Improving Cooperative Trajectory Mapping Applications with Encounter-based Error Correction

Wei Chang, Jie Wu, and Chiu C. Tan
Department of Computer and Information Sciences
Temple University, Philadelphia, PA 19122
Email: {wei.chang, jiewu, cctan}@temple.edu

***Abstract*—Cooperative trajectory mapping is an emerging technique that allows users to create such a map by using data that is collected from each other's mobile phones. This technique has been proposed for many applications, such as people localization, public transportation tracking, and traffic monitoring. Unlike the traditional GPS, cooperative trajectory mapping only requires information about the departure distance and moving direction from the previously reported position. This avoids problems like the high energy consumption of GPS and weak GPS signals in indoor conditions. However, the new technique also brings about other problems, such as measurement errors in cooperative trajectory mapping, which is when a measurement error causes the spatial relations among users to be wrong. We propose an encounter-based error correction algorithm to efficiently reduce measurement errors. Extensive simulation experiments are performed to validate our solutions.**

***Index Terms*—Cooperative trajectory mapping, encounter, measurement error, mobile phones, relative error.**

I. Introduction

Maps are useful in helping people navigate through unfamiliar places. However, ensuring that the maps remain up-to-date is a constant challenge. Moreover, electronic maps may not always be available, which thus motivates researchers to consistently develop more accurate and efficient map construction techniques. Cooperative trajectory mapping is an emerging technique for map construction that takes advantage of the different sensors that are embedded in smartphones to create maps of users' trajectories. This type of map is known as a *trajectory map*.

Unlike traditional map construction problems, GPS is generally not used when cooperatively building the map due to its high energy consumption [1], [2] and the unavailability of GPS signals in certain environments, such as indoors. Instead, the smartphone's sensors, like the accelerometer and electronic compass, are used to collect information like the departure distance, moving speed, and direction between consecutive sampling times [1]. This data is then transmitted to a central depository via a 3G or 4G connection, which, in turn, processes the data from multiple users to construct a trajectory map. This type of map can be used in various applications, such as traffic monitoring [3], public transportation tracking [4]–[6], and people localization [2], [7], [8].

An important issue that arises when constructing a trajectory map is dealing with measurement errors from the sensor data. In this paper, the measurement error is also known as measurement noise, or noise. A slight measurement error can have a larger impact in the overall map if left uncorrected. For example, a 1° error on a compass will result in a difference of 1.74 miles after a user has traveled 100 miles. Prior researchers have also recognized the importance of measurement errors, but to date, research has only used a simple noise model to address the problem [1].

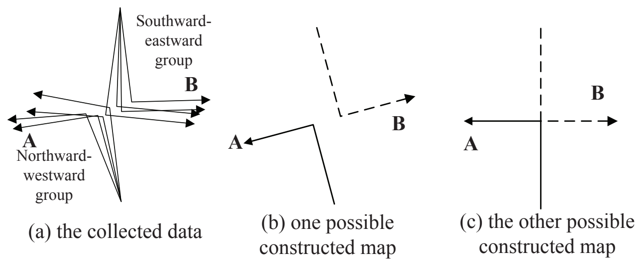


Fig. 1. The effect of a measurement error. There are two groups of users: one northward and turning westward and the other southward and turning eastward. Because of the measured noise, the reported trails of users may be different, as shown in Fig. 1(a). Only based on the collected trajectories, the central server may create two different maps, as shown in Figs. 1(b) and 1(c).

We illustrate the impact of error correction in the example shown in Fig. 1. Suppose that there are two groups of users; one group of users is moving North and then turning West, while the other group moves South and then turns East. Users in each respective group are moving along the same path. In Fig. 1(a), we use solid arrows to depict each user's reported trails. We see that, due to the effects of measurement noise, there is some variance in the reported trails, even though all of the users are traveling along the same path. As a result, the server that is using the collected data can build either the map shown in Fig. 1(b), or the one depicted in Fig. 1(c). We cannot simply average the data to build the map since it is possible that the noise may not follow a normal distribution. Furthermore, during the process of map building, two spatially disjointed paths may be falsely reported as a pair of paths intersecting with each other (*false positive*), or two joined paths may be depicted as unrelated (*false negative*).

In this paper, we propose an encounter-based error cancellation algorithm, which can correct measurement errors, even if every phone has different error properties. By letting the server periodically check for any inconsistencies between users' reported trajectories and their encounters with other users. An example of an inconsistency is when two users do not report meeting each other but their trajectories suggest otherwise. Whenever an inconsistency is found, the server will adjust the reported trajectories accordingly. Furthermore, our solution will make opportunistic use of any permanent access points (APs) that a user encounters to improve error cancellation.

The main contributions of our paper are as follows:

1) we are the first to explore the use of encounter information to correct the measurement error in cooperative trajectory mapping applications;
2) we design a realistic measurement error model that considers both systematic errors and random noise;
3) we propose an encounter-based error cancellation algorithm that is effective against systematic and random noise;
4) we validate the effectiveness of our solutions through extensive simulations. In particular, we focus on the impact of false positive and false negative intersections on the performance of the traditional shortest path routing protocol.

The remainder of the paper is organized as follows. In Section II, we introduce some related work. The system model, previous solution, and challenges are given in Section III. In Section IV, we provide the framework of our solution. Sections V and VI respectively introduce our accessorial anchor-based error reducing algorithm (AAER) and anchor-free error reducing algorithm (AFER). The technical details are presented in Section VII. We have a case study in Section VIII, which focuses on a routing application: friend locator. The performance analysis and evaluation are described in Section IX. We make a conclusion and provide our future research goals in Section X.

II. Related Work

One of the earliest applications of cooperative trajectory mapping is a mobile social network-based [9] navigation system that was proposed by Constandache et al. [1]. Each user in the mobile social network will periodically report his trajectory and his encounter information to the server. The server will use this information to build a set of directions and displacements that allows friends to locate each other. Later work by Constandache et al. [10], and Thiagarajan et al. [2], [11] also applied a similar idea to other applications. The main difference of our work is that prior research used a relatively simple noise model and only considered noise cancellation by a single user, while we consider a more realistic noise model that has both systematic errors and random noise. We use encounter information among multiple users to reduce errors.

In Thiagarajan et al. [12], the authors propose a low-energy and accurate trajectory mapping approach, and they also discuss the trade-off between GPS and multi-sensor devices. However, their work is based on the assumption that the electronic map is available to use. In our paper, we assume that the server needs to build the trajectory map from users' uploaded data, and then users can be localized on this map.

