Encounter-based Noise Cancelation For Cooperative Trajectory Mapping

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 a map by using data collected from each participant's mobile phones. Unlike the traditional localization problem, where GPS is usually required, cooperative mapping only requires information about the relative distance and direction from the previously reported position. In this paper, we consider the problem of measurement error, which is when the measurement error causes the spatial relations among users to be wrong, in cooperative trajectory mapping. We propose an encounter-based error canceling algorithm to efficiently reduce measurement errors. Extensive simulation experiments are performed to validate our solutions.**

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

I. Introduction

Cooperative trajectory mapping is an emerging technique that takes advantage of different sensors embedded in smartphones to create maps of users' trajectories. This type of map is known as a *trajectory map*. GPS is generally not used when building the map due to its high energy overhead [1], [2] and the unavailability of GPS in certain environments, such as indoors. Instead, the smartphone's sensors, like the accelerometer and electronic compass, are used to collect information like the 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 create a trajectory map. This type of map can be used in various applications, such as traffic monitoring [3], public transportation tracking [4], [5], and people localization [2], [6], [7].

An important issue that arises when building a trajectory map is dealing with measurement errors from the sensor data. In this paper, the measurement error is also known as noise. A slight measurement error can have a larger impact in the overall map if left uncorrected. During the process of map building, two disjointed paths may be falsely reported as a pair of paths intersected with each other, or two joined paths may be depicted as unrelated. Prior researchers have also recognized the importance of measurement errors but have only used a simple noise model to address the problem.

In this paper, we propose an encounter-based error cancellation algorithm. We let the server periodically check for any inconsistencies between users' reported trajectories and their encounters. When an inconsistency is found, the server will adjust the trajectories accordingly.

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

II. Related work

One of the earliest applications of cooperative trajectory mapping is a mobile social network-based 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. [8], and Thiagarajan et al. [2], [9] 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.

Cooperative trajectory mapping shares similar characteristics of the inertial navigation system (INS) used in submarine navigation [10]. Both techniques are subject to drifting because of the sensors' noise [11]. INS research has two general approaches to address this problem. The first approach is to use filtering techniques, such as the Kalman filter [12] and particle filters [13], to limit the effects of the noise. The second approach is to apply noise cancellation methods using GPS, assisted GPS or Wi-Fi [6], [7], [14]. A key difference is that our technique is more flexible since we emphasize on the related locations of each user rather than the physical locations.

Finally, Priyantha et al. [15] proposed an anchor-free localization (AFL) algorithm to resolve the localization problem in sensor networks. The goal of [15] 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

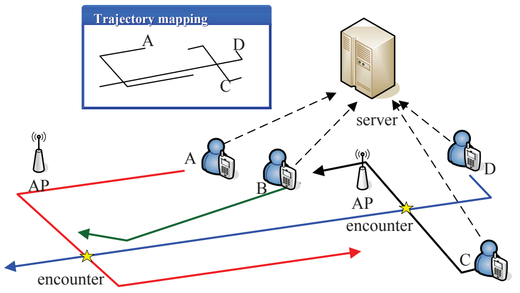

Fig. 1. System model.

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 2 basic components. (1) A server and (2) smartphones. The server collects users' data and uses that information to build the trajectory map, and also provides additional services based on the constructed map, such as routing. Users report their trails and encounters to the server.

Besides these two basic components, our solution takes advantage of any AP, such as a WiFi AP, that a user 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 users 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.

We assume that 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 built that is into the smartphone [1].

The smartphone will periodically report the *movement list* and *encounter list* to the server via a 3G or 4G connection [16]. The movement list consists of a series of displacement 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:

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

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].

TABLE I
Table of Notation

L	Reported displacement
$k(t)$	Systematic error in displacement measurement
l	Real displacement
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

B. Existing Noise Cancellation Solutions

The general idea behind error cancellation in prior work [1] is that each user's noise can be corrected by some physical references. 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. 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. Since [1] considered that the amount of noise is proportional to time, we can also proportionally use the instantaneous correcting vector to adjust the historical trail.

This solution is inadequate due to the following reasons. Firstly, [1] only use false negative encounters in the error cancellation. By incorporating both false positive and negative encounters, we can improve the error cancellation. Secondly, the direction of the adjustment vector that is used is the same in all of the data. This is inaccurate since the compass will also have a systematic error, which should be accounted for.

