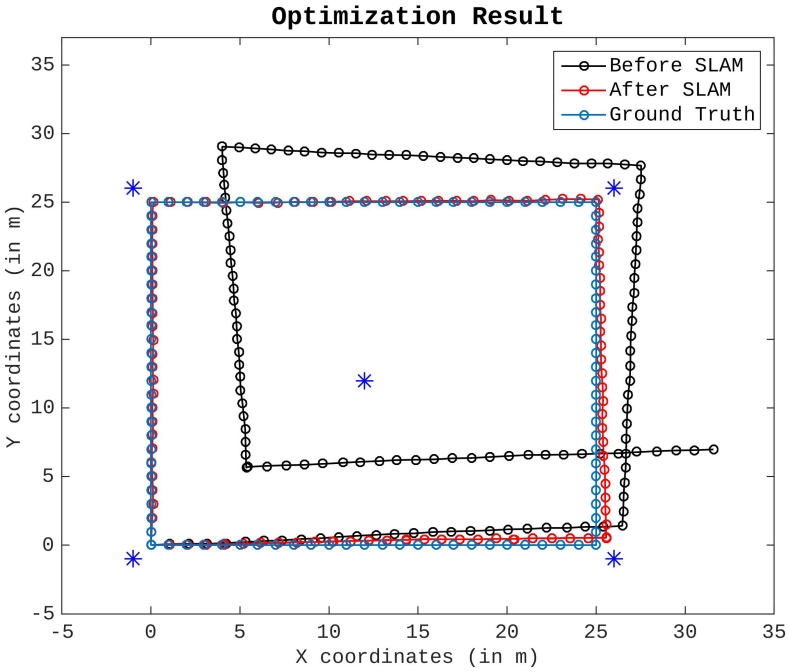


Figure 4.13: An Optimal Solution Found by LM

We also implement the SLAM algorithm by utilizing LM method that is already available in Matlab Optimization Toolbox. The walking path optimization result is presented in Figure-4.13 and the optimized walking path is nearly identical to the ground truth, as we can see.

4.6 Further Modifications on Graph SLAM Setup for Smartphone

Until this section, we only convert general SLAM problem to the smartphone SLAM problem by relying on the simulation environment. Nonetheless, the assumption that has been made is not realistic when considering the real-world data. For instance, RSSI measurement scans take about 2-3 seconds to complete. By the time we received the results, the user will be at the different positions already. Also, not having enough frequency in the scan rate disagrees with the assumptions that we will have at least one measurement for each. For SLAM to optimize the drifted path accurately, true positive loop closures are required but RSSI are prone to produce more *false positives*[7] than *true positives*[8]. We find out that the simple dissimilarity function that we created is not able to produce enough positive. RSSI requires more sophisticated algorithms to recognize the revisited places. That is the reason we experimented magnetic field sensor measurements to perform loop closures and compare to RSSI. Moreover, we try to tackle the false positive loop closures with more robust SLAM algorithm.

[7]False positive is a wrong statement that say a particular condition or attribute exist
[8]True positive says the condition or the attribute exists and it is a correct statement.

4.6.1 Magnetic Field Loop Closures

In addition to RSSI, we can also use magnetic field distortions to recognize the revisited place because it has unique fingerprints indoor. By comparing current step's magnetic field with the previous steps, we can tell if there is a correlation or not like we do with the RSSI.

By following the similar principle, we define a dissimilarity function for magnetic field measurements:

$$dis(MF_i, MF_j) = \sqrt{(MF_i^x - MF_j^x)^2 - (MF_i^y - MF_j^y)^2 - (MF_i^z - MF_j^z)^2} \tag{28}$$

Also, the same error model function that we used for the RSSI loop closures applies for the MF caused loop closures.

$$f_{LC}(\boldsymbol{x}_{ij}) = e_{LC}(\boldsymbol{x}_{ij}) = e_{MF}(\boldsymbol{x}_{ij}) = d(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{29}$$

4.6.2 How About Some False Positive Loop Closures?

Landmark observations in SLAM can provide such results that the place that is recognized as a revisited place, in fact, is a different place. In Figure-4.14, blue lines represent the false loop closures.