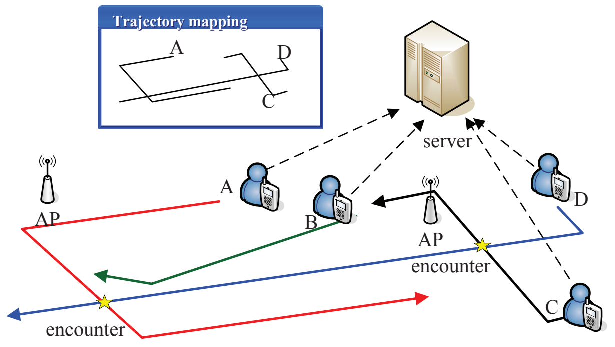


Fig. 2. System model.

Cooperative trajectory mapping shares similar characteristics with the inertial navigation system (INS) that is used in submarine navigation [13]–[15]. Both techniques use the recorded moving distance and direction to determine a location, and both are subject to drifting because of the sensors' noise [16]. INS research has two general approaches to address this problem. The first approach is to use filtering techniques, such as the Kalman filter [17], [18] and particle filters [19], [20], to limit the effects of the measurement noise. The second approach is to apply noise cancellation methods using GPS, assisted GPS, or Wi-Fi [7], [8], [21], [22]. A key difference in our approach is that our technique is more flexible since we emphasize the related locations of each user rather than the physical locations. A recent paper [23] also considers the problem of correcting systematic errors by reducing relative errors. However, that solution only considers the case of existing APs, and does not provide solution for AP-free case.

Finally, Priyantha et al. [24] proposed an anchor-free localization (AFL) algorithm to resolve the localization problem in sensor networks. The goal of [24] is to determine the position coordinates of every sensor via local node-to-node distance, even if the physical location of the nodes is unavailable. However, this solution cannot be used to build a trajectory map because the positions are static spot locations. In the process of creating the trajectory map, we consider the trajectories of moving nodes.

III. Background

A. System Model

A cooperative trajectory mapping system has two basic components, as shown in Fig. 2.

1) **A remote server.** The server collects users' data and build the trajectory map with that information. The server also provides additional services based on the trajectory map, such as friend locator or location routing.
2) **User smartphone.** All users will report their trails and encounters to the server. The reported trails are stored as the displacement and direction between consecutive samplings. The details about trails and encounters will be explained later.

Besides these two basic components, our solution can also take advantage of any AP, such as a WiFi AP, that a user

TABLE I
TABLE OF NOTATION

L	Reported displacement
$k(t)$	Systematic error in displacement measurement
l	Real displacement
N	Number of participants in the system
λ	Random noise caused by accelerometer
Θ	Reported moving direction
θ	Real moving direction
$\Delta\theta(t)$	Electronic compass systematic error parameter
δ	Random noise caused by compass
T	Cycle time for reporting data to the server
x_t, y_t	The coordinates of a user's position at time t
$\overrightarrow{E}$	Accumulated error
$\overrightarrow{V}$	Adjustment vector

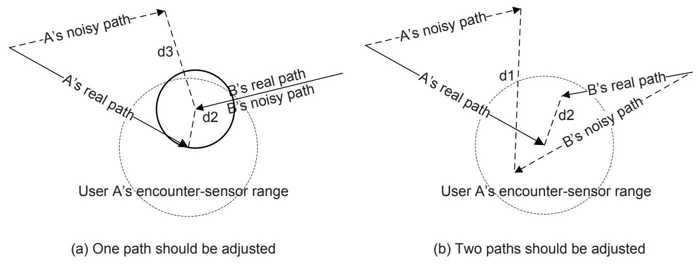

A's noisy path
d3
A's real path
B's real path
B's noisy path
d2
User A's encounter-sensor range
(a) One path should be adjusted
A's noisy path
d1
A's real path
B's real path
d2
B's noisy path
User A's encounter-sensor range
(b) Two paths should be adjusted

Fig. 3. The challenge of making adjustments without the auxiliary information. In Fig. 3(a), even if the server knows that user B's data is accurate, the server is only given the reported data of A and B, real distance d_2, and current distance of A and B in the map being constructed; thus, correctly adjusting user A's path is hard. In a more general case: both users have measurement errors, as shown in Fig. 3(b).

encounters. An AP serves as a fixed location reference, and the physical location of the AP does not need to be known. The purpose of the AP is to quickly establish the spatial relationship among each user's local movement trails and to provide an external global reference for noise cancellation. The AP will periodically broadcast time-stamped beacons, and when a user receives the beacon, he will record the encounters and report to the server. Note that the broadcasting of the beacon is part of the 802.11 standard; thus, any WiFi AP can be used.

We assume that there are N participants in this system. Each user's mobile phone is equipped with an accelerometer, a compass, a wireless receiver, and an encounter sensor. The accelerometer and compass are used to determine a user's displacement and direction, respectively. The wireless receiver is used to receive beacons that are transmitted from the AP. The encounter sensor is used to periodically signal and record the presence of other users. This can be accomplished by using a Bluetooth module [1] built into the smartphone [1].

A user maintains two lists in his smartphone, a *movement list* and an *encounter list*. The smartphone will periodically report the two lists to the server via a 3G or 4G connection [26]. The movement list consists of a displacement series and the moving direction from the last recorded position. The encounter list consists of timestamps and user IDs that denote when the encounter occured. We use mathematical $0°$ to represent East and $180°$ to represent West. The position of a user at time t can be computed by:

$$\begin{pmatrix} x_t \\ y_t \end{pmatrix} = \begin{pmatrix} x_{t-1} \\ y_{t-1} \end{pmatrix} + L \times \begin{pmatrix} \cos(\Theta_t) \\ \sin(\Theta_t) \end{pmatrix}, \tag{1}$$

where L is the reported displacement between two measurements, and Θ is the moving direction. All of the symbols used in this section can be found in Table. I. Note that the trail of each user is recorded in his own coordinate system, which is only relative to the initial (unknown) location of the user [1].

The server will use the two lists from each user to derive a trajectory map. If each measurement is accurate, the users' mutual position information at each time is fully preserved by the server. We will illustrate an application for this map in Section VIII.

[1]As mentioned by [1], although there are several Bluetooth-based neighbor discovery techniques [25], the low detecting rate of short-lived encounters is a problem. In this paper, we assume that all encounters can be detected.

B. Existing Noise Cancellation Solutions

The general idea behind error cancellation in prior work is that each user's noise can be corrected by some physical references [1]. If a user passes by the AP (the user is in the communication range of the AP), the server can then compute the amount of accumulated errors, which causes the trails to drift; then, the user's trails can be repositioned. Let the detected *accumulated error* be $\overrightarrow{E}$. The corresponding *adjustment vector*, which is generated by the server for error cancellation, is $\overrightarrow{V}$ ($\overrightarrow{V} = -\overrightarrow{E}$). If user A encounters user B, who has just been repositioned with the help of an AP, the trail of A can also be corrected since the position of B is likely to be more accurate. Consider that the amount of noise is proportional to time [1] if each user moves with a constant speed. We can also proportionally use the instantaneous correcting vector to adjust the historical trail. For example, let t_1 be the previous time for adjustment; the server detects a new adjustment vector $\overrightarrow{V}(t_2)$ at t_2. Then, the adjustment vector of the position at time t ($t_1 < t < t_2$) is given as:

$$\overrightarrow{V}(t) = \frac{t - t_1}{t_2 - t_1} \overrightarrow{V}(t_2). \tag{2}$$

The solution is inadequate due to the following reasons. Firstly, only false negative encounters are used in the error cancellation; false positive encounters are ignored. By incorporating both false positive and negative encounters, we can improve the error cancellation. Secondly, the direction of the adjustment vector used is the same in all of the data. However, since there is a systematic error of compasses, we should use rotation to adjust the trajectory, which means the direction of the adjustment vector should be different. In this paper, we solve the measurement error problem by using both false positive and false negative, and we use different adjustment directions with different data points.

