BLE based Smartphone Localization

Zhengyang GU
Junior System Engineer, Valeo

November 7, 2017

Contents

List of Figures

1 Introduction

1.1 Bluetooth Low Energy

Bluetooth Low Energy devices operate in the 2.4 GHz free license-free band and so share the same propagation characteristics as 2.4 GHz WiFi transceivers(to see the frequency allocation in Fig.1). For WiFi, each access point uses a particular radio channel of width at least 20MHz. In contrast, BLE advertisements are broadcast in three much narrower(2MHz) channels. These channels are nominally labeled 37, 38 and 39 and are wildly spaced at 2402MHz, 2426MHz and 2480MHz respectively. These frequencies are chosen to minimize interference with common WiFi deployments. The advertising mode, permitted in the BLE standard, enables a very short message at a very flexible update rate. These messages can be used to allow device to detect close proximity to a specific location based on the Received Signal Strength Indicator(RSSI). In this way, location specific triggers and information can be provided to the user. In our case, we try to detect the location of the smartphone in respect to a vehicle based on the RSSI measured by the smartphone. In fact, we install 8 beacons(FRONTLEFT, FRONTRIGHT, REARLEFT, REARRIGHT, LEFT, MIDDLE, RIGHT, TRUNK) which operate in the mode advertising. The positions of these beacons can be found in Fig.2.

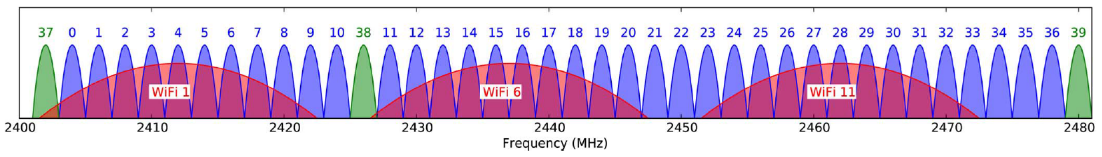

Figure 1: BLE channels and WiFi channels in 2.4 GHz

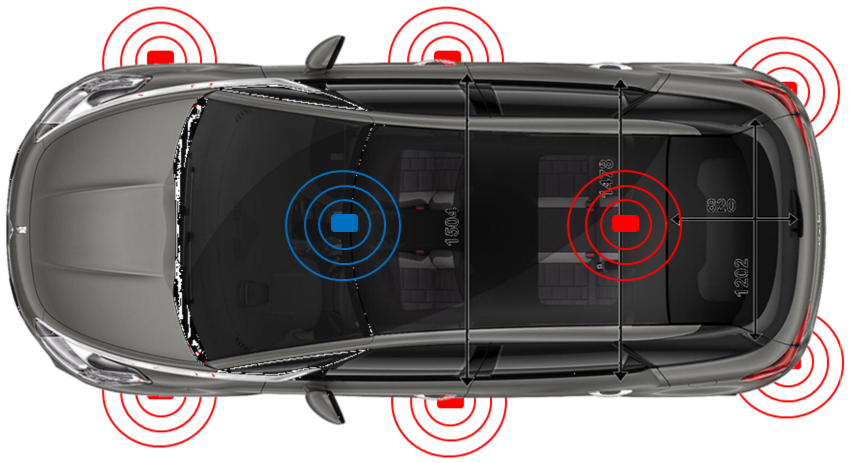

Figure 2: Position Configuration for 8 beacons

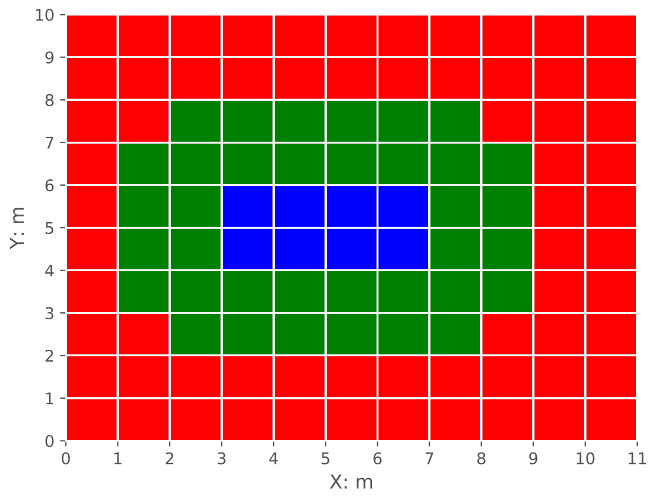

Figure 3: Location Map around a vehicle

1.2 Passive Entry Passive Start

The **Passive Entry Passive Start** system allows the driver to lock and unlock the vehicle's doors without touching a key, and to start or stop the engine simply by pushing the ignition button. The lock, unlock and start functions are based on the location of the user(suppose the user takes the smartphone with him). To be more specific, when the driver is far away from the vehicle(for example, more than 2 meters away), he can apply lock operation; when the driver is near the vehicle(less than 2 meters away), he can apply unlock operation; when the driver is inside of the vehicle, he can start the engine directly. Fig.3 shows a map of location around a vehicle. The red squares represent the lock zone while the green squares for the unlock zone and the blue squares for the start zone. Our objective is to find the exact zone in which stands the user in order to allow him the corresponding operation(lock, unlock or start).

1.3 Thatcham Constraint

As we can see, start vs no start and unlock vs lock are adversarial operation pairs. In order to secure the vehicle, we need to add constraints to the start and unlock operations. Thatcham standard requires that we can't start the engine as long as we are outside of a vehicle and we need to guarantee that we can never unlock the vehicle if we are far away from it(2 meters away). This document will focus mainly on the localization of the smartphone taking particularly the thatcham constraint into consideration.

2 Challenge

Before we talk about the detailed algorithm for the localization, let's look at some of the challenges that we can have in smartphone positioning with BLE. These challenges include the BLE signal characteristics(RSSI variation, uncertain channel offset, fast fading phenomenon), device measurement(orientation of smartphone, different marks of smartphone) and environment influence(obstacle attenuation, multipath or strong reflection) etc.

2.1 RSSI Variation

We found that the RSSI value measured by the smartphone is not very stable even thought we put the smartphone in a static position. Fig.4 shows an example of 4 RSSI values(LEFT, MIDDLE, RIGHT, TRUNK beacons) in the same channel in a static position of approximately 2 meters left from the vehicle) for 12 seconds. In terms of RSSI variation, we have 1.78dBm standard deviation for LEFT beacon, 3.54dBm for MIDDLE beacon, 3.43dBm for RIGHT beacon and 1.63dBm for TRUNK beacon.

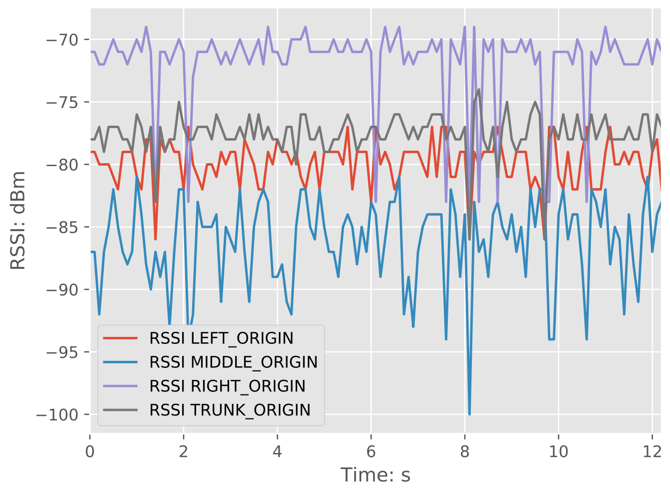

Figure 4: RSSI Variation over time in static position and same channel

2.2 Channel Offset

Section 1.1 shows that BLE advertisements are broadcast on three narrow advertising channels. These channels are nominally labeled 37, 38 and 39. By default, the smartphone scan each channel for nearly 5 seconds periodically. Fig.5 shows an example of 4 RSSI values(LEFT, MIDDLE, RIGHT, TRUNK beacons) in a static position of 1.5m left away scanning in three channels. As we can see, signal power doesn't stay the same for different channels. Two factors could cause this: uneven

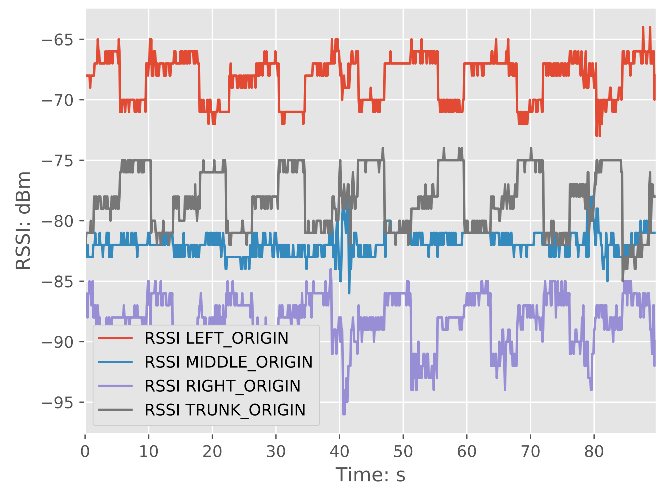

Figure 5: Channel Offset in a static position of 1.5 meters away

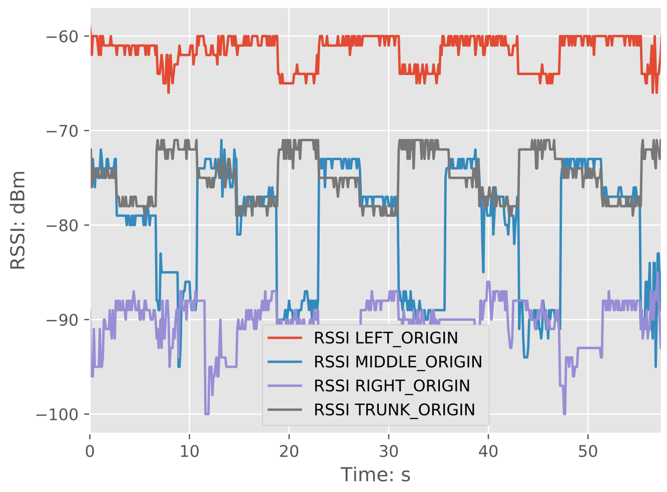

Figure 6: Channel Offset in a static position of 0.5 meters away

channel gain and multipath interference. The former occurs because the embedded antennas rarely have a flat response across the entire 2.4GHz band. The addiction of multipath interference causes further difference over time which can be deduced from the fact that the offset between channels are different for these 4 beacons. Fig.6 shows the result in another static position of 0.5m. It presents a different pattern of offset(to see the MIDDLE beacon) which proves again the existence of multipath interference.