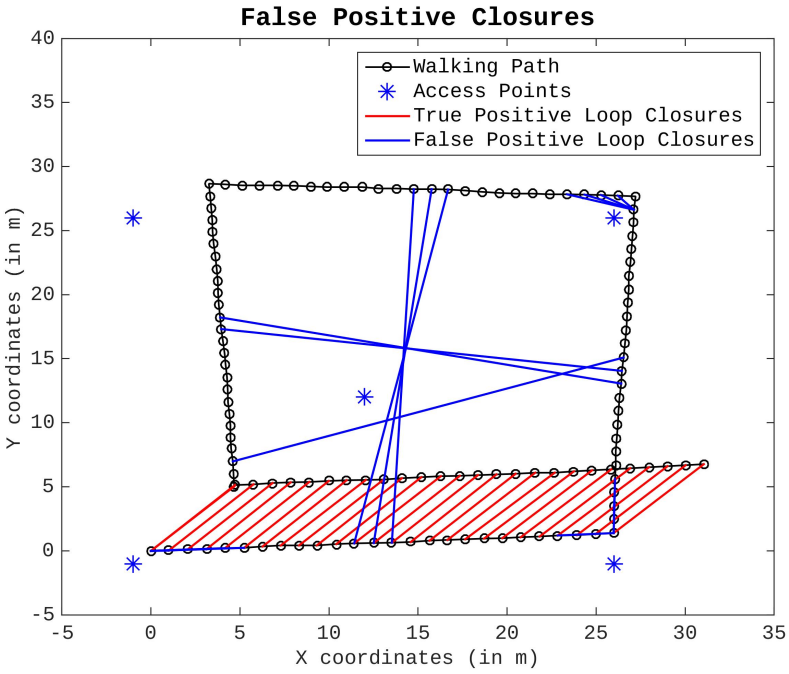


Figure 4.14: False Positive Loop Closures in the Simulation Environment

Unfortunately, these erroneous data associations have a huge impact on the optimizer's performance. Notice in Figure-4.15 that resulting optimized walking is unsuccessful. This problem is known as a convergence problem in SLAM. Filtering out the all outliers during the front-end is not a realistic design choice since there will be always some outliers in place recognition, which will eventually result in an false positive. Therefore, we have to deal with these false positives in the back-end where the optimization is performed.

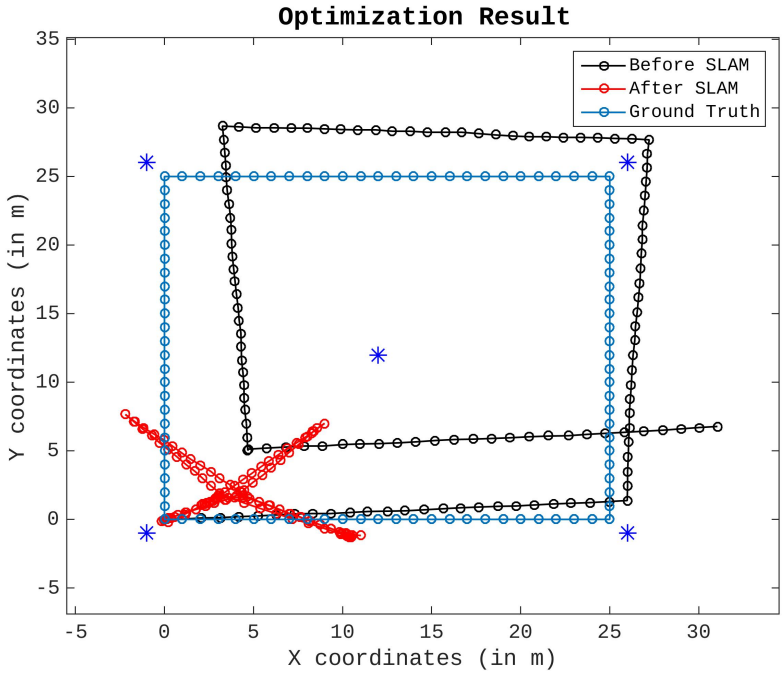


Figure 4.15: Optimized Path Results

The *robust back-end* for Graph SLAM approach [**?**] tackles this particular problem. The intuition behind this method is to disable the false positive loop closures by relying on expected drift effect. For example, we know that total drift caused by IMU will be up to 2 meters by the time the first loop closure occurs with the given walking path route. Therefore, we can assume that false positives loop closures that are a father than 2 meters happens to be outliers and they need to be deactivated during optimization.