C. Challenges

In order to correct measurement errors, we need to solve the following three issues.

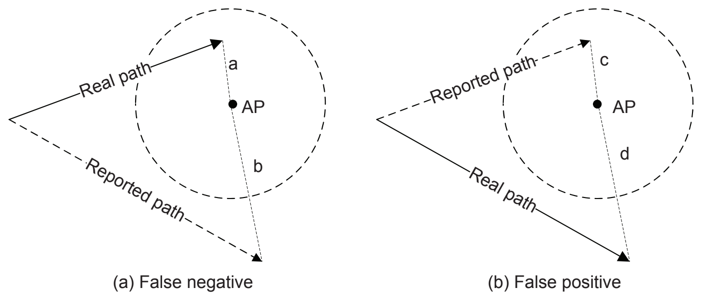


Fig. 4. An example of false negative and false positive when a user encounters an AP.

Firstly, every user's accelerometer and compass may exhibit different error parameters. Without knowing these parameters, we cannot correct the trails since we cannot determine the extent of the error of each user. For example, in Fig. 3 (a), there are two users' reported trails, and both of them consist of noise. Suppose that the server can compute the spatial distance d_1 of the reported trails and can obtain the real distance d_2 from the reported encounter information; we cannot determine how much of an adjustment should be made and in which direction. Even if only one user is inaccurate, without knowing the identity of the accurate user, a correct adjustment is still hard, as shown in Fig. 3 (b). Moreover, users' error parameters may slightly change with time. Knowing some error parameters at a given time cannot guarantee that the problem can also be solved in the future.

Secondly, there are two types of encounter errors which can be detected, and they should be treated differently. *False positive* means that two physically disjointed trajectories are falsely reported as a pair of intersecting trails, while *false negative* represents the situation where two physically joined trajectories are depicted as unrelated. Fig. 4 illustrates both types of error. In the false negative case, the server can obtain the real distance between the users (or between a user and an AP) by the encounter sensors and the false distance between their reported trails. However, in the false positive case, the server cannot attain the real distance since the user is out of the range of the AP.

Lastly, since each user may not move at a constant speed, there is a special case of the false positive error: two reported trails have a spatial intersection with no physical encounter. Considering the speed restriction, two users may pass the intersection at different time. Hence, in the false positive case, we should only consider the reported trails which definitely will have an encounter.

IV. Solution Framework

A key feature of our solution is that when two users meet, each user will independently report their encounter with the other to the server. Hence, we can use encounter information to adjust users' trajectories. Another feature of our solution is about relative measurement errors: although the absolute error of devices may be large and hard to detect, the relative errors can be easily detected by multiple times of physical encounters. If we relatively adjust users' trajectories and let the relative errors of adjusted trajectories be small, the cooperative trajectory mapping system can work well. In a nutshell, our method exploits the relative errors and physical encounters of users, through which different users can interpret the same data in the same way. As a result, instead of estimating absolute error parameters, the relative errors are considered.

We illustrate how the features are used in our solution by using the following example. Consider user A who is physically at location α (loc_α); however, user A reports loc_β to the server because of the accumulated noise. Without some physical reference, we cannot detect the error or measure the magnitude of the error. However, if another user B, who encountered with A before, and who has higher measurement accuracy than A, comes across A, we can determine the relative error of A to B, and we can also compute the error's magnitude. Since the noise is proportional to time, the server can further correct the previously reported trails. Based on our error model, in a long period of time, the systematic error parameters can be regard as fixed ones since we assume that they change very slowly according to time. Therefore, when users encounter each other many times in such period, the system can estimate their relative errors and further adjust their trajectories.

Our proposed method consists of three steps to reduce error:

1) at each smartphone, the application will apply the Kalman filter to eliminate random noise. Considering that the Kalman filter only requires an individual user's historical moving pattern, rather than some global information, it is more efficient to apply the Kalman filter at the user side. After filtering, users will report their moving trails and encounter information to the central server;
2) at the server side, the server will first coarsely adjust the data by previously estimated relative error parameters, then it will detect any false positive cases and false negative cases by using the reported data. After that, the server will slightly adjust the reported locations of users to eliminate those detected inconsistency; during the path correction, the server will also make some hypothesis about the direction of corrections in false positive conditions. Next, the server will use the new upcoming encounter to verify and adjust the hypothesis;
3) after correcting each user's trail, the server will compute the relative error parameters of each user from multiple encounters, and it will also refine the estimated parameters based on new adjusted data. When users report their locations at the next observing time step, the server will first use the parameters to coarsely adjust the position and then will make a slight correction.

When there is no AP present, steps (2) and (3) are carried out by our proposed *anchor-free error reducing algorithm* (AFER); when there are APs present, steps (2) and (3) are our proposed *accessorial anchor-based error reducing algorithm* (AAER).

A. Accessorial anchor-based error reducing algorithm (AAER)

When there are APs present, the chance of encounters is increased, and the APs can be used as fixed references for relative error parameter estimation. The procedure of AAER is shown by Algorithm 1. The details about line 2 of Algorithm 1 can be found in Section VII: B. At each time step, after collecting all of the trails from users, the server will recursively use a hypothesis-based mass-spring (HBMS) adjustment algorithm to estimate each user's real position, as shown in Algorithm 2. The HBMS algorithm will be discussed in Section VII: C. In HBMS, the reported position will be adjusted by different adjustment vectors, which seems like a position being pushed and pulled by some force. In this paper, the *adjustment force* $\overrightarrow{F}$ represents the adjustment vector being used during a position adjustment.

There are two types of adjustment force used in our algorithm: false positive-caused adjustment force and false negative-caused adjustment force. Since the adjustment direction in false positive is uncertain, we use two hypotheses to temporarily store the possible adjustment positions. Later, we use the encounter information to further adjust the hypotheses and to eliminate the wrongs.

