Detecting and Correcting WiFi Positioning Errors

Yuki Tsuda, Quan Kong, Takuya Maekawa
{tsuda.yuuki,kong.quan,maekawa}@ist.osaka-u.ac.jp
Graduate School of Information Science and Technology
Osaka University
2-1 Yamadaoka, Suita, Osaka, 565-0871, JAPAN

ABSTRACT

Recent advances in GPS and WiFi-based positioning technologies for mobile phones have triggered many location-based services. However, GPS positioning quickly drains a phone's battery and cannot be used indoors. On the other hand, WiFi positioning provides energy-efficient indoor and outdoor positioning with reasonable accuracy. However, WiFi positioning sometimes makes large errors caused by various reasons, e.g., the movement of reference WiFi access points. In this paper we attempt to detect and correct such errors automatically by performing outlier detection in time series. So, we solve this problem by comparing a user's current measurement at time T with her coordinate point at time T predicted from her past coordinate history, and judging whether the current measurement is correct or not by computing the distance between the measurement location and the predicted location. However, it is difficult to predict the user's coordinates accurately with a single prediction method (predictor) because the user's context (e.g., migration speed and sparseness of past coordinates) greatly affects predictor performance. We thus design a context-aware error detection method by employing an ensemble of predictors that have different strengths and weaknesses.

Categories and Subject Descriptors: H.3.4 Information storage and retrieval: Systems and software; I.5.2 Design Methodology: Pattern analysis.
General Terms: Design, Experimentation.
Keywords: Location sensing; reliability; pattern classification.

INTRODUCTION

The recent proliferation of positioning technologies including Global Positioning System (GPS) and WiFi-based positioning has triggered many location-based applications on mobile phones. The applications include tourist information systems, location sharing with friends, and lifelogging. We can also embed measured coordinate data in a photograph or a microblog post (e.g., tweet) as metadata. While GPS provides very accurate positioning (approximately 10 meter error) outdoors, it has the following drawbacks. (1) GPS quickly drains a phone's battery [13]. This is a very serious problem for mobile phones, which have little battery capacity. (2) Because GPS employs GPS satellites, it is not available indoors.

On the other hand, while WiFi positioning is somewhat less accurate (20-40 meter errors [22]) than GPS, this method has the following advantages. (1) Its battery consumption is much lower than that of GPS. Fig. 1 shows the power consumption of a phone battery over time. This graph tells us that GPS consumes much more power than WiFi positioning. (2) Because WiFi positioning employs signals emitted from WiFi access points (APs), it can measure coordinates even when the mobile phone is indoors. (3) This method can measure the coordinates of a device that is not equipped with a GPS receiver, e.g., laptop PCs and some tablets. Recently, W3C Geolocation API in HTML5 has been established, and WiFi-based location information is being widely used in web browsing on laptop PCs.

On the basis of the above facts, we can reduce a phone's battery consumption mainly by employing the WiFi positioning. Also, we can capture the phone's location whether the phone is indoors or outdoors. Here WiFi positioning has been developed based on fingerprinting techniques. Fingerprinting employs a training phase in which WiFi signals (i.e., the unique MAC addresses of APs and the signal strengths from APs) are observed at known coordinates. A set of APs and their signal strengths become a *fingerprint* that is unique to those coordinates. The fingerprints are collected using a *war-driving* technique [15] and stored in a WiFi positioning database. In the positioning phase, the observed WiFi signals at unknown coordinates are compared with the stored fingerprints to determine the closest match.

As mentioned above, basically, the WiFi positioning error was reported to be about 20 to 40 meters. However, the WiFi positioning errors are sometimes much larger than the basic errors (from a few hundred meters to tens of kilometers). This kind of error is mainly caused by the following reasons. (1) The movements of WiFi APs over time cause the positioning errors. This is because people move into and out of apartments, homes, and offices with APs. Also, recently, WiFi mobile hotspot devices have proliferated. The devices are equipped with WiFi access points and mobile Internet access functionalities. WiFi-enabled equipment can connect with the Internet via these devices. In addition, some express trains have been equipped with WiFi APs. (In the following, we refer to such access points as anomalous access points.) Because the WiFi positioning determines a phone's coordinates based on fingerprints collected in advance, anomalous APs

UbiComp'13, September 8–12, 2013, Zurich, Switzerland.

ACM 978-1-4503-1770-2/13/09...$15.00.
http://dx.doi.org/10.1145/2493432.2493460

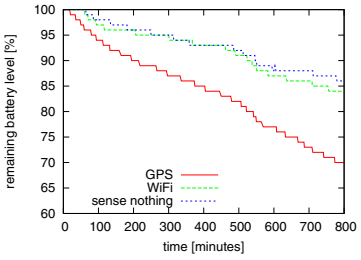

Figure 1. Mobile phone power consumption during continuous sensing. (GPS or WiFi roughly per minute with Google Nexus One)

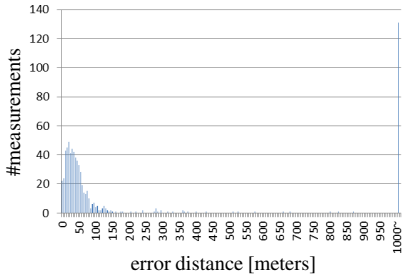

Figure 2. Distribution of WiFi positioning errors.

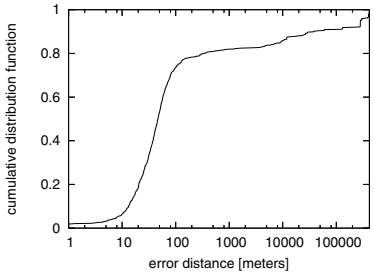

Figure 3. Cumulative distribution function (CDF) of WiFi positioning.

have negative effects on the positioning method when they move. (2) The WiFi positioning accuracies also depend on the density of the APs around the phone. When the number of the APs is small, the estimated position may be inaccurate. (3) Although it depends on the coverage area of the WiFi positioning provider, the estimated position may have an error when the density of fingerprints around the phone is low.

The WiFi positioning errors do not occur very frequently (9.8 % of measurements in our experiment when subjects walked around an urban city). However, because people spend most of their days indoors, their coordinates are usually measured with WiFi positioning. So, the effect of the errors is significant in our daily lives. That is, we consider it very important to cope with WiFi positioning errors. In this work, we try to judge automatically whether or not a measured coordinate point with WiFi is correct (i.e., detect a positioning error). This also enables us to achieve energy-efficient hybrid positioning. That is, a phone mainly employs WiFi positioning and automatically switches to GPS only when a coordinate point measured with WiFi is estimated to include an error.

We can simply regard the problem (detecting positioning errors) as an outlier detection problem in a time series. That is, an outlying measurement is detected based on past measurements. We can solve this problem by estimating a current measurement with past measurements and comparing the estimated value with an actual current measurement [18, 8, 6]. When the estimated value and the actual measurement differ greatly, the actual measurement is determined to be an outlier (error). Assume that we predict a current coordinate point from past coordinate points (trajectory) by using the Kalman filter [10], which is usually used to estimate current coordinates. That is, we compare a coordinate point predicted by the filter with a coordinate point actually measured with WiFi positioning, and judge whether or not the actual measurement is correct. This approach may work well when sufficient numbers of past measurements are available. However, sometimes this predictor may deal badly with real WiFi positioning data. For example, because WiFi positioning cannot measure a phone's coordinates everywhere