As our localization algorithm is based only on the RSSI value, the channel offset will make the localization result no coherent over time if we don't process it. There are two possible solutions to remove or reduce the channel effect. The first is to enforce the smartphone to scan in the same channel. This can be done by modifying the scan configuration of the smartphone. By default, the smartphone scans channel 37, 38 and 39 in the order and about 5s for each channel. We can stop and restart regularly the BLE before it changes to another channel. For example, we can stop and restart the scan with a period of 4s or stop and restart the scan each time we find a channel 38. In this way, the smartphone remains scanning the channel 37. An alternative is to measure the RSSI value in 3 channels at the quasi same time and then use the average value to reduce the channel offset effect. However, we can't do this with Android System as Android System can't change the scan channel rapidly which turns out that these 3 channel values couldn't represent the same location of user. However, this could be done with iOS because iSO changes the scan channel at a rate of about 30ms which means that we can have all the 3 channel values each 100ms.

For the time being, we still apply the first strategy to remove the channel offset effect. However, regular stop and restart scan could cost a lot of battery. Another disadvantage is that the latest Android version(Android 7.0) doesn't support this stop and restart scan configuration. So we are obligated to search another strategy in the future.

2.3 Fast Fading Effect

Free space path loss(FSPL) is the loss of signal strength that would result from a line-of-sight path through a free space. FSPL is proportional to the square of distance between the transmitter and the receiver, and also proportional to the square of frequency. Eq.1 shows the formula of FSPL in decibels.

$$\mathbf{FSPL}(\text{dB}) = 20\log_{10}(\frac{4\pi f d}{c}) \tag{1}$$

where f is the frequency of signal, d is the distance between the transmitter and the receiver, c is the speed of light. In this way, the received power(RSSI) can be written as:

$$\mathbf{Pr} = \mathbf{Pt} - \mathbf{FSPL} = \mathbf{Pt} - 20\log_{10}(\frac{4\pi f d}{c}) \tag{2}$$

where **Pt** et **Pr** are transmitted power and received power respectively. Fig.7 shows the theoretical received power over distance according the Eq.2. We can deduce that theoretically, a noise of 6dB drop in the received power results in two times of the real distance.

However in experiment, the result can be very different. Fig.8 shows the experimental received power over distance. The smartphone is moved from 0.7m to 3m away from the beacon in a nearly open space. As we can see from the figure, deep multipath fades are evident with 5 dB drops in power across just 10cm of movement. This

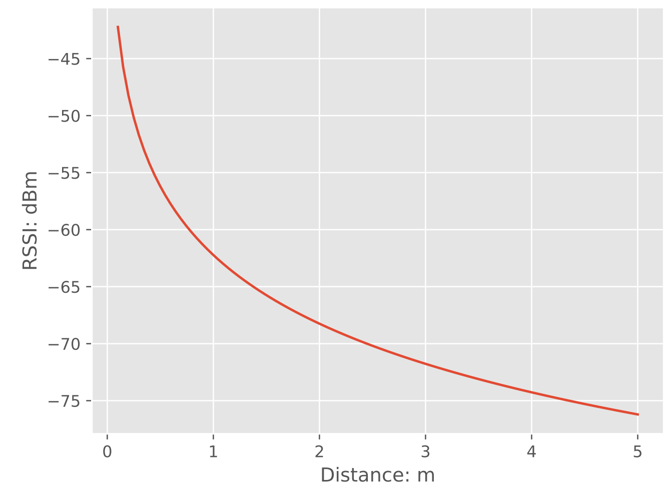

Figure 7: Theoretical Received Power over distance

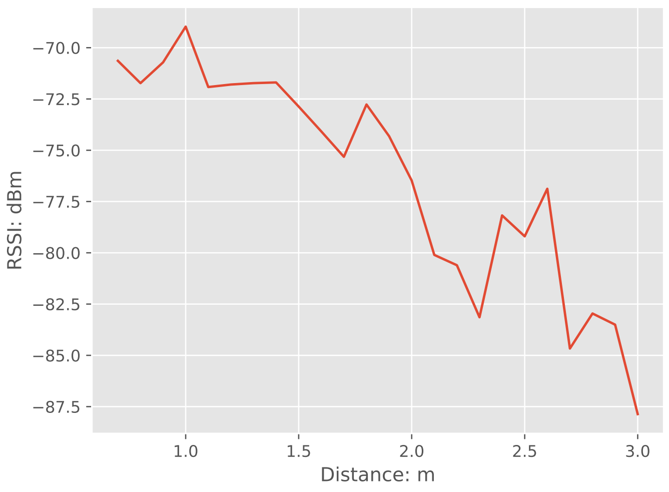

Figure 8: Experimental Received Power over distance

fast fading phenomenon can be more severe in an indoor environment such as in an underground park where the fade can be up to 30dB in across just 10cm of movement. In fact, the fast fading phenomenon is related to the bandwidth of signal. Larger the bandwidth of signal is, less severe the fast fading is. So, the fades are notably less severe for WiFi which uses 20MHz instead of 2MHz which is used for BLE.

From a signal fingerprinting perspective, deep fade presents a challenge since RSSI value can vary so dramatically over a spatial range that is smaller than the expected accuracy.

2.4 Orientation of Smartphone

As antenna of the smartphone is not isotropic, the RSSI value could vary greatly if we rotate the smartphone. Fig.9 shows the horizontal diagram of a smartphone. The measurement was taken in an anechoic chamber where the smartphone was put in the same distance(1.5m) from the transmitter and was rotated around z-axis 22.5° at each time(to see the rotation axis in Fig.10). As we can see, RSSI value can vary 15 dB when due to the direction of smartphone. Again, from a signal fingerprinting perspective, the smartphone direction information needs to be used to correct the signal power in order to have the same level of RSSI in all the directions.

In our experiment, we try to correct the signal by using the direction of smartphone relative to the beacon and a standard diagram of smartphone which is shown in Fig.9. During the experiment, the smartphone was put in the same distance of 2.5 meters away from the beacon and it was rotated manually around z-axis. Fig.11 shows the result of correction through the diagram. The figure in the top shows the original and corrected RSSI variation over time; the bottom left shows the distribution of the original and corrected signal; the bottom right shows the box figure of these two signals.

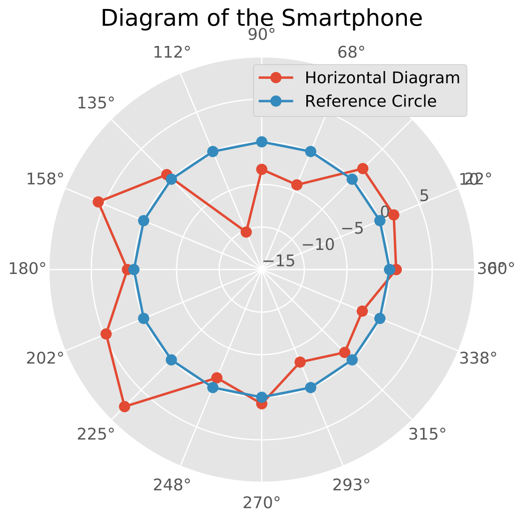

Figure 9: Experimental Horizontal Diagram of a smartphone

As we can see, the improvement of stability is not very evident using this compensation strategy by the diagram. There are two possible factors which could cause this. The first may be that the diagram was measured in an anechoic environment while the correction signal was taken in an outside environment with the beacon installed in a vehicle where multipath exists all the time. So the diagram measured in the anechoic chamber can't represent the variation of power in the real environment. Another possible reason is that the direction given by the smartphone is not precise especially