Algorithm 1 The AAER algorithm

```
1: for Each sampling time $T$ do
2:     Find of false positive and false negative by mutual encounter and APs
3:     Use HBMS algorithm to adjust the reported trajectories
4:     Record the adjusted positions
```

Algorithm 2 The HBMS algorithm

```
1: Verify previous direction hypothesis by current encounter
2: Adjust by new false negative and estimated relative errors
3: Set up new false positive direction hypothesis
4: for Each hypothesis do
5:     Compute adjustment force and adjust trajectories
6:     if Find a new false positive case from historic data then
7:         Set up new false positive direction hypothesis
8: Update current error parameters
```

B. Anchor-free error reducing algorithm (AFER)

When there are no APs present, we use AFER to correct the errors. Consider that if all of the users have exactly the same error parameters, the cooperative trajectory mapping system can still be used well even if there are huge errors when comparing to the real data; if the majority of the users have same error, we just need to adjust the other users by changing their data according to the majority of the users. Hence, our goal is to find out the relative errors among users by their multiple times of encounters. Since the encounter-based error parameter may not be very accurate, considering that the parameters could slightly change, we further use the adjustment force to refine the estimated error parameters and to eliminate the inconsistency.

Algorithm 3 The AFER algorithm

```
1: for Each sampling time $T$ do
2:     Find out false positive and false negative error
3:     Use the HBMS algorithm to adjust the reported trajectories
4:     Record the adjusted positions
```

Algorithm 3 depicts AFER. The second line in algorithm 3 will be discussed in Section VII: B. The details of the HBMS algorithm will be discussed in Section VII: C. The adjustment's force still has two types of errors, the false positive-caused adjustment force and the false negative-caused adjustment force; they can be computed in the same way as previously stated. The only difference is that, instead of using the distance among users and APs, we use the distance between users. We use error parameters, which were calculated previously, to make an initial estimation at the beginning of each time step. If both of the error parameters of two users are known, the server will use the latest corrected parameter. Then, we make an error cancellation based on the newly reported data.

Unlike AAER, we assume that there is at least one error-free random walker and the identity of this user is known by the central server. The reason for having this assumption is that in order to avoid tortuosity of the constructing map we need to find some physical references. Consider that there are several users moving in a large region by following some moving patterns. Without the physical references, it is possible that the final constructed map is contorted while sub-regions' maps are relatively accurate.

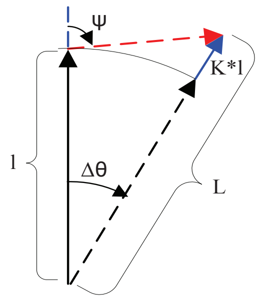
V. Algorithm Details

In order to use AFER and AAER, we first need to determine the noise model. Then, we will discuss several auxiliary functions. By the end of this section, we will have an additional discussion.

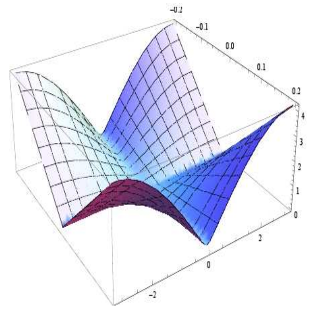
A. Noise Model

The accelerometer and compass each have their own respective noise model. Table I contains the notations used. We first consider the accelerometer. There are two types of errors: the systematic errors and the random errors. The systematic error is proportional to the moving time or moving distance. Moreover, the magnitude of the systematic error may change over time. For example, consider a person walking continuously for a couple of hours. The size of their average step at the beginning will be different from that towards the end. We use $k(t)$ to represent a systematic error which may slightly change over a long period of time. We let λ denote the random noise, which follows the normal distribution. L represents the reported displacement, and l is the real displacement.

$$L = l + k(t) \times l + \lambda \tag{3}$$



(a) Sensor error



(b) Error change pattern

Fig. 5. Sensor error and the change pattern of its magnitude. The left figure illustrates the symbols that we used in our noise model; the right figure shows the change pattern of the error magnitude according to different parameter values.

The readings from an electronic compass, Θ, can be regarded as the sum of real data (θ), random noise (δ), and systematic noise ($\Delta\theta$). This systematic noise may also change with time.

$$\Theta = \theta + \Delta\theta(t) + \delta \tag{4}$$

To demonstrate the effect of slightly modified noise, we temporarily ignore the random noise. Assume that $p = k(t) + 1$. The accumulated error $\overrightarrow{E}$ in a time period can be computed by:

$$\text{The amount of } \overrightarrow{E}\text{: } |\overrightarrow{E}| = l \times \sqrt{p^2 - 2p\cos(\Delta\theta) + 1} \tag{5}$$

$$\text{The direction of } \psi\text{: } \cos(\psi) = \frac{p\cos(\Delta\theta) - 1}{\sqrt{p^2 - 2p\cos(\Delta\theta) + 1}} \tag{6}$$

Based on our assumption of the noise model, Fig. 5(a), we use simulations to generate the change pattern of the error magnitude as shown in Fig. 5(b), when $\Delta\theta$ varies from $-\pi$ to π and k changes from -0.2 to 0.2. If one of the noise parameters is relatively large, both of the errors cannot be neglected.

The random noise, λ and δ, can be eliminated by letting each user's smartphone apply the Kalman filter to process the data before uploading to the server. This can be accomplished since the random noise follows the normal distribution, and the moving pattern of a user can be computed by the user's smartphone itself. The Kalman filter is a set of mathematical equations that provides an efficient computational (recursive) means for estimating the state of a process in a way that minimizes the mean of the squared error [27].

B. False Positive and False Negative Error Detection

At each reporting time, the server will obtain users' reported relative positions and their distance from nearby users. In order to detect the false negative error, the server needs to compare two records: the encounter readings and trajectories. From there, the server derives an error vector (the error's magnitude and direction). For the false positive error, the actual distance between users is unknown (the distance is measured by the signal strength of bluetooth) since they are not within the sensor range of each other. We temporarily use the sensor radius of bluetooth to represent the actual distance.

TABLE II
TABLE OF NOTATION FOR AUXILIARY FUNCTIONS

S_{max}	the maximum speed
(x_b, y_b) / (x_e, y_e)	beginning or end location of a displacement
t_b / t_e	beginning or end time of a displacement
d	length of a displacement
$(x(A,t), y(A,t))$	the location of user A at time t
R	sensing range

However, there is a special case when dealing with the false positive error. If the encounter sampling time is different from the cycle time when reporting data, then the server needs to determine whether two spatial encounter trails have physically encountered each other at some point in time. However, because the instantaneous velocity of a user may vary, we should consider all of the possible moving conditions of a user. In order to simplify the solution, we add a new dimension time to the traditional X-Y coordinates. Table II contains the notations that are used in the remaining parts of this paper.