C. Challenges

In order to correct measurement errors, we need to solve three issues: (1) every user's accelerometer and compass may exhibit different error parameters. Without knowing the values, we cannot correct the trails since we cannot determine the extent of the error of each user. Moreover, users' error parameters may slightly change with time. (2) there are two types of encounter errors, 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. 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. (3) 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 that the users may pass the intersection at different time. Hence, we should only consider the trails with definite encounters in the false positive case.

IV. Solution Framework

A. Overview

A key feature of our solution is that when two users meet, they will independently report their encounter with the other to the server; although the absolute error of devices may be large, if the relative errors are small, the cooperative trajectory mapping system can still work well. Our proposed method consists of 3 steps: (1)each smartphone will apply the Kalman filter [17] to eliminate random noise. Considering that the Kalman filter only requires an individual user's moving pattern, 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 server; (2) at the server side, the server will first detect any false positive and false negative cases by using the reported data. Then, the server will slightly adjust the reported locations, letting the relative error between users becoming small; during the path correction, it 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 error parameters of each user. 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. Steps (2) and (3) are our proposed *accessorial anchor-based error reducing algorithm* (AAER).

Algorithm 1 The AAER algorithm

```
1: for Each sampling time $T$ do
2:     Find of false positive and false negative by encounters
3:     Estimate current error parameter by AP encounter
4:     Use HBMS algorithm to adjust the reported trajectories
5:     Record the adjusted positions
```

Algorithm 2 The HBMS algorithm

```
1: Verify previous direction hypothesis by current encounter
2: Set up new false positive direction hypothesis
3: for Each hypothesis do
4:     for $i = 1 : G$ (recursively reposition, Section V.C.2)) do
5:         Compute adjustment force $\frac{F}{G}$
6:         Move estimation position and compute the adjustment force
7:     Record the positions with minimal adjustment force
8: Update current error parameter
```

B. Accessorial anchor-based error reducing algorithm

We use APs to increase the chance of encounters. Algorithm 1 shows the procedure of AAER. The details about line 2 of Algorithm 1 can be found in Section V: 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 V: C. 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. 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.

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

V. Technical Details

In order to use AAER, we first need to determine the noise model. Then, we will discuss several auxiliary functions.

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. We use $k(t)$ to represent a systematic error which may slightly change over a long period of time. The reported displacement, L, can be represented as $L = l + k(t) \times l + \lambda$. In the same way, the readings from an electronic compass, Θ, can be regarded as $\Theta = \theta + \Delta\theta(t) + \delta$.

To illustrate the effect of 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: $\overrightarrow{E}$: $|\overrightarrow{E}| = l \times \sqrt{p^2 - 2p\cos(\Delta\theta) + 1}$, The direction is ψ: $\cos(\psi) = \frac{p\cos(\Delta\theta)-1}{\sqrt{p^2-2p\cos(\Delta\theta)+1}}$. If one of the noise parameters is relatively large, both of the errors cannot be neglected.

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 the encounter readings with 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 since they are not within the bluetooth sensor range of each other. We temporarily use the sensor radius to represent the actual distance.

However, there is a special case when dealing with the false positive error: the server needs to determine whether two spatial encounter trails have physically encountered each other at some point in time. 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 this section.

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. So, there are two trajectory boundary functions:

$$\frac{x - x_b}{x_e - x_b} = \frac{y - y_b}{y_e - y_b} = \frac{t - (t_e - d/S_{max})}{d/S_{max}} \tag{2}$$

$$\frac{x - x_b}{x_e - x_b} = \frac{y - y_b}{y_e - y_b} = \frac{t - t_b}{d/S_{max}} \tag{3}$$

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 position 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, whether a piece of an encounter record will be generated or not can be determined by the truth value of following formula:$R^2(a^2 + c^2) - (ad - bc)^2 \geq 0$ where,

$$a = \frac{(x(A, t_e) - x(A, t_b)) - (x(B, t_e) - x(B, t_b))}{t_e - t_b} \tag{4}$$

$$b = -a \times t_b + x(A, t_b) - x(B, t_b) \tag{5}$$

$$c = \frac{(y(A, t_e) - y(A, t_b)) - (y(B, t_e) - y(B, t_b))}{t_e - t_b} \tag{6}$$

$$d = -c \times t_b + y(A, t_b) - y(B, t_b) \tag{7}$$

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 V: 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 V: 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 V: C-4. Then, we estimate the optimal position, recursively. 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 V: 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:$\overrightarrow{F_{ij}^-} = \overrightarrow{u} \times (\widetilde{d_{ij}} - d_{ij})$, where $\overrightarrow{u}$ is the unit vector from location i to j.