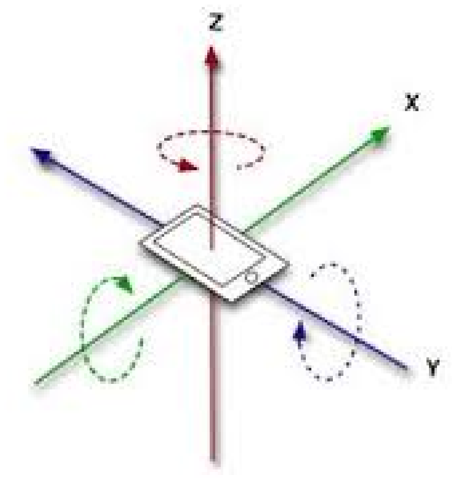

Figure 10: Rotation Axis of a smartphone

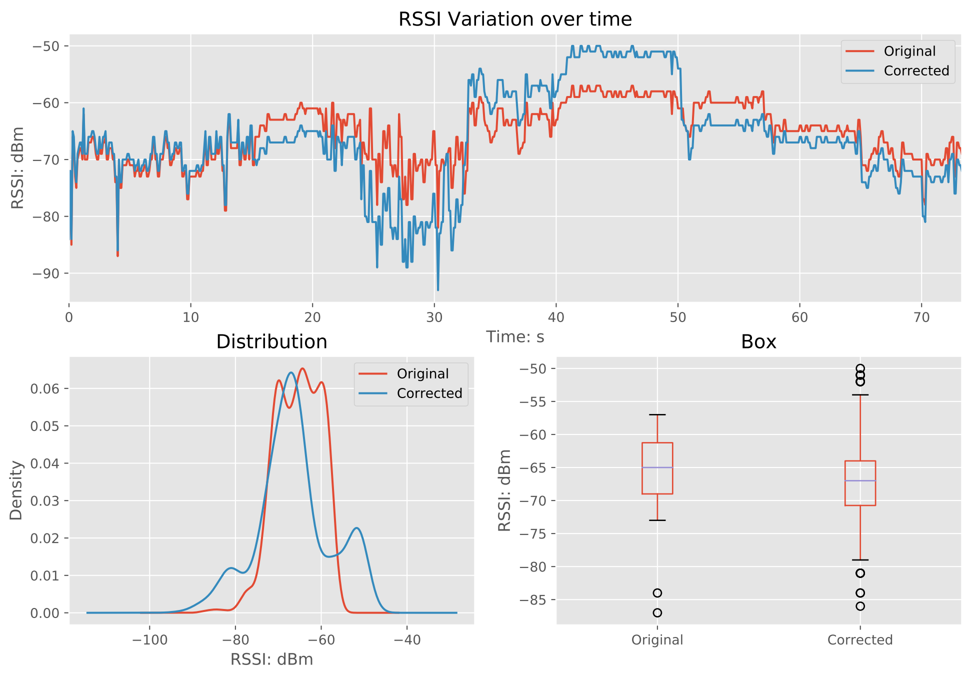

Figure 11: Correction of RSSI through the diagram

in an environment where there are many metals. So a more robust method needs to be applied in the future.

2.5 Other Influences

We have seen some principal challenges that we could encounter during the smartphone BLE localization. In fact, there are still some other challenges such as the smartphone model, body attenuation, inside or outside environment, WiFi interference and so on.

Fig.12 shows the influence of the smartphone model. As our localization is based only on the RSSI value, so we will not have the same result if the received power is not the same through different smartphone models. During the experiment, the two smartphones(Samsung A3 and A5) were put in the same position for 1 minute. However, the difference of these two signals can be 5dB. In order to allow the localization algorithm work for all the smartphones, a calibration step needs to be implemented. This can be done by first instructing the user to put the smartphone in the same position of a vehicle for a certain time and comparing the average RSSI value with the reference and then adding this smartphone offset for all the latter measurements.

Fig.13 shows the influence of body attenuation. We know that the body contains a large amount of water which shares the same resonance frequency with the BLE. So the body obstacle can attenuate the signal power dramatically which is the frequent case when the user puts his smartphone in the back pocket. From the Fig.13, we see that the signal power can be attenuated by 15dB due to the body obstacle.

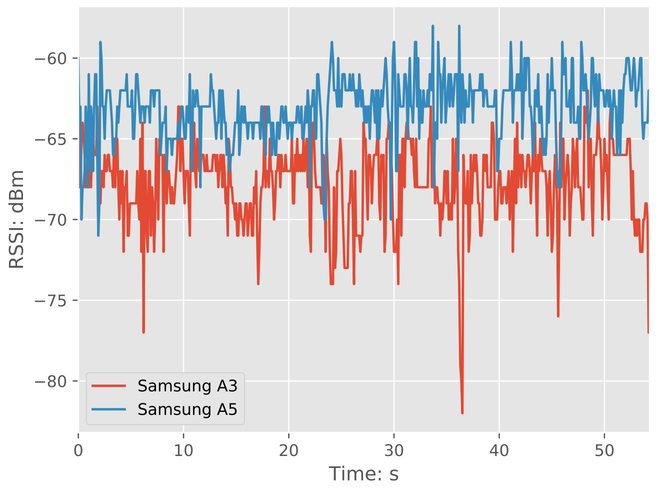

Figure 12: Influence of smartphone model

Fig.14 shows the influence of inside or outside environment. We have seen in section 2.3 that multipath can cause the signal power drop dramatically across just some

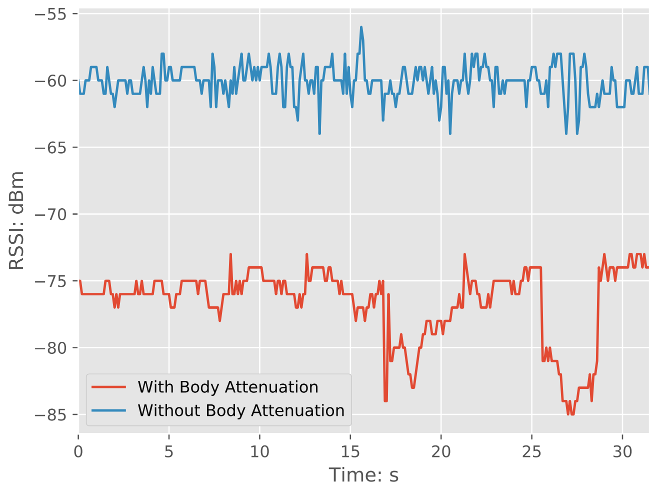

Figure 13: Influence of body attenuation

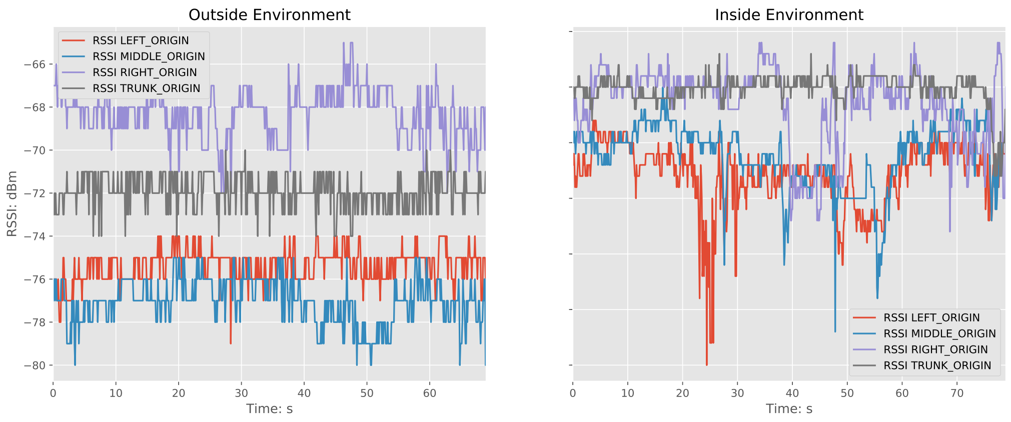

Figure 14: Influence of inside or outside environment

centimeters. This phenomenon can be worse in an inside environment. In our experiment, the smartphone was put in the right side of the vehicle and in the same distance from the beacon. As we can see from the figure, the RSSI doesn't have the same level of value in different environments. In an outside environment, it's normal to have a larger RSSI value for the beacon RIGHT than the beacon LEFT because the smartphone is relative more distant from the LEFT beacon. However, it's not the case in an inside environment where RSSI LEFT is much higher due to the multipath. Another observation is that a tiny change in an inside environment can cause the RSSI behave so differently. This can be deduced from the RSSI LEFT in an inside environment which shows a tremendous drop in 25 seconds due to the environment changes.

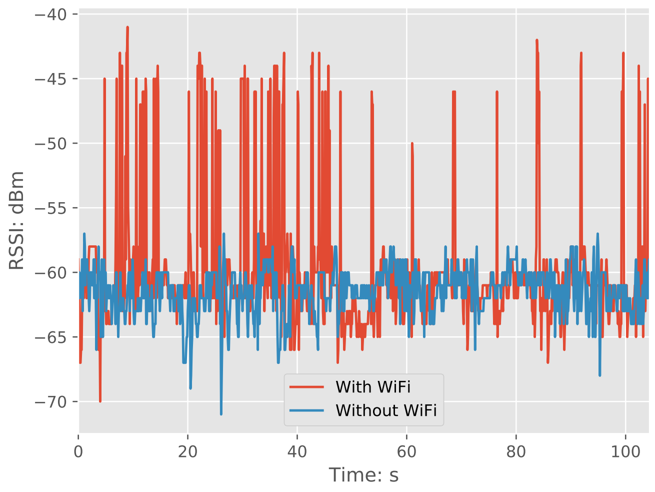

Figure 15: Influence of WiFi interference

Fig.15 shows the influence of WiFi interference. We have seen in Fig.1 that BLE and WiFi occupy the same 2.4 GHz band. WiFi could therefore have some interferences to BLE measurements. In our experiment, the smartphone was put in the same static position in different WiFi configurations. Fig.15 illustrates a strong noise in the RSSI value if we activate the WiFi of the smartphone. So ideally, we need to close WiFi of the smartphone in order to prevent this kind of interferences.