Given a specific distance, there are multiple ways in which a user can move. For instance, the user can first move at his maximum speed to finish the reported displacement and then stop and wait at the end. Alternatively, the user can also wait first at the beginning and then move to complete the distance just on time. Therefore, there are two trajectory boundary functions:

$$\frac{x - x_b}{x_e - x_b} = \frac{y - y_b}{y_e - y_b} = \frac{t - (t_e - d/S_{max})}{d/S_{max}} \tag{7}$$

$$\frac{x - x_b}{x_e - x_b} = \frac{y - y_b}{y_e - y_b} = \frac{t - t_b}{d/S_{max}} \tag{8}$$

Assuming that we have two users, A and B, both report one displacement in a time interval from t_b to t_e. The initial position of A is $(x(A,t_b), y(A,t_b))$, and the end of the displacement is $(x(A,t_e), y(A,t_e))$. Similarity, we have B's displacement from $(x(B,t_b), y(B,t_b))$ to $(x(B,t_e), y(B,t_e))$. Hence, at a given time t, users A and B should definitely encounter with each other if their reported trajectories satisfy the following formula:

$$(x(A,t) - x(B,t))^2 + (y(A,t) - y(B,t))^2 \leq R^2 \tag{9}$$

After simplification, we can get:

$$R^2(a^2 + c^2) - (ad - bc)^2 \geq 0 \tag{10}$$

where,

$$a = \frac{(x(A,t_e) - x(A,t_b)) - (x(B,t_e) - x(B,t_b))}{t_e - t_b} \tag{11}$$

$$b = -a \times t_b + x(A,t_b) - x(B,t_b) \tag{12}$$

$$c = \frac{(y(A,t_e) - y(A,t_b)) - (y(B,t_e) - y(B,t_b))}{t_e - t_b} \tag{13}$$

$$d = -c \times t_b + y(A,t_b) - y(B,t_b) \tag{14}$$

Hence, a pair of spatial intersected trajectories without any corresponding encounter record may or may not be the false positive case. In our solution, the false positive only refers to the cases, where the reported data indicates definitely encountered while having no corresponding encounter records. Clearly, the probability of having a false positive case is much less than that of having a false negative case.

C. Hypothesis-based Mass-spring Adjustment (HBMS)

The HBMS is used to estimate the optimal positions of users. HBMS first computes the adjustment force in false positive and false negative cases, respectively, which will be discussed in Section VII: C-1. Since the adjustment direction of false positive is unknown, the HBMS algorithm will make two hypotheses about the correction's direction. Then, HBMS will recursively reposition each user's position based on the hypothesis. The details of a recursive reposition can be found in Section VII: C-2. In order to enhance the efficiency of HBMS, we first use some error parameter, which has been computed in previous steps, to make a coarse correction which will be introduced in Section VII: C-4. Then, we refine the positions of users based on the detected relative errors. After finding the optimal position, HBMS will update the error parameter of users' trails. Wrong hypotheses will be eliminated later in Hypothesis Verification, which can be found in Section VII: C-3.

1) Adjustment Force: assume that there are two users, i and j, who are neighbors. In the false negative case, by using reported trails, we can compute the relative distance $\widetilde{d_{ij}}$ between users, and we can also obtain the real physical distance d_{ij} through RSSI readings. The adjustment's force $\overrightarrow{F_{ij}^{-}}$ can be calculated as:

$$\overrightarrow{F_{ij}^{-}} = \overrightarrow{u} \times (\widetilde{d_{ij}} - d_{ij}), \tag{15}$$

where $\overrightarrow{u}$ is the unit vector from location i to j. Since the trajectories of users are relatively adjusted, the adjustment forces will be associated with the users, whose trajectories have not been updated for a long time or have only been adjusted according to a small portion of users' data.

In the false positive case, we cannot obtain the real distance d_{ij} or the adjustment direction. As shown in Fig. 6, the real path can be located at either the same side of the error path or the other side. Therefore, we need two hypotheses to respectively store the adjustments.

$$\overrightarrow{F_{ij}^{+}} = \pm \overrightarrow{u} \times (R - d_{ij}) \tag{16}$$

The synthesized force of a node in a hypothesis is the sum of the forces gotten from all of the nodes' neighbors. We have to mention that we cannot guarantee the accuracy of the false positive adjustment; what we do just eliminates some obvious errors during the process of map construction. Moreover, as we have mentioned in Section VII.B, the false positive cases are much harder to detect than false negative cases; the amount of false positive-caused adjustment force is much less than that of false negative. Hence, there are only a few hypotheses.

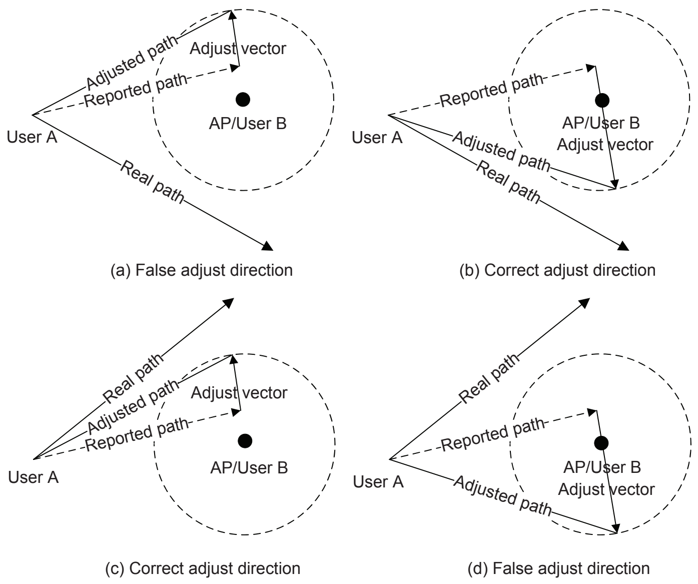


Fig. 6. The possible adjustments in the false positive case. The server receives a reported trajectory from user A and detects that the false positive case has happened. Since the server cannot get any information about the real path (the error-free path), the server needs to check both adjustment directions.

2) Recursively Reposition: changing one user's path will also impact the historic path of other users. In other words, a false positive adjustment at time t may cause the happening of another false positive case at some time between last adjusted time and current time. As a result, the estimated position adjustment should be accomplished recursively: once a new false positive been detected in the historic data, new hypotheses will be generated to eliminate the errors.

3) Hypothesis Verification: the position hypothesis can be verified by using follow-up encounters with other users. In AAER, the hypothesis can be checked by using encounters with other users whose paths were just adjusted or had encountered an AP. In AFER, if one hypothesis incurs too many false-encounters at a later time, the hypothesis will be eliminated. This idea comes from the fact that if all of the users have the same error in their sensor device, the relative position relationship may still be correct. Although using an AP is not a prerequisite in the HBMS, using APs will allow us to easily verify a hypothesis and estimate error parameters. Moreover, in order to reduce the computing complexity, the hypotheses will be deleted if they can not be verified in a period of time.