$$
\begin{aligned}
\boldsymbol{X}^*, S^* = & \arg\min_{\boldsymbol{X}, S} = \\
& \underbrace{\sum_i \| f(\boldsymbol{x}_i, u_i) - \boldsymbol{x}_{i+1} \|^2_{\Sigma_i}}_{\text{PDR Constraints}} \\
& + \underbrace{\sum_{ij} \| sig(s_{ij}) \cdot e_{LC}(\boldsymbol{x}_{ij}) \|^2_{\Lambda ij}}_{\text{Switched Loop Closure Constraints}} \\
& + \underbrace{\sum_{ij} \| \gamma_{ij} - s_{ij} \|^2_{\Xi}}_{\textit{Switch Prior Constraint}}
\end{aligned} \tag{30}
$$

$$
\text{where} \quad sig(s_{ij}) : \mathbb{R} \to (0, 1) = \frac{1}{1 + e^{-s_{ij}}} \tag{31}
$$

By following this logic, the author proposes to introduce switch variables into the objective function. By doing so, we are able to switch off the false positives because the switch prior constraint part will produce larger errors comparing to switched loop closure constraints part. During our simulation experiments, we assume that the largest drift when the first loop closure happens is less than 5 meters. Therefore, we take $\gamma_{ij} = 5$ for all loop closure constraints as an initial guess, which means all loop closures are active at the beginning of the optimization process.

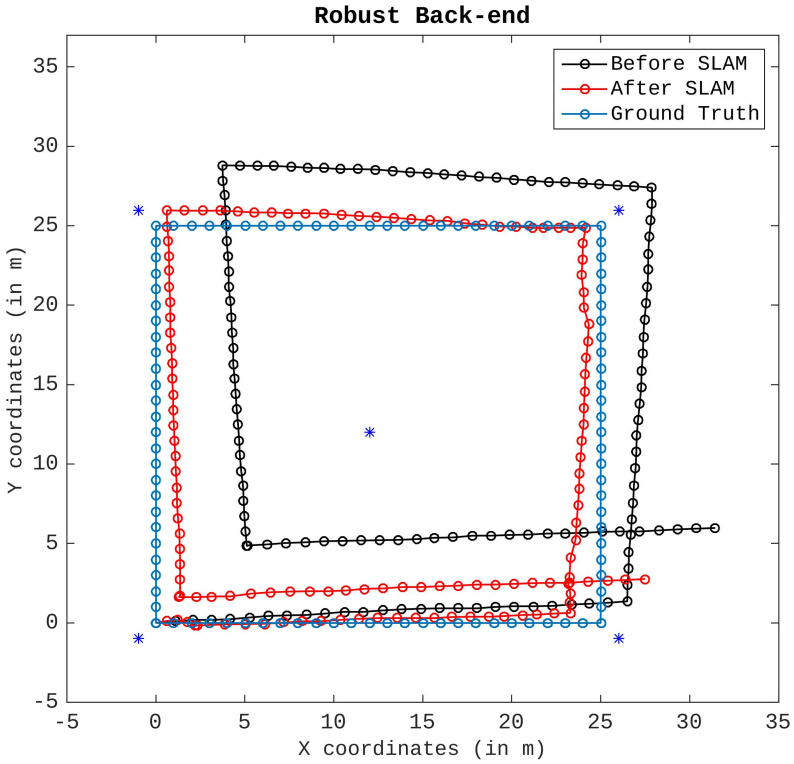


Figure 4.16: Robust SLAM Back-end Optimizer Results

By updating our SLAM formulation, we manage to converge to the optimal solution, which results in such an optimized path which is close to the ground truth, as we see in Figure-4.16.