3 Algorithm

3.1 Fingerprinting

The basic idea of RSSI-based localization is to compare the measured RSSI value to a model of RSSI for each position and then identify the position that gives the best match. There are two kinds of RSSI models: *path loss model* and *power map model.* The *path loss model* models RSSI values as a decreasing function of transmitter-receiver distance. This method suffers from high noise and from multipath effects, particularly when there is no line of sight between transmitter and the receiver which is the usual case when we install the beacons in the vehicle. The *power map model* requires to know the RSSI value in a number of points scattered in the environment and save these values in a database. This can be done by applying an off-line measurement for all the positions. The set of RSSI values that are collected for each position of the map from various anchors(namely beacons for us) is called a fingerprint of that position. Besides, *power map* method is also known as *fingerprinting. Fingerprinting* methods usually provide better performances than *path loss model* based methods.

The most challenging aspect of the *fingerprinting* based methods is to formulate distance calculation that can measure the similarity between the observed RSSI and the known RSSI fingerprints. Machine Learning techniques can formulate that similarity and thus be applied to the location estimation problem. In this section, we will talk about leverage of Machine Learning techniques for zone prediction and coordinate prediction.

3.2 Prediction of Zone

Fig.3 shows the three zones that we need to classify: **start**, **access** and **lock**. The **start** defines all the areas inside of a vehicle including the trunk area; the **access** defines areas outside of a vehicle and until 2 meters away while **lock** defines areas which are more than 2 meters away. In fact, we can still divide the **access** zone into four small areas: **left**, **right**, **front** and **back**, which allows us to know exact direction of user when he is in the **access** zone. This section will focus on three different zones and we will talk about Machine Learning based algorithm for localization.

3.2.1 Measurement

Before we apply the Machine Learning methods, we need to prepare the data set. There are mainly two ways to construct the data set for us. The first is a dynamic measurement while the other is a static measurement. Dynamic measurement is to take the smartphone in the hand and walk slowly in each zone for some time(for example, 4 minutes for each zone). The data set can also be augmented by demanding different users do the same measurement. This moving nature of this measurement is important as it simulates the real user case and introduces some noises in the RSSI value which can make Machine Learning model more robust. Multi-user measurement is also crucial as it allows us to know the exact performance of model using so called separated train and test set, which means we use some users' data sets to train a model and test the performance in another user's data set(in our case, we use three first users' measurements as the train set and the last user's measurement as the test set).

Static measurement is to take the measurements while the smartphone is put in a static position. Fig. 16 shows all the points where we need to measure. In each point, we put the smartphone in face of the vehicle and measure RSSI for 15s. Static measurements is better for the production phase as it can guarantee the nearly same results for each data set construction while dynamic measurement can vary from one person to another. However, in this section, the data set was constructed in a dynamic manner as it allows us to evaluate the performance of Machine Learning model and then choose the best model based on the performance on the test set.

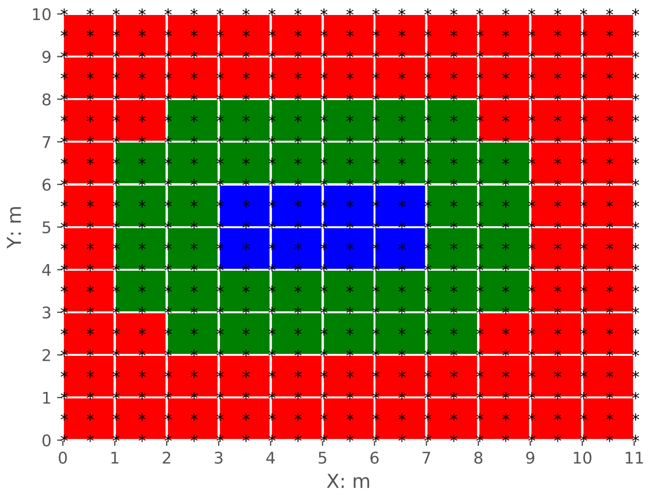

Figure 16: Point Measurement for date set construction

In our measurement, we have 8 RSSI values which correspond to 8 beacons for each position. We call these 8 RSSI values a feature vector or a fingerprint to be specifically. We give those fingerprints of a same zone the same label. In this way, we have three labels in total(**start**, **access** or **lock**). Before applying a Machine Learning model, we need to see how these fingerprints vary in different zones. In order to visualize these fingerprints in a figure, we need to reduce the dimension of each fingerprint while keeping the maximum infos. There are mainly two algorithms which can do this.

The first is PCA(Principal Component Analysis). PCA is a statical procedure that uses an orthogonal transformation to project each observation(sample) in some principal components. The transformation is defined in such a way that the first principal component has the largest possible variance and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. Fig.17 shows the result of PCA in a 3D scatter plot. We can see from the figure that red and green fingerprints are highly mixed which signifies the difficulty in distinguishing between **access** and **lock** zone.

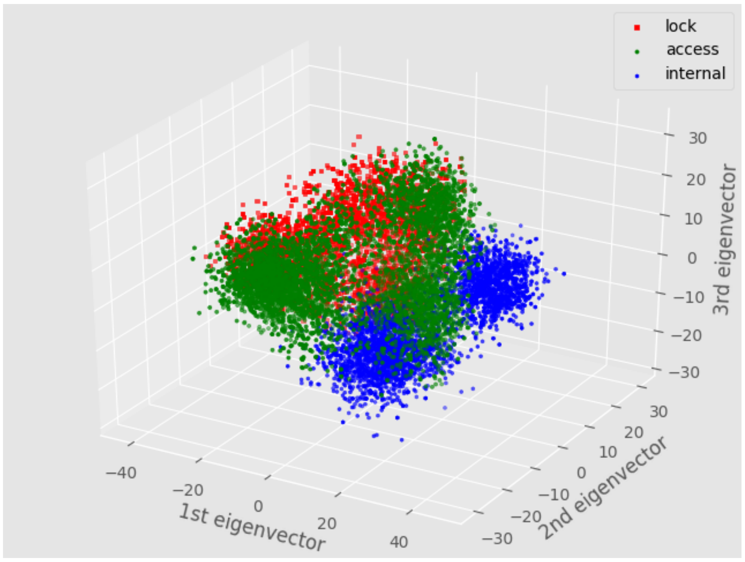

Figure 17: Representation of data with PCA

Another algorithm of dimensionality reduction is t-SNE(t-distributed Stochastic Neighbor Embedding). The t-SNE is a nonlinear dimensionality reduction technique that is particularly suited for embedding a high-dimensional data into a space of two or three dimensions. The t-SNE algorithm compromises two stages. First, t-SNE constructs a probability distribution for each pair of samples in such a way that similar samples have higher probability while dissimilar samples have lower probability. Second, t-SNE defines a similar probability distribution in the low dimensional map(two or three dimensions), and it minimizes the KL divergence between these two distributions. Fig.18 shows the result of t-SNE in a 2D scatter plot. Again, we find that **start** fingerprints and **access** fingerprints are well separated while **lock** fingerprints and **access** fingerprints are a bit of mixed. This mixture of two zones present some difficulties for Machine Learning algorithms, as the object of Machine Learning is to find a function of boundary which separates the data.

3.2.2 Training Result

In our experiment, we use three first users' measurements to train different Machine Learning models such as Logistic, LDA(Linear Discriminant Analysis),

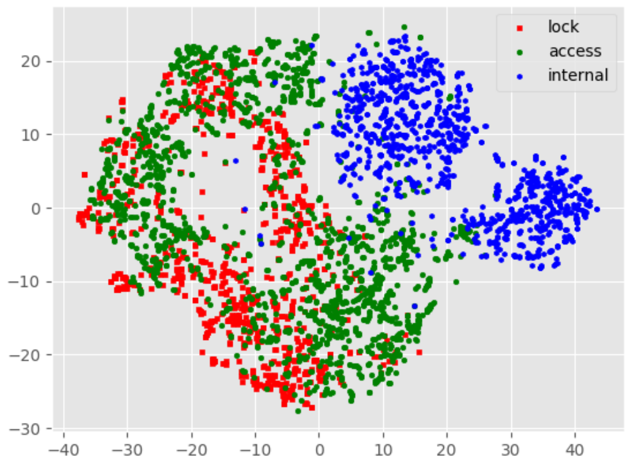

Figure 18: Representation of data with t-SNE

QDA(Quadratic Discriminant Analysis), SVM(Support Vector Machine), RF(Random Forest), GBM(Gradient Boosting), MLP(Multi Layer Perceptron) and then evaluate the performance in a separated test set. The performance is evaluated in terms of precision, recall and f1-score. Precision(P) is defined as the number of true positives(T_p) over the number of true positives(T_p) plus the number of false positives. Recall(R) is defined as the number of true positives(T_p) over the number of true positives(T_p) plus the number of false negatives(F_n):

$$P = \frac{T_p}{T_p + F_p}, \quad R = \frac{T_p}{T_p + F_n} \tag{3}$$

A model with high recall but low precision returns many results, but most of its predicted labels are incorrect compared to real labels; a model with high precision but low recall is just the opposite, returning very few results. So, precision or Recall alone is not sufficient to evaluate the performance. An alternative is to use the value of f1-score (F_1) which is defined as the harmonic mean of precision and recall:

$$F_1 = 2 * \frac{P * R}{P + R} \tag{4}$$

Fig.19 shows the performance result for all the methods. We can see that RF gives the best performance which has a value of 0.85 for f1-score. Fig.20 shows the details of confusion matrix. We can find that the model predicts well for the **internal** fingerprints while predicting badly when the real label is **lock**. The result is coherent to data representations(to see Fig.17 and Fig.18) where the cloud for **internal** is well separated whilst the cloud for **lock** is mixed with **access**. This also means that we can't have 100% separation between **access** and **lock**. What we can do is to guarantee the false positives in sacrifice of reducing the true positives for **access**. In this way, we guarantee the Thatcham constraint to some extent.