4) Error Parameter Estimation: we can quickly determine the relative error parameters of users if we can find a group of n different users who directly encountered each other n times, which will allow us to generate n linearly independent equations. This is because the systematic error parameters change slightly through time. For example, assume that the central server finds out that users A and B encountered each other twice in a period of time, then the relative errors between them can be estimated. However, since the systematic error may change with time, the estimated relative error parameter

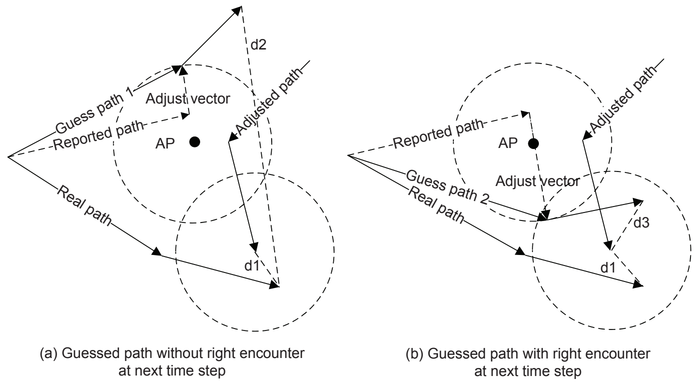


Fig. 7. The verification of a hypothesis.

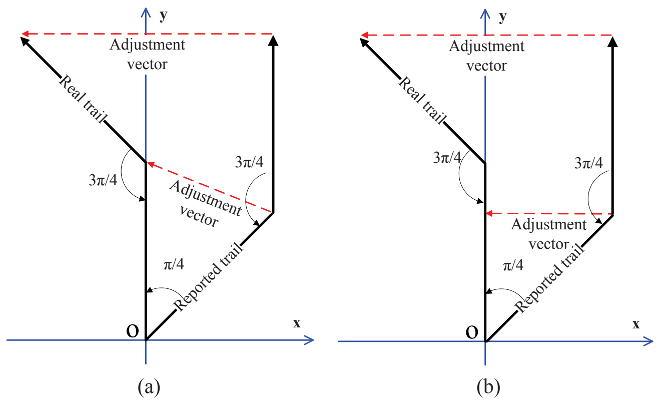


Fig. 8. The historical error cancellation.

at current time may not be accurate in future. Hence, the estimated relative error parameters should be updated. There are two ways to update the parameter: either by multiply encounters in future, or by the detected inconsistency of future data, which causes adjustment force.

In order to quickly find the optimal location during the HBMS, at the beginning of each time step's adjustment, we can use the prior estimated relative error parameters to coarsely adjust the trails in advance. After each time period, we compute the time interval between the nodes' (involved in the false encounter) previous adjustment times and current times. Then, we calculate the ratio of the adjusted amounts to the corresponding time interval and update the estimated parameters. At the next time step, we first use the parameter to coarsely adjust the trails, and then we apply the HBMS.

D. Historical Error Cancellation

Users may not have any encounters in a period of time. Also, the reported trajectory in this time period may not be accurate. Once we know how to adjust the instant position, the historical positions could also be corrected. We name the process of applying error cancellation to the historical reported positions as *historical error cancellation*.

Assume that, at time t, the server finds out an adjustment vector. The existing solution [1] is to reposition the historical trails by using a proportioned adjustment vector, which we have mentioned in the background part of our paper. However, this adjustment is not true if the compass contains systematic errors. In Fig. 8, suppose that the systematic error for a compass is $\pi/4$, which means that moving directly north should be reported as moving towards north-east. If we use the existing cancellation algorithm, the adjustment vector in the middle will be parallel to the instantaneous adjustment vector, which is incorrect, as shown in Fig. 8(b). We cannot simply apply the direction of the current adjustment vector to users' historical trajectories.

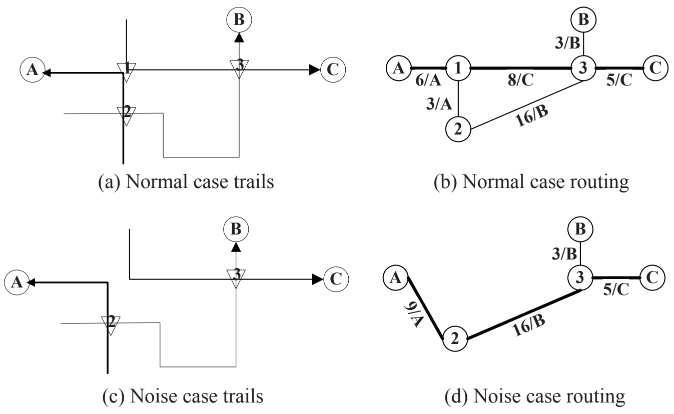
Our solution is shown in Fig. 8(a). During the adjustment, the server should first compute the degree of the error angle. If the time interval between the instantaneous time and the previous reposition is not too long, all of the points in the trails should have the same error angle. Therefore, we shrink or expand the length of each reported displacement and then rotate the reported trails with the computed error angle. The details are as follows: suppose that at time t, the server detects a false negative, and finds out that the real direct distance from the user's current location to last adjusted location is L. However, according to the reported trajectory, this direct distance is L_N. Since the systematic error parameters only slightly change with time, we can regard them as fixed. Therefore, the displacement error parameter can be computed by $K = \frac{L_N}{L} - 1$. By using this computed K, we first adjust the length of each displacement, and then we rotate the whole trajectory from the last adjusted place.

E. Additional Discussion

The adjustment results of false positive and false negative should hold different weights because the false positive can only correct the trails partially; the adjustment results of false negative are much more accurate than those of false positive. Therefore, one corrected false positive position can be further adjusted by a new-found false negative. For example, t_1 is the latest time for false negative-based reposition, and t_2 ($t_2 > t_1$) is the nearest time for false positive-based adjustment. If at t_3 ($t_3 > t_2 > t_1$) the server detects another false negative case, the historical trails from t_1 to t_3 can be adjusted. Thus, the server should also store the originally reported positions if a false positive adjustment was made. The reason that we still use the case of false positive is that if a certain application requests the built map at time t_2, without applying the false positive adjustment temporarily, there will be some false intersections in the map, which will definitely change the relative position relationship of the intersection. As a result, we use false positive for temporary adjustments and use false negative for permanent repositioning.

VI. Routing Application: Friend Locator

We describe a representative application that can make use of a trajectory map. Friend locator is a typical application of cooperative trajectory mapping: a server periodically collects users' trajectories and answers the routing request that helps one user to find another. The response of the routing consists



(a) Normal case trails (b) Normal case routing (c) Noise case trails (d) Noise case routing

Fig. 9. The effect of measurement error during routing. Figs. 9(a) and 9(c) represent the reported trajectories. Figs. 9(b) and 9(c) represent the corresponding routing graph. The digital stands for the length of a path and the letter indicates who reported the path.

of the reported trails from several users whose trajectories have spatial intersections with the others. In detail, the path information consists of a series of directions and distances (such as turn right, continue for 10 m, etc.). These directions are obtained from the reported trails.