5 Evaluation

Now that we prepare our SLAM recipe for Smartphone, it is time to evaluate how it performs with the real-world data and I will try to answer these two following questions in this section:

- Which of sensors that used for loop closures performs better than the other in terms of path convergence?
- What is the accuracy of the converged path results?

5.1 Device and Environment Settings

The device that we used in our experiments is the Lenovo PB2 and its technical summary is the following:

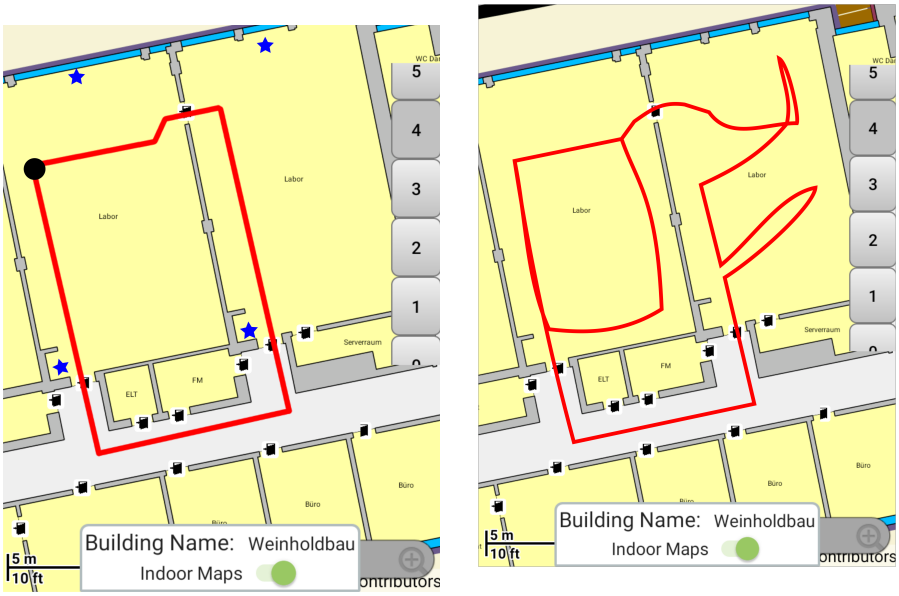
Table 2: Test Device Properties

Phone Model	Lenovo PB2-690M
Android OS	6.0.1
Accelerometer	Bosch Sensortec BMI160
Gyroscope	Bosch Sensortec BMI160
Magnetometer	Asahi Kasei Microdevices AK09911

Overall, we collected 3 different datasets, each having different route in 4^{th} floor's two office of the Weinholdbau Campus of TU Chemnitz, as it is given in Figure-5.1. The datasets' properties are the following:

- *Dataset 1*[9] has rectangle shape route and is completed in 258 steps by covering the same path 5 times.
- *Dataset 2* represents a more natural walking style and the route is completed in 425 steps by covering the same path 4 times.
- *Dataset 3* covers a bigger region comparing to other two ones. The route is completed in 926 steps by covering the same path 3 times.

[9]BLE beacons location are represented as blue stars on the figure.



(a) Dataset 1

(b) Dataset 2

(c) Dataset 3

Figure 5.1: Collected Datasets

Since any precise ground truth map information wasn't available for the indoor environment in which we collected our dataset, we build it with PDR algorithm. Although PDR has drift issues, it provides accurate positioning results for a short time. Note that, PDR algorithm that is used for the ground truth contains accelerometer and gyroscope sensor information for the ground truth. That's the reason we don't experience substantial drift. However, we don't have an absolute heading information unless the magnetometer isn't fused. Hence, we manually oriented the initial heading so that we have a global reference frame

that we perform comparisons. The other issue with the ground truth is that we don't know how much exactly the created ground truth deviates from the real walking path. However, the main goal of this paper is to find out how SLAM is able to recover the walking path shape.