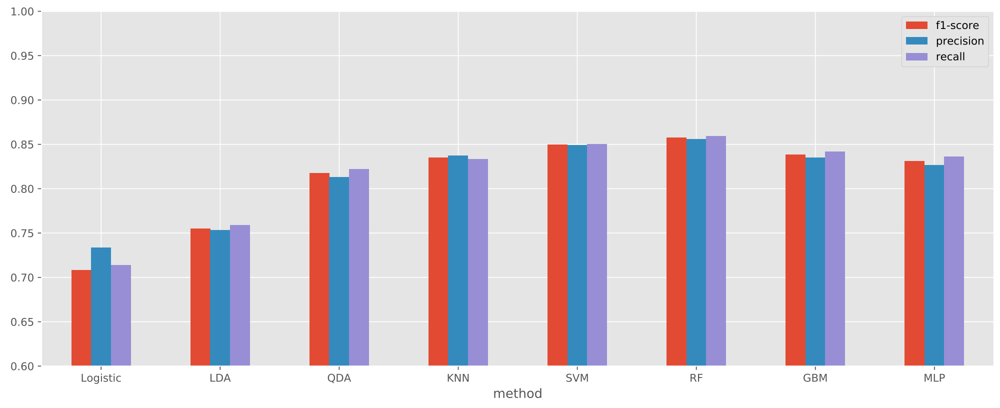

Figure 19: Performance Comparison in validation set

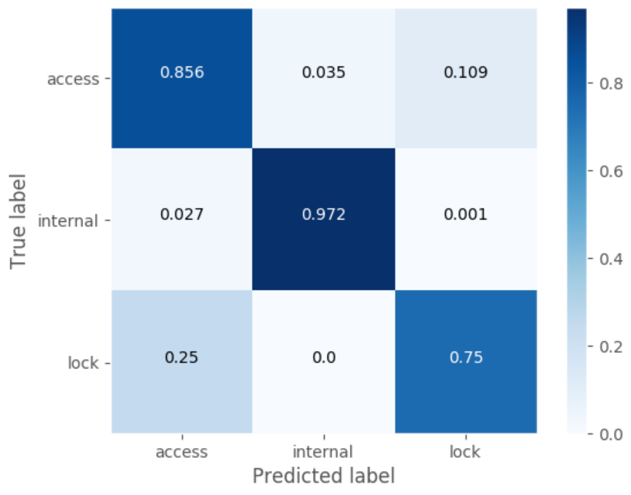

Figure 20: Confusion Matrix result of Random Forest

3.2.3 Stability Improvements

We have seen in the previous section that there exists a compromise between **access** and **lock**. In this section, we will focus on some strategies that will make the system more stable and most importantly to be more compliant with Thatcham. These strategies can be divided into two categories: preprocessing and post-processing. The preprocessing(RSSI Filter and Hysteresis) is to filter the RSSI values before applying the Machine Learning model while the post-precessing(Majority Vote and Probability Threshold) is to affect the final label after applying the model.

- RSSI Filter
  We have seen in Fig.4 that the RSSI value varies even though in a static position. This variation can be even worse when the user is in movement. Fig.21(red line)

shows an example of RSSI variation during the movement. The drop of RSSI value can be 15dB between two successive sampling times due to the movement. We can apply a unilateral limiter to prevent RSSI from dropping so fast. To be specific:

$$\tilde{x}(n) = \begin{cases} x(n), & x(n) - x(n-1) >= -1 \\ x(n-1) - 1, & x(n) - x(n-1) < -1 \end{cases} \tag{5}$$

where x and $\tilde{x}$ are original and filtered RSSI respectively and n signifies the index of sample. In this way, we allow the maximum drop of 1dB between two successive samples. Likely, we can filter the RSSI in a bilateral manner:

$$\tilde{x}(n) = \begin{cases} x(n-1) + 1, & x(n) - x(n-1) >= 1 \\ x(n-1) - 1, & x(n) - x(n-1) < -1 \\ x(n), & \text{else} \end{cases} \tag{6}$$

However, we found that filtering RSSI in bilateral manner can make the RSSI values too smooth and in turn increases the response time. In order to have a response time less than 0.5s, we choose unilateral limiter for our algorithm. Fig.21 shows an example of applying unilateral filter. The blue line illustrates the improvements of smoothness in RSSI variation. Fig.22 shows the improvements of performance by applying the preprocessing: we have 3% of improvements in terms of f1-score.

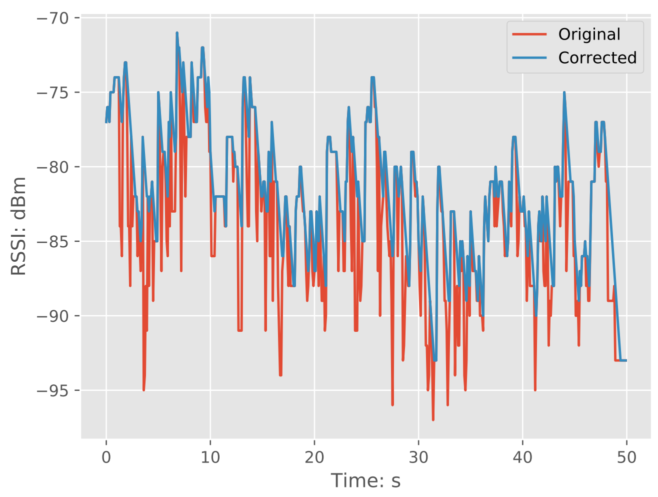

Figure 21: Preprocessing of RSSI values

- Majority Vote

  In an Andoird System, each fingerprint(sample) takes about 100 ms to be updated which means that we can have at least 4 or 5 successive samples in each 0.5s. During this period, the user is not possible to move a long distance(if we

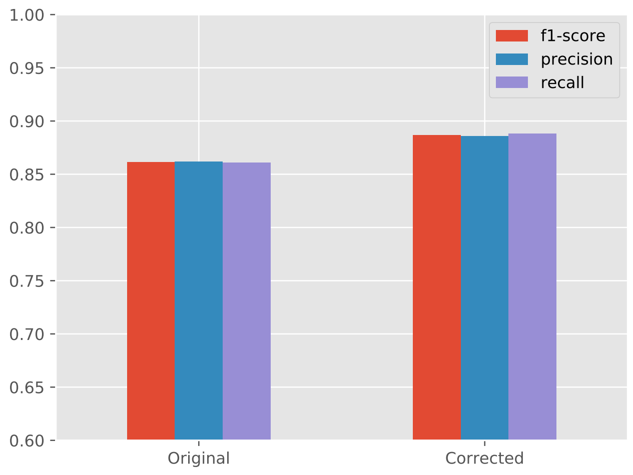

Figure 22: Performance Comparison with preprocessing

take a typical speed of 5km/h for example, the user can displace 0.7m in 0.5s), so we can logically make the hypothesis that the user is still in the same zone during this time. That's why we can take a majority vote to all the predicted labels as the final affected label. As such, majority vote allows us to remove the sudden wrong label due to the movement noise.

- Probability Threshold
  Random Forest model can give us not only the final label, but also the probabilities of each dominant class(in our case, the probabilities for **internal**, **access** and **lock**). The final label is affected to the class who has the largest probability. This means the final label can possibly be wrong if we have two probabilities that are close(for example, 0.45 for one class, 0.5 for another and 0.05 for the rest one). Fig.23 shows the distribution of max probability in correctly labeled and incorrectly labeled cases. We can see from the figure that we tend to have a larger max probability for the correctly labeled cases while having a relatively smaller max probability(around 0.5) for the incorrectly labeled cases. In order to have a more stable result, we can apply a threshold comparison before the predicted label changes:

$$\tilde{y}(n) = \begin{cases} y(n), & P_{y=y(n)} >= \sigma \\ \tilde{y}(n-1), & P_{y=y(n)} < \sigma \end{cases} \tag{7}$$

where y and $\tilde{y}$ are predicted and final label respectively, $P_{y=y(n)}$ is the probability corresponding to $y(n)$ and σ is the probability threshold.