The implementation details of the application are as follows. When user A sends a routing request to the server, the server first computes the spatial intersections of the users' trails, which we term *inner routing nodes*. The distance between each pair of inner routing nodes is the length of the path that the user covered. We term the current position of a user as the *outer routing node*, which is linked with one, and only one, inner routing node. Every edge in the routing graph is also associated with actual walking trajectories, which consist of several displacements and turning angles. After getting the spatial intersections of the paths, the server builds a routing graph. Then, we apply Dijkstra's shortest path algorithm to the graph. After we obtain the shortest path of the routing graph, the server will return it to the requester.

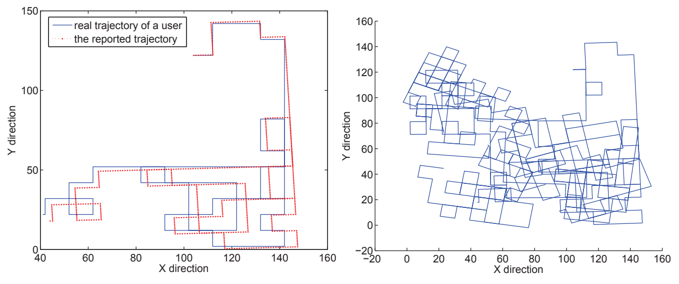
Algorithm 4 The friend locator algorithm (server side)

```
1: Compute spatial intersection based on collected trails
2: for Each pair of intersection do
3:     if The intersections are directly connected by a user's trails then
4:         Add an edge between the intersections
5:         Set the length of the real trail as the weight of the edge
6:         Associate the real trajectory with the edge
7: Apply shortest path algorithm on the routing map
8: Find the real trajectories, which are associated with the shortest path
9: Return a list consisted of moving directions and displacements
```

However, the quality of this application is restricted by the measurement error of the sensor devices. Those error paths

Algorithm 5 The friend locator algorithm (user side)

```
1: Send a routing request to the server
2: Receive a list consisted of moving direction and displacements
3: for Each tuple of the list do
4:     if User cannot find a corresponding path then
5:         Resend routing request
6:     else
7:         Move as the list guided
8: if All of tuples in list have been taken but the user does not arrive the destination then
9:     Resend routing request
```



(a) User's real & reported trails. (b) Reported trails of 10 users.

Fig. 10. An example of simulation used data.

may cause both false positive intersections and false negative intersections. Fig. 9 shows the effect of the measurement error on the length of the shortest routine path. We can illustrate this using the example in Fig. 9. There, we have three users, A, B, and C. The arrow lines represent the paths of users, and the triangles between the paths are the spatial intersections. Fig. 9 (a) is the noise-free example, and Fig. 9 (b) is the corresponding routing graph. Suppose that user A wanted to be guided to C. The shortest path is shown as the bold line in Fig. 9 (b). However, assume that there is an error undetected in Fig. 9 (c), such that the inner routing node 1 is missing. The shortest path will become Fig. 9 (d) with its distance changed from 19 to 30.

VII. Performance Analysis and Evaluation

A. Experimental Goals

We use Matlab to perform our simulation experiments. We are interested in the following questions.

1) The relationship between the length of the observation time and the accuracy of our algorithm.
2) User density and the accuracy of our algorithm.
3) AP density and the accuracy of our algorithm.
4) The distribution of error parameters and the accuracy of our proposed detection algorithm.

B. Evaluation Metric

The metric we used to evaluate our algorithm is *Inaccuracy*. This is computed by using the shortest path algorithm based on the adjusted trajectories of users. If there is an error in the routing, such as returning a non-existing path, the user will first search nearby places: if the user find a similar path as routine indicated, he will continue following the navigation; if the user can not find the path, he will send the routine request again from the last existing intersection. Since the realistic walking length of the user must be shorter or equal to the real shortest path, we use the ratio of the additional length to the shortest length to measure our methods and others. Thus:

$$Inaccuracy = \left| \frac{\widetilde{d_{ij}} - d_{ij}}{d_{ij}} \right|, \tag{17}$$

where $\widetilde{d_{ij}}$ represents the total length of the real walking path, and d_{ij} is the length of the real shortest path.

C. Simulation results

We first synthetically generate a 15×15 grid map and set the distance between neighborhood as 10 distance units. Then, we randomly generate the noise-free moving trajectories of every user. The speed of each user varies from 1 distance unit per second to 10 distance units per second. We convert the coordinates of trajectories to sensor readings, which consist of displacement and the moving direction. The shape of noisy trajectories is shown in Fig. 10. The parameters of the noise are also generated randomly. The distribution of the parameter follows normal distribution.

In order to guarantee that the routing request can always be responded to, we only use the trails which are connected. The encounter sensors' sampling times are the same as users' trail reporting times. For the consideration of generality, each data point in our simulation is the average result of 5 simulations. The shortest path algorithm used in our simulation is the Dijkstra algorithm. Consider that if the relative positions of users are correct, then the spatial intersections of their trails are correct. In order to show the importance of the correct spatial encounter, in our method, we use the routing paths, which are consisted of spatially jointed trajectories. We compare our results with a modified version of [1]. Note that in their method the routing path is composed by several users' trajectories, which are joined only at the physical encounter places. Moreover, a special pruning algorithm is also used in [1]. Since the pruning algorithm only affects the computing speed rather than accuracy of routing, we do not use the pruning algorithm. For ease discussion, we name the modified solution as Modified Proportional Error Cancellation (MPEC).

The first tested factor is the length of the observations. We choose different sample times from 200 to 500. Each APs' sensor range is set as 6 distance units, and each encounter's sensor range is 3. The initial positions of the users are randomly deployed. Fig. 11 shows our simulation result. We can see that our AP-free solution is related with observation time, since as the time growing, there are more physical encounters happened. However, since the error parameters change with time, the AP-free solution can not eliminate all the errors.

The second tested factor is the user density, as shown in Fig. 12. We test 5 to 11 users in the grid map. Although

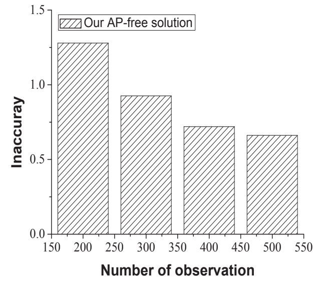


Fig. 11. AP-free solution with time.

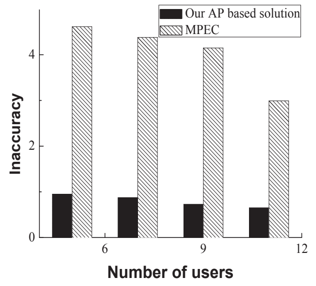


Fig. 12. User density vs. inaccuracy.

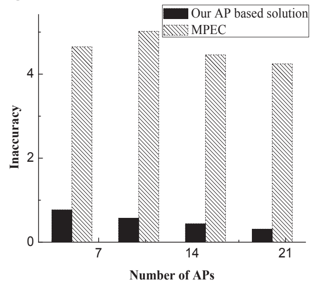


Fig. 13. AP density vs. inaccuracy.