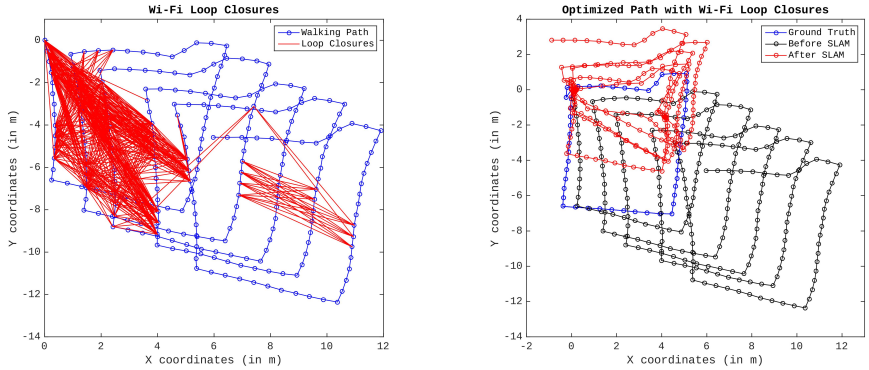
5.2 Wi-Fi Loop Closures

In the simulation environment, we assume that each step has at least one RSSI measurement from its surrounding access points. This is not a realistic assumption. The Lenovo PB2 devices provides 1-2 seconds Wi-Fi scan period, which is relatively fast Wi-Fi comparing to the other Android smartphones in the market, but it is still not enough for assumption for at least one RSSI measurement for each step.

By keeping these limitations in mind, we build the dataset as follows:

- Walking Path: Dataset 1
- IMU Scan Rate: 20 ms
- Wi-Fi Scan Rate: 1-2 ms (depends on the OS)
- Dissimilarity threshold: 5 dBm
- Temporal distance threshold[10]: 10 s

Based on the given setup, we experience loop closures that is shown in the Figure-5.2a



(a) Loop Closures (b) Optimized Path

Figure 5.2: Dataset 1 Optimization with Wi-Fi

Then, we run LM optimization algorithm and the resulting walking path configuration is shown in Figure-5.2b. Clearly, the optimal solution is not founded due to the lack true positive loop closures. The majority of the loop closures are false positives. Therefore, LM algorithm does not converge.

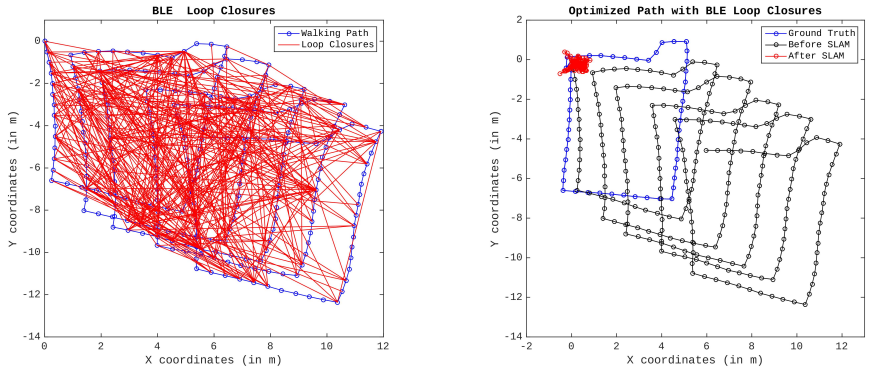
[10]When recognizing revisited places, consecutive steps' fingerprint information is highly correlated with each other. Therefore, we reject loop closures occurred within 10 s.

5.3 BLE Loop Closures

The other RSSI type measurement is collected by using BLE technology. The main difference between Wi-Fi and BLE in terms of SLAM is that BLE provides much higher scan rate, which could meet the assumption for at least one RSSI measurement for each step. However, the problem with the BLE is the deviation of RSSI measurement, which causes many false positive loop closures. In Figure-5.3a, the graph is almost covered by the loop closures due to the false positives.