In addition, we can choose different values of σ for each change of labels: from **lock** to **access**, from **access** to **lock**, etc. Taking thatcham into consideration, we choose a larger threshold for the passage from **lock** to **access** and a relatively smaller threshold for the passage from **access** to **lock**. In the current version of algorithm, we have:

$$\begin{cases} \sigma_{\textbf{access}\rightarrow\textbf{lock}} = 0.5 \\ \sigma_{\textbf{lock}\rightarrow\textbf{access}} = 0.8 \\ \sigma_{\text{others}} = 0.6 \end{cases} \tag{8}$$

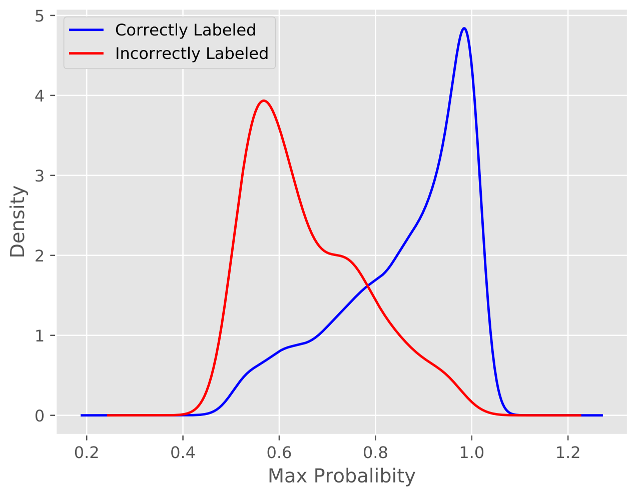

Figure 23: Distribution of Max Probability

- Hysteresis
  Like the way we use a threshold comparison in the case of change of label, we can also add a hysteresis δ to the filtered RSSI $\tilde{x}(n)$. The hysteresis δ is a function of last affected label $\tilde{y}(n-1)$:

$$\tilde{\tilde{x}}(n) = \tilde{x}(n) + \delta(\tilde{y}(n-1)) \tag{9}$$

  The result of $\tilde{\tilde{x}}(n)$ is then used as the entry for the Machine Learning model. We have found in our experiment that there is no need to add a hysteresis for each change of label. In the current algorithm, we apply the hysteresis in the case when the last affected label is **lock**:

$$\delta(\tilde{y}(n-1)) = \begin{cases} -2, & \tilde{y}(n-1) = \textbf{lock} \\ 0, & \tilde{y}(n-1) = \textbf{others} \end{cases} \tag{10}$$

3.3 Prediction of Coordinate

In the previous section, we have talked about how to predict location zone(**start**, **access** or **lock**) through Machine Learning Classification algorithm. However, the returned position is very coarse. In this section, we will focus on predicting the exact coordinate of user in relative to a vehicle using Machine Learning Regression algorithm.

The data set in this section was constructed using point measurement as shown in Fig.16. The data was collected in a map of 11m $\times$ 10m(not including the inside vehicle region) where the distance between the grid point is 0.5m. At each position($\mathbf{P}_i = [x_i, y_i]^T$), the RSSI value($\mathbf{R}_i$) was collected from all the beacons for 15s:

$$\mathbf{R}_i = [\mathrm{R}_{i1}, \mathrm{R}_{i2}, \mathrm{R}_{i3}, \mathrm{R}_{i4}, \mathrm{R}_{i5}, \mathrm{R}_{i6}, \mathrm{R}_{i7}, \mathrm{R}_{i8}]^T$$

where i denotes the i^{th} sampling result and $1, 2, \cdots 8$ denotes eight beacons respectively. The structure of data, which is a matrix containing the RSSI values and the corresponding coordinates is given below:

$$\text{data} = \begin{bmatrix} \mathrm{R}_{i1} & \mathrm{R}_{i2} & \mathrm{R}_{i3} & \mathrm{R}_{i4} & \mathrm{R}_{i5} & \mathrm{R}_{i6} & \mathrm{R}_{i7} & \mathrm{R}_{i8} & x_1 & y_1 \\ \mathrm{R}_{21} & \mathrm{R}_{22} & \mathrm{R}_{23} & \mathrm{R}_{24} & \mathrm{R}_{25} & \mathrm{R}_{26} & \mathrm{R}_{27} & \mathrm{R}_{28} & x_2 & y_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ \mathrm{R}_{n1} & \mathrm{R}_{n2} & \mathrm{R}_{n3} & \mathrm{R}_{n4} & \mathrm{R}_{n5} & \mathrm{R}_{n6} & \mathrm{R}_{n7} & \mathrm{R}_{n8} & x_n & y_n \end{bmatrix} \tag{11}$$

With this data set, we can see how RSSI varies spatially for each beacon. Fig.24 is an example which shows the RSSI variation across the space from the same MIDDLE beacon. There exists some kind of pattern between the position and the RSSI and this pattern can be modeled as a function. We will apply Neural Network to find this function Φ which maps the RSSI vector($\mathbf{R}$) and the position vector($\mathbf{P} = [x, y]^T$):$\mathbf{P} = \Phi(\mathbf{R})$.

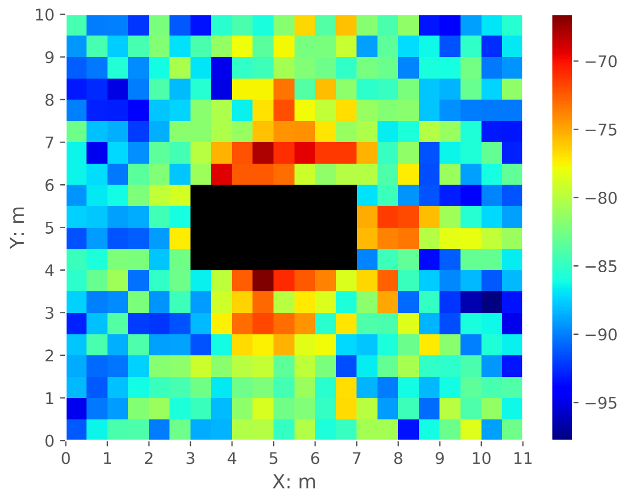

Figure 24: Heatmap for MIDDLE Beacon

3.3.1 Multilayer Perceptron based Object Tracking

Neural Network is a strong tool which can model any function between the inputs and the outputs. In this section, we will use a Multilayer Perceptron neural network which is a feed-forward neural network consisting an input layer, some hidden layers and an output layer. Fig.25 shows a 3-4-4-1 MLP neural network which has 3 input nodes, 4 nodes in the first and second hidden layer and 1 output node. In our case, we use 2 separated 8-64-64-1 neural networks for coordinate x and coordinate y respectively. In a MLP, two successive layers are connected by a weight matrix and a bias vector. The weight matrix and bias vector are updated iteratively so that the calculated output is as close as possible to the real output. The difference of these two outputs can be defined using a loss function. For example, the loss function of coordinate x neural network can be written as MSE(Mean square Error):

$$\text{loss} = \text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2 \tag{12}$$

where $\hat{x}_i$ is estimated coordinate and x is the real output.

In order to train a MLP, the data set is divided randomly into two separate sets: train set(75%) and valid set(25%). The train set is used to find the best parameters(wight matrix and bias vector) of the MLP while the valid set allows us to see the quality of the corresponding parameters. Fig.26 and Fig.27 show the training result for each coordinate: P_x and P_y. The quality(metric) is measured using RMSE:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2} \tag{13}$$

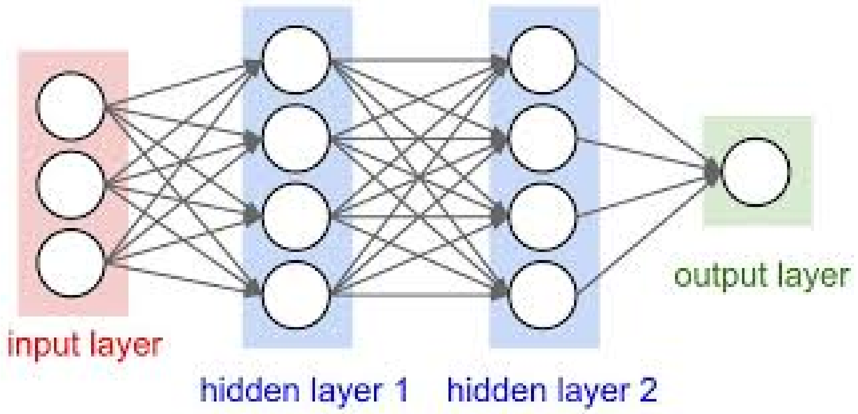

Figure 25: MLP Neural Network Architecture

The RMSE metric can be considered as the average error for our prediction. As we can see from these two figures, we have an error of 1.2m for P_x and 0.65m for P_y,

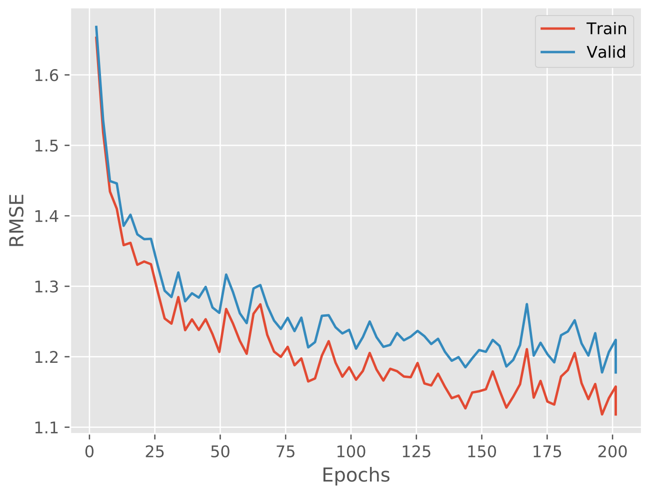

Figure 26: MLP result for P_x coordinate

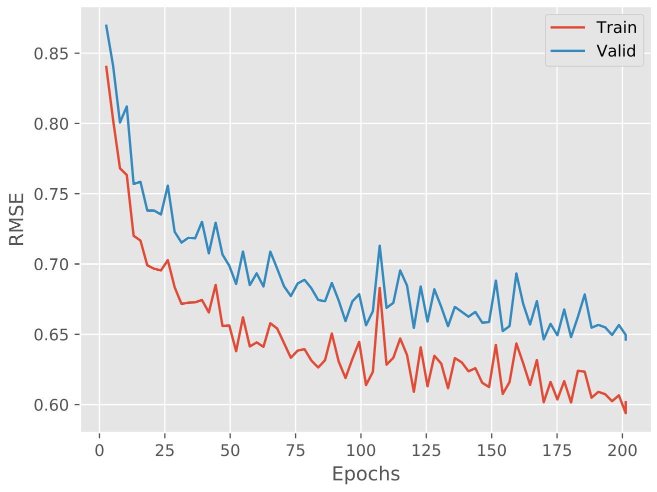

Figure 27: MLP result for P_y coordinate

which gives us a total error of about $\sqrt{1.2^2 + 0.65^2} = 1.36$ meter.