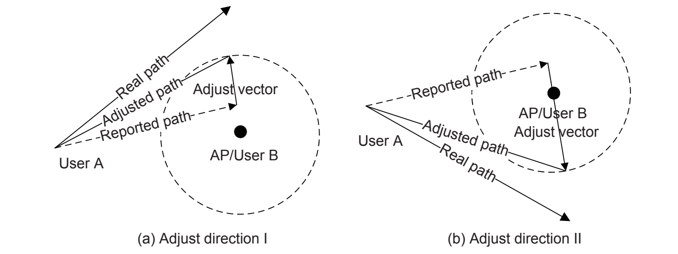

Fig. 2. The possible adjustments in the false positive case. The server receives a reported trajectory from user A and detects that the false positive case has happened. Since the server cannot get any information about the real path (the error-free path), the server needs to check both adjustment directions.

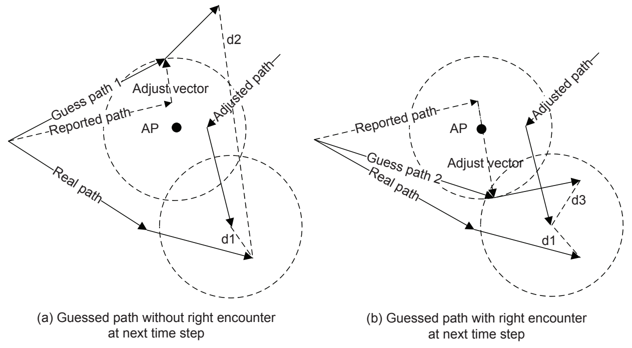

Fig. 3. The verification of a hypothesis.

In the false positive case, we cannot obtain the real distance d_{ij} or the adjustment direction. As shown in Fig. 2, 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 $\overrightarrow{F_{ij}^+} = \pm\overrightarrow{u} \times (R - d_{ij})$.

The synthesized force of a node in a hypothesis is the sum of the forces gotten from all of the nodes' neighbors: $\overrightarrow{F_i} = \Sigma_j(\overrightarrow{F_{ij}^+} + \overrightarrow{F_{ij}^-})$. The total force of a map is given by:$\overrightarrow{F} = \Sigma_i\overrightarrow{F_i}$.

2) Recursively Reposition: changing one user's path will also impact the historic path of other users. As a result, the estimated position adjustment should be accomplished by several smaller adjustments. In each adjustment period, we let each estimated position only move $\frac{\overrightarrow{F_i}}{G}$, where G is the server-specified granularity. A small G means a more accurate map, but it also entails higher computing complexity and more time to construct the map. The adjustment process stops when the total forces of the map stop decreasing.

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. 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.

4) Error Parameter Estimation: 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 adjustment's ratio 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

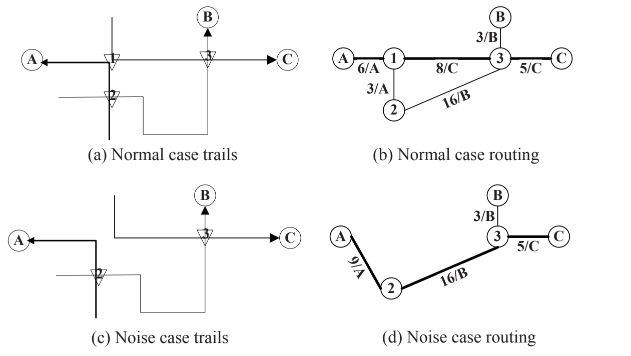

Fig. 4. The effect of measurement error during routing. Figs. (a) and (c) represent the reported trajectories. Figs. (b) and (c) represent the corresponding routing graph. The digital stands for the length of a path and the letter indicates who reported the path.

times. Then, we calculate the ratio of the adjusted amounts to the corresponding time interval and store the ratio. At the next time step, we first use the stored parameter to coarsely adjust the trails, and then we apply the HBMS.

D. *Historical Error Cancellation*

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, as mentioned in the background part of our paper. However, this adjustment is not true if the compass contains systematic errors. In our solution, 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 rotate the reported trails with the computed error angle and then shrink or expand the length of each reported displacement.

VI. Routing Application: Friend Locator

Friend locator is a representative 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 of the reported trails from several users whose trajectories have spatial intersections with the others. The implementation details are presented by Algorithms 3 and 4.