Dataset setup for the BLE is as follows:

- Walking Path: Dataset 1
- IMU scan rate: 20 ms
- BLE scan rate: 200 ms
- Dissimilarity threshold: 1 dBm
- Temporal distance threshold: 10 s



(a) Loop Closures (b) Optimized Path

Figure 5.3: Dataset 1 Optimization with BLE

With the presence of great amount of false positives, we can see that LM does not converge to the optimal solution that we seek, as we see in Figure-5.3b.

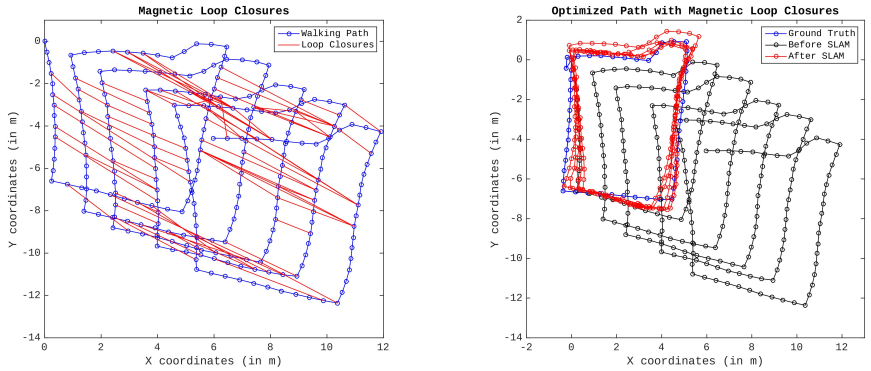
5.4 Magnetic Loop Closures

The last loop closure method is the magnetic field. None of the issues we had in RSSI does not appear in this case, as we can see in Figure-5.4a. Majority of the loop closure are true positives. The main reason for such a successful recognition of revisited places is that we store the 3-axis local magnetic field data based on the orientation of the device's body. If the user revisits any place again in future by holding the device at the similar position and orientation, the dissimilarity function is able to detect this behavior as a loop closure.

Magnetic field loop closure setup is as follows:

- Walking Path: Dataset 1,2 and 3

- IMU scan rate: 20ms
- Magnetometer scan rate: 20 ms (same magnetic field value that is used in IMU)
- Dissimilarity threshold: 1 µF
- Temporal distance threshold: 10 s

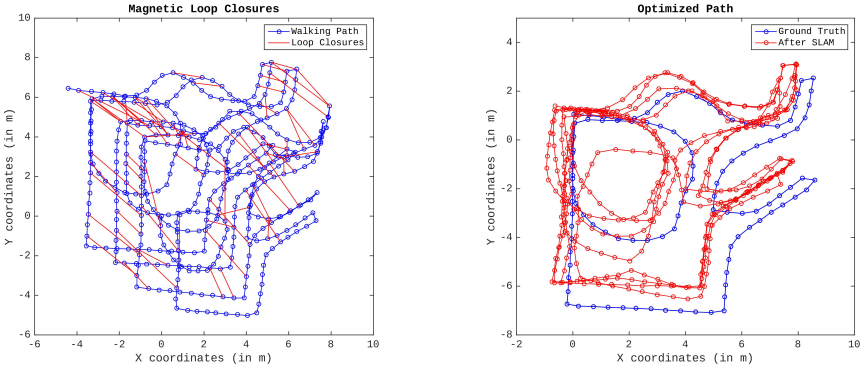


(a) Loop Closures (b) Optimized Path

Figure 5.4: Dataset 1 Optimization with Magnetic Field

The resulting optimized path by the magnetic field is shown in Figure-5.4b. SLAM achieves to converge to an optimal solution that is close to the ground truth.

Again with the Magnetic Field loop closures on Dataset 2, majority of the loop closure are true positives as it is seen in Figure-5.5a.