The error can still be reduced by applying a filter to the coordinate. A simple filter is to limit the distance between two successive coordinate. Fig.28 illustrates the filter for the case when the distance between two successive coordinates is greater than a certain threshold. In the figure, $\mathbf{P}_{n-1}$ and $\mathbf{P}_n$ denote the estimated coordinate for the time $n-1$ and n respectively, $\tilde{\mathbf{P}}_n$ denotes the filtered coordinate. If the distance between two successive time exceeds a certain threshold Δ_d, the filtered coordinate is chosen to be the point which has a displacement of Δ_d and lies in the same estimated direction $\overrightarrow{\mathbf{P}_{n-1}\mathbf{P}_n}$. The threshold Δ_d can be chosen according to the user speed. For

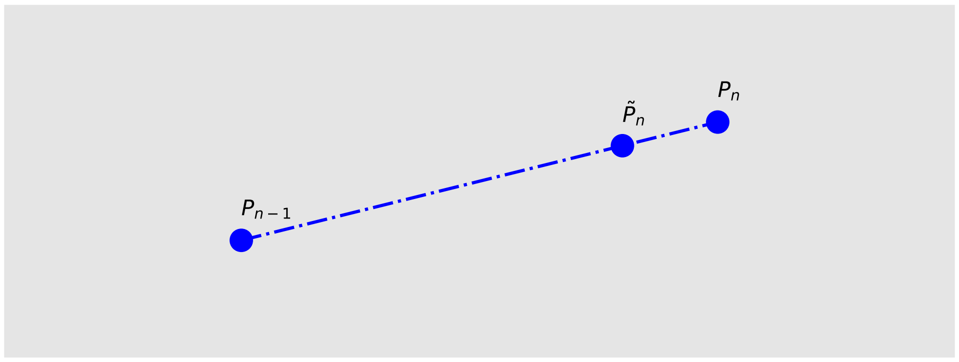

Figure 28: Distance Limited Filter

example, given the user moves with a speed of 1.5m/s, the displacement between two successive time(100ms) is 0.15m. In this way, the distance-limited filter can be written as follows:

$$\overrightarrow{\mathbf{O}\tilde{\mathbf{P}}}_n = \begin{cases} \overrightarrow{\mathbf{OP}}_n, & \|\overrightarrow{\mathbf{P}_{n-1}\mathbf{P}_n}\| <= \Delta_d \\ \overrightarrow{\mathbf{OP}}_n * \frac{\Delta_d}{\|\overrightarrow{\mathbf{OP}}_n\|}, & \|\overrightarrow{\mathbf{P}_{n-1}\mathbf{P}_n}\| > \Delta_d \end{cases} \tag{14}$$

where $\mathbf{O}$ denotes the original coordinate $(0, 0)$ and $\Delta_d = 0.15$.

3.3.2 Kalman Filter Smoother

In the previous section, we have used a distance-limited filter to reduce the coordinate error. In this section, we will focus on an another filter: kalman filter. The kalman filter provides the minimum mean square error solution to a linear system problem when the process under observation is completely represented by the state model. We will compare two typical state models: constant velocity(CV) model and constant acceleration(CA) model.

In a CV model, the object is considered to move in a constant velocity while the acceleration component is considered to be the noise. The system model can be written as follows:

$$\begin{aligned} x_k &= x_{k-1} + v_{x,k-1}\Delta t + \tfrac{1}{2}a_x\Delta t^2 \\ v_{x,k} &= v_{x,k-1} + a_x\Delta t \\ y_k &= y_{k-1} + v_{y,k-1}\Delta t + \tfrac{1}{2}a_y\Delta t^2 \\ v_{y,k} &= v_{y,k-1} + a_y\Delta t \end{aligned} \tag{15}$$

where a_x and a_y are considered to be noise: $\mathbf{q}_k = [a_x, a_y]^T$. The state vector for the CV model is defined as: $\mathbf{x}_k = [x_k, v_{x,k}, y_k, v_{y,k}]^T$. In terms of measurement, we can use the estimated coordinate $[P_x, P_y]$ provided by MLP in the last section: $\mathbf{y}_k = [P_x, P_y]^T$. In this way, the CV model can be represented as follows:

$$\begin{aligned} \mathbf{x}_k &= \mathbf{F}\mathbf{x}_{k-1} + \mathbf{G}\mathbf{q}_k \\ \mathbf{y}_k &= \mathbf{H}\mathbf{x}_k + \mathbf{r}_k \end{aligned} \tag{16}$$

where:

- $\mathbf{x}_k$: the state vector
- $\mathbf{F}$: the transition matrix
- $\mathbf{G}$: the process noise matrix
- $\mathbf{q}_k$: the process noise vector
- $\mathbf{y}_k$: the measurement vector
- $\mathbf{H}$: the measurement matrix
- $\mathbf{r}_k$: the measurement noise

with:

$$\mathbf{F} = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{G} = \begin{bmatrix} \frac{1}{2}\Delta t^2 & 0 \\ \Delta t & 0 \\ 0 & \frac{1}{2}\Delta t^2 \\ 0 & \Delta t \end{bmatrix}$$

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{17}$$

In a CA model, the object is considered to move in a constant acceleration with an acceleration noise: $\mathbf{q}_k = [\Delta a_x, \Delta a_y]^T$. The state vector for CA model is defined as: $\mathbf{x}_k = [x_k, v_{x,k}, a_{x,k}, y_k, v_{y,k}, a_{y,k}]^T$. The measurement vector is constructed by combining the coordinate and the measured acceleration by smartphone: $\mathbf{y}_k = [P_x, a_x, P_y, a_y]^T$. Likely, the CA model can be written as follows:

$$\begin{aligned} x_k &= x_{k-1} + v_{x,k-1}\Delta t + \tfrac{1}{2}a_{x,k-1}\Delta t^2 \\ v_{x,k} &= v_{x,k-1} + a_{x,k-1}\Delta t \\ a_{x,k} &= a_{x,k-1} + \Delta a_x \\ y_k &= y_{k-1} + v_{y,k-1}\Delta t + \tfrac{1}{2}a_{y,k-1}\Delta t^2 \\ v_{y,k} &= v_{y,k-1} + a_{y,k-1}\Delta t \\ a_{y,k} &= a_{y,k-1} + \Delta a_y \end{aligned} \tag{18}$$

This can also be reformed as Eq.16 with different transition, process noise and measurement matrix:

$$\mathbf{F} = \begin{bmatrix} 1 & \Delta t & \frac{1}{2}\Delta t^2 & 0 & 0 & 0 \\ 0 & 1 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta t & \frac{1}{2}\Delta t^2 \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{G} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{19}$$

In both models(CV and CA), the Kalman filter process has two steps: the prediction step, where the next state of the system is predicted given the previous measurement, and the update step, where the current state of the system is estimated given the measurement at that time step. These two steps can be written as follows:

- Prediction step:

$$\begin{aligned} \mathbf{x}_k^- &= \mathbf{F}\hat{\mathbf{x}}_{k-1} \\ \mathbf{P}_k^- &= \mathbf{F}\hat{\mathbf{P}}_{k-1}\mathbf{F}^T + \mathbf{G}\mathbf{Q}\mathbf{G}^T \end{aligned} \tag{20}$$

- Update step:

$$\begin{aligned} \mathbf{K}_k &= \mathbf{P}_k^-\mathbf{H}^T(\mathbf{H}\mathbf{P}_k^-\mathbf{H}^T + \mathbf{R})^{-1} \\ \hat{\mathbf{x}}_k &= \mathbf{x}_k^- + \mathbf{K}_k(\mathbf{y}_k - \mathbf{H}\mathbf{x}_k^-) \\ \hat{\mathbf{P}}_k &= (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}_k^- \end{aligned} \tag{21}$$

where:

- $\mathbf{x}_k^-$: predicted state vector
- $\hat{\mathbf{x}}_k$: updated(estimated) state vector
- $\mathbf{P}_k^-$: error covariance of the predicted state vector
- $\hat{\mathbf{P}}_k$: error covariance of the updated state vector
- $\mathbf{K}_k$: Kalman filter gain
- $\mathbf{Q}$: process error covariance
- $\mathbf{R}$: measurement error covariance

In the actual implementation of the filter, the measurement error covariance $\mathbf{R}$ is usually measured prior to the operation of the filter. Measuring the measurement error covariance is generally possible. We can take some off-line samples in order to determine the variance of the measurement error. In our case, we can use the predicted coordinate error provided by MLP for calculating $\mathbf{R}$: we choose an identity matrix in the reality which has the same order with MLP error. The determination of the process error covariance $\mathbf{Q}$ is generally more difficult as we don't have the ability to directly observe the process that we are estimating. To simplify the algorithm, we choose a weighted identity matrix(for example, $\mathbf{Q} = 0.5 * \mathbf{I}_p$).