However, the quality of this application is restricted by the measurement error: those error paths may cause both false positive intersections and false negative intersections. Fig. 4 shows the effect of the measurement error on the length of the routine result. We have 3 users, A, B, and C. The arrow lines represent the paths of users, and the triangles between the paths are the spatial intersections. Fig. 4 (a) is the noise-free example, and Fig. 4 (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. However, assume that there is an error undetected in Fig. 4 (c), such that the intersection 1 is missing. The shortest path will become Fig. 4 (d) with its distance changed from 19 to 30.

Algorithm 3 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 associated with the shortest path
9: Return a list consisted of moving directions and displacements
```

Algorithm 4 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:         Go back to last passed intersection; 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
```

VII. Performance Analysis and Evaluation

A. *Evaluation metric*

We use Matlab to perform our simulation experiments. The metric we used to evaluate calls *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 send the routine request again.

$$Inaccuracy = \left| \frac{\widetilde{d_{ij}} - d_{ij}}{d_{ij}} \right|, \tag{8}$$

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

B. *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. 5. 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.

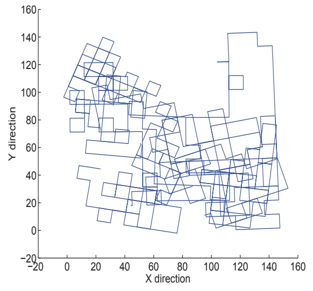

Fig. 5. Data illustration.

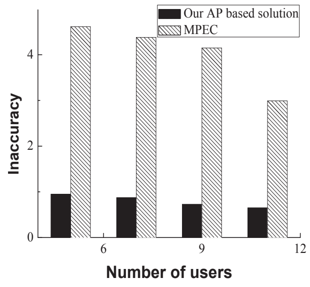

Fig. 6. User density vs. inaccuracy.

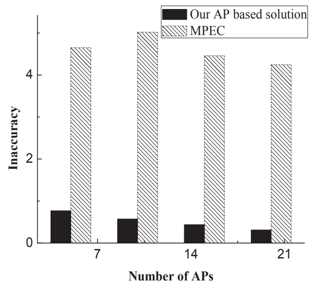

Fig. 7. AP density vs. inaccuracy.

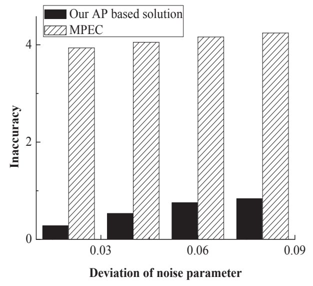

Fig. 8. Error vs. inaccuracy.

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 user density, as shown in Fig. 6. We test 5 to 11 users in the grid map. Although 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 second 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. 7 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. 8 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. Our future work will consider techniques to determine the presence of a malicious user, who always reports wrong trails, from the normal user, who has relatively large amounts of noise. In addition, we plan to build the system on a real platform for evaluation.

Acknowledgment

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

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] P. Bahl and V. Padmanabhan, "RADAR: An in-building RF-based user location and tracking system," in *IEEE INFOCOM*, 2000.
[7] Y. Cheng, Y. Chawathe, A. LaMarca, and J. Krumm, "Accuracy characterization for metropolitan-scale Wi-Fi localization," in *ACM MobiSys*, 2005.
[8] 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.
[9] A. Thiagarajan, J. Biagioni, T. Gerlich, and J. Eriksson, "Cooperative transit tracking using smart-phones," in *ACM SenSys*, 2010.
[10] J. Farrell and M. Barth, *The global positioning system and inertial navigation*. McGraw-Hill Professional, 1999.
[11] P. Gilliéron, D. Buchel, I. Spassov, and B. Merminod, "Indoor navigation performance analysis," in *ENC GNSS*, 2004.
[12] G. Welch and G. Bishop, "An introduction to the Kalman filter," *University of North Carolina at Chapel Hill, Chapel Hill, NC*, 1995.
[13] 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.
[14] M. Youssef, A. Youssef, C. Rieger, U. Shankar, and A. Agrawala, "Pinpoint: An asynchronous time-based location determination system," in *ACM MobiSys*, 2006.
[15] N. Priyantha, H. Balakrishnan, E. Demaine, and S. Teller, "Anchor-free distributed localization in sensor networks," in *ACM SenSys*, 2003.
[16] 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.
[17] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," Chapel Hill, NC, USA, Tech. Rep., 1995.