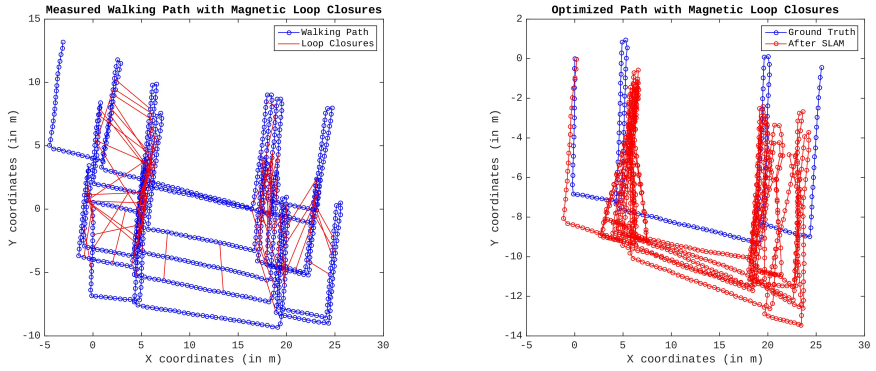


(a) Loop Closures (b) Optimized Path

Figure 5.5: Dataset 2 Optimization with Magnetic Field

In Figure-5.5b optimized path result is shown. In this dataset, we experience more deviations from the ground truth comparing to Dataset 1. However, the optimizer still manages to converge to an acceptable solution since the topology is kept mostly intact.

The third and last dataset that we collected expands to a bigger region by covering more rooms and the reference path of the walking route. Now, our algorithm faces the challenge against converging to an optimal path. Remember that our robust SLAM back-end deactivates the false positive loop closures because we assume that outliers could only occur when the distance between two loop closure point is too large. However, the assumption fails when we detect the false positive loop closures between room 463 and room 460, as it is shown in Figure-5.6a. At this point, the algorithm cannot reject these false positives because our threshold for rejecting false positive is 10 meters. Therefore, the resulting path is at these poses are distorted between these two rooms, as we can see in Figure-5.6b. If we, on the other hand, lowered the threshold to 4 meters, we could reject them. By doing so, we would also be rejecting most of the true positives as well and we would be able to converge at all. This is the trade-off that the algorithm faces.



(a) Loop Closures

(b) Optimized Path

Figure 5.6: Dataset 3 Optimization with Magnetic Field

5.5 Root Mean Square Error Evaluation

Finally, we provide the statistical error evaluation and the method, selected for the evaluation, is the *Root Mean Square Error (RMSE)*. This method measures the average deviation of the estimated positions from the ground truth positions. We only consider 2D in our experiments:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_{x,y}^{(i)} - \widehat{\boldsymbol{x}_{x,y}^{(i)}})^2} \tag{32}$$

where $\boldsymbol{x}_{x,y}^{(i)} = (x_i, y_i)$ are the ground truth positions and $\widehat{\boldsymbol{x}_{x,y}^{(i)}} = (\widehat{x}_i, \widehat{y}_i)$ are the estimated positions. Here is the comparison across all datasets that we collected so far:

Table 3: RMSE Comparison Between Datasets

Dataset	Loop Closure Type	RSME (x,y)	Convergence
1	BLE	(2.9278,3.8333)	No
1	Wi-Fi	(0.6863,2.8734)	No
1	Magnetic Field	(0.6835,0.6835)	Yes
2	Magnetic Field	(2.1992,2.1173)	Yes
3	Magnetic Field	(5.4413, 3.2115)	No

6 Conclusion

In this research paper, we studied SLAM problem to solve the indoor mapping problem. After stating our motivation for the paper, we shortly described how PDR algorithm works and identify its limitations. Since the goal of the paper is to recover PDR's walking path drifts by the Graph SLAM, we explained the general formulation. Then, we convert this formulation to such a form that one can utilize it with the smartphone. As a result of this study, we gained the following insights:

- Graph SLAM has a great potential to be a *recovery mechanism* against the drifted walking path caused by the smartphone's IMU.
- A general Graph SLAM formulation fails to converge to the optimal solution with the presence of false loop closures. Therefore, more *robust* version of SLAM is required to deal with the outliers.
- *Magnetic field* outperforms both Wi-Fi and BLE in terms of loop closure detection. However, the map builder individuals must walk by following such routes that they constantly revisit the place as frequent as possible. SLAM

7 Bibliography