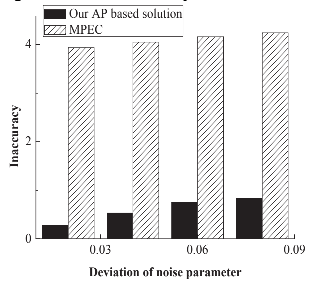


Fig. 14. Error vs. inaccuracy.

the estimated error parameter can be updated when users encounter each other, the error of a user at one time may also be accumulated, which further impacts the quality of the friend locator application.

AP density is our third consideration. In the grid map, we randomly deploy 5 to 20 APs. Intuitively, if the density of an AP is large enough, the accuracy of the application will still be high even if the error parameters may change with time. Moreover, after encountering an AP, the estimated error parameter can still be used for corrections in a period of time. Fig. 13 is our simulation result when the error parameter is 0.08 for displacement error deviation and 0.2 for compass.

Our last tested factor is the initial deviation of the error parameters. During simulation, we first set up an initial error in the sensors. Then, we let the noise slightly increase or decrease along with time. The initial amount of errors may have some significant impact on the spatial encounter-based routing results, especially the structure of spatial intersections of the reported trajectories, if they are not corrected in time. However, since the routing results of MPEC only use physical encounters, the errors only affect the real walking distance of users, who follows the previous user's trajectory. Fig. 14 shows our simulation results.

VIII. Conclusion

In this paper, we consider the problem of accumulative measurement errors in cooperative trajectory mapping. We use a realistic noise model and propose an encounter-based error cancelation algorithm that is effective against measurement errors. Then, we use extensive simulations to validate our solution. Our future work will consider some extensions. The first one is how we can determine a malicious user, who always reports wrong trails, from the normal user, who has relatively large amounts of noise. The second extension is about reducing the computing complexity of the HBMS algorithm. Finally, we plan to build the system on a real platform such that we can test the following items: the magnitude of the measurement errors, the impacts of road structures, traveling patterns, and battery drain on multi-sensors.

Acknowledgment

This research was supported in part by NSF grants ECCS 1128209, CNS 1065444, CCF 1028167, CNS 0948184, CCF 0830289, and REU 1156574.

References

[1] I. Constandache, X. Bao, M. Azizyan, and R. Choudhury, "Did you see Bob?: human localization using mobile phones," in *ACM MobiCom*, 2010.

[2] I. Constandache, R. Choudhury, and I. Rhee, "Towards mobile phone localization without war-driving," in *IEEE INFOCOM*, 2010.

[3] P. Mohan, V. Padmanabhan, and R. Ramjee, "Nericell: rich monitoring of road and traffic conditions using mobile smartphones," in *ACM SenSys*, 2008.

[4] A. Repenning and A. Ioannidou, "Mobility agents: guiding and tracking public transportation users," in *ACM AVI*, 2006.

[5] J. Froehlich, T. Dillahunt, P. Klasnja, J. Mankoff, S. Consolvo, B. Harrison, and J. Landay, "UbiGreen: investigating a mobile tool for tracking and supporting green transportation habits," in *ACM CHI*, 2009.

[6] S. Carmien, M. Dawe, G. Fischer, A. Gorman, A. Kintsch, J. Sullivan, and F. James, "Socio-technical environments supporting people with cognitive disabilities using public transportation," *ACM TOCHI*, 2005.

[7] P. Bahl and V. Padmanabhan, "RADAR: An in-building RF-based user location and tracking system," in *IEEE INFOCOM*, 2000.

[8] Y. Cheng, Y. Chawathe, A. LaMarca, and J. Krumm, "Accuracy characterization for metropolitan-scale Wi-Fi localization," in *ACM MobiSys*, 2005.

[9] N. Kayastha, D. Niyato, P. Wang, and E. Hossain, "Applications, Architectures, and Protocol Design Issues for Mobile Social Networks: A Survey," *Proceedings of the IEEE*, vol. 99, no. 12, pp. 2130 –2158, dec. 2011.

[10] A. Thiagarajan, L. Ravindranath, K. LaCurts, S. Madden, H. Balakrishnan, S. Toledo, and J. Eriksson, "VTrack: accurate, energy-aware road traffic delay estimation using mobile phones," in *ACM SenSys*, 2009.

[11] A. Thiagarajan, J. Biagioni, T. Gerlich, and J. Eriksson, "Cooperative transit tracking using smart-phones," in *ACM SenSys*, 2010.

[12] A. Thiagarajan, L. Ravindranath, H. Balakrishnan, S. Madden, L. Girod *et al.*, "Accurate, low-energy trajectory mapping for mobile devices," *ACM NSDI*, 2011.

[13] P. Maybeck and G. Siouris, "Stochastic models, estimation, and control, Volume I," *IEEE Transactions on Systems, Man and Cybernetics*, 1980.

[14] J. Farrell and M. Barth, *The global positioning system and inertial navigation*. McGraw-Hill Professional, 1999.

[15] D. Titterton and J. Weston, *Strapdown inertial navigation technology*. Peter Peregrinus Ltd, 2004.

[16] P. Gilliéron, D. Buchel, I. Spassov, and B. Merminod, "Indoor navigation performance analysis," in *ENC GNSS*, 2004.

[17] G. Welch and G. Bishop, "An introduction to the Kalman filter," *University of North Carolina at Chapel Hill, Chapel Hill, NC*, 1995.

[18] R. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, 1960.

[19] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P. Nordlund, "Particle filters for positioning, navigation, and tracking," *IEEE Transactions on Signal Processing*, 2002.

[20] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 174–188, 2002.

[21] D. Niculescu and B. Nath, "VOR base stations for indoor 802.11 positioning," in *MobiCom*, 2004.

[22] M. Youssef, A. Youssef, C. Rieger, U. Shankar, and A. Agrawala, "Pinpoint: An asynchronous time-based location determination system," in *ACM MobiSys*, 2006.

[23] W. Chang, J. Wu, and C. Tan, "Encounter-based noise cancelation for cooperative trajectory mapping," in *IEEE PerCom*, 2012.

[24] N. Priyantha, H. Balakrishnan, E. Demaine, and S. Teller, "Anchor-free distributed localization in sensor networks," in *ACM SenSys*, 2003.

[25] Smith, T.J. and Saroiu, S. and Wolman, A., "Bluemonarch: a system for evaluating bluetooth applications in the wild," in *ACM MobiSys*, 2009.

[26] F. Fitzek, A. Kopsel, A. Wolisz, M. Krishnam, and M. Reisslein, "Providing application-level QoS in 3G/4G wireless systems: a comprehensive framework based on multirate CDMA," *IEEE Transactions on Wireless Communications*, 2002.

[27] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," Chapel Hill, NC, USA, Tech. Rep., 1995.