In our experiment, we try to compare the performance of these two Kalman filters in different scenarios using simulation. In each scenario, we add a gauss noise for every measurement. The performance is compared using RMSE. Tab.1 shows the comparison result. As we can see from the table, both CV and CA model reduce largely the error. Besides, CV model outperforms CA model slightly in a static or

Table 1: Comparison of Kalman Filter performance in different scenarios

Scenario	Description	Raw RMSE	CV RMSE	CA RMSE	Figure
Scenario 1	static	1.279	0.166	0.194	Fig.29
Scenario 2	$v_x = 2, v_y = 1$	1.267	0.200	0.225	Fig.30
Scenario 3	$v_x = 2, a_y = 1$	1.272	0.857	0.208	Fig.31
Scenario 4	random a_x, a_y	1.268	0.606	0.212	Fig.32

constant-velocity scenario while CA model outperforms CV model in a no-constant-velocity scenario. Despite the better performance in CA model during the simulation, we choose to implement the CV model in our algorithm finally. The reasons are as follow:

- CV model already improves a lot our performance.
- CA model requires to measure the acceleration in the vehicle's coordinate system which in turn needs both the direction of the vehicle and the smartphone. The complementary complicity can create some artifact errors.

In the simulation, CV model shows an error of 0.6m for a random-acceleration scenario. However, in a real use case, we don't achieve such performance as described in the simulation. The trajectory in a real situation using the kalman CV filter still contains lots of noise. Therefore, further research needs to be done in the future.

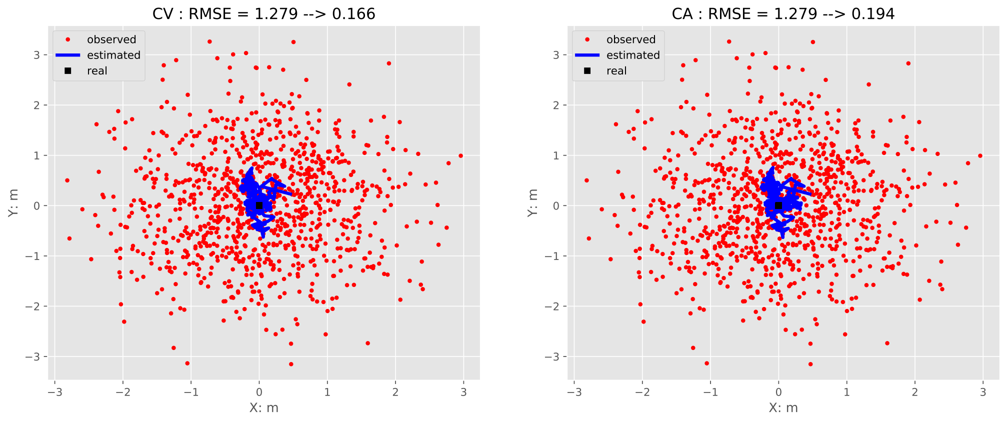

Figure 29: Comparison of CV and CA in a static case

3.4 IMU based Perspective

In the previous sections, we have used BLE for our localization algorithm. In fact, we could still use IMU sensor modules to increase the positioning accuracy. Such sensor modules contain three orthogonal accelerometers, three orthogonal magnetometers and three orthogonally mounted angular rate sensors. These sensor modules exist

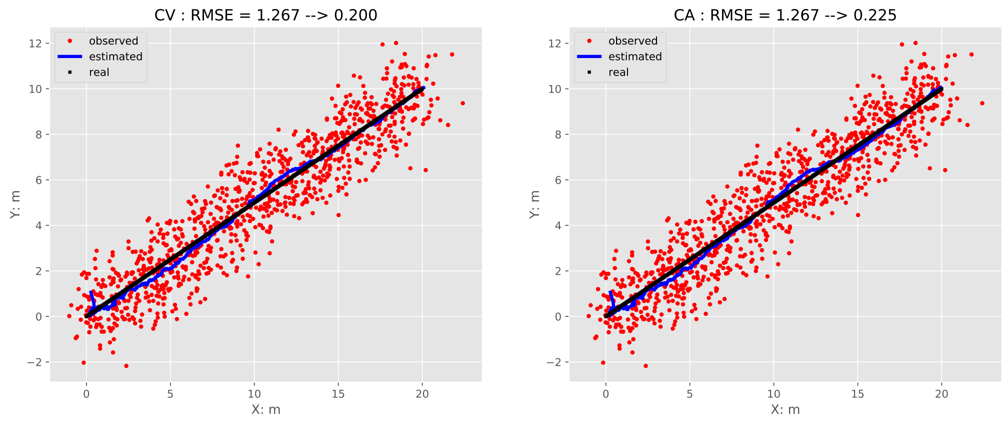

Figure 30: Comparison of CV and CA in a constant-velocity case

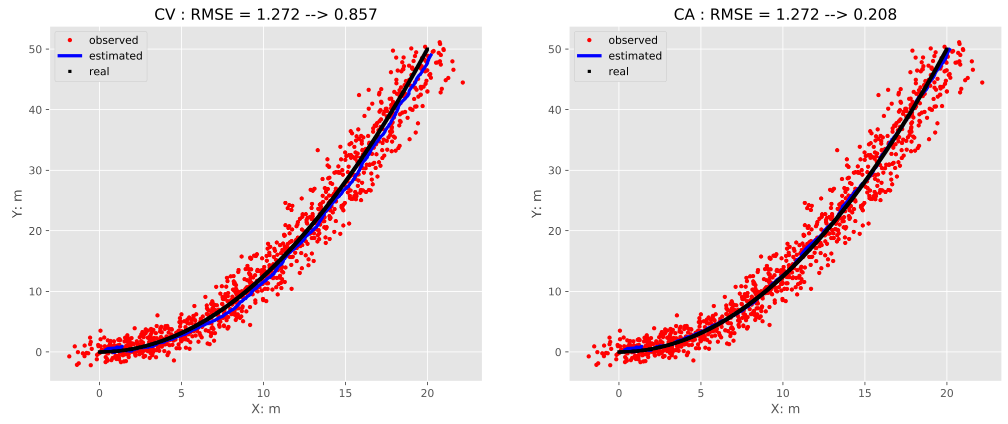

Figure 31: Comparison of CV and CA in a constant-acceleration case

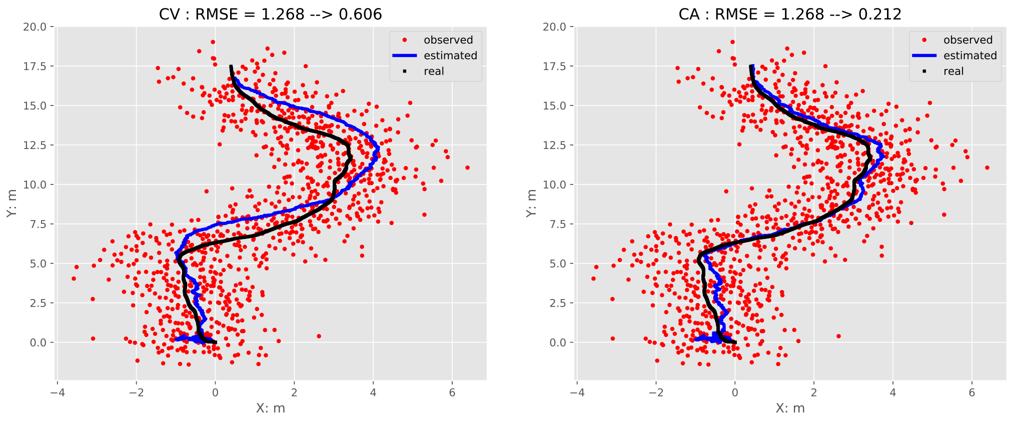

Figure 32: Comparison of CV and CA in a random-acceleration case

nearly in all of the smartphone. In this section, we will talk about the possible perspectives of using IMU for our localization algorithm.

In order to find the relative location of the user to the vehicle, we need to measure the location of user and the location of the vehicle in the same coordinate system respectively. There are mainly two coordinate systems: device coordinate(local coordinate system) and earth coordinate(global coordinate system). The device coordinate system is shown in Fig.10 which measures the device's rotation in relative to a device itself. The earth coordinate is usually defines as **NED**(**N**: north, **E**: east, **D**: downside) which measures the device's rotation in relative to the earth. IMU measurements use the device coordinate system which needs to be converted in a global coordinate system. Article[1] compares some typical algorithms for estimating a device rotation in a global coordinate system. The details of Madgwick Algorithms can be found in [2]. An alternative is to use Android's built-in conversion algorithm. In spite of the default algorithm provided by Android System, we still need to verify its accuracy of estimation.

The global rotation of the smartphone is crucial in using IMU for positioning as it allows us to project the accelerometers in the global coordinate system and then allows us to know the displacement of user in the global coordinate system. A double integration can be applied to the projected accelerometer to get the occurred displacement. However, as the IMU module in the smartphone is not of great quality, there exists some offsets in the accelerometer values. The accelerometer values need to be corrected in order to prevent the calculated distance from diverging. Article[3] presents an algorithm for correcting this offset. They are based on the fact that at each moment when a step occurs or finishes, the speed of that user is zero. These drift-corrected accelerometer values can then be used for a double integration to calculate the displacement. The most important part of this algorithm however is to detect accurately the moment when a step occurs or finishes. Article [3] .

References

[1] Estefania Munoz Diaz, Fabian de Ponte Muller, Antonio R. Jimenez, Francisco Zampella *Evaluation of AHRS Algorithm for Inertial Personal Localization in Industrial Environment.*

[2] Sebastian O.H. Madgwick, Andrew J.L Harrison, Ravi Vaidyanathan. *Estimation of IMU and MARG orientation using a gradient descent algorithm.* 2011 IEEE International Conference on Rehabilitation Robotics.

[3] Xiaoping Yun, Eric R.Bachmann, Hyatt Moore IV, James Calusdian *Self-contained Position Tracking of Human Movement Using Small Inertial/Magnetic Sensor Modules.* 2007 IEEE International Conference on Robotics and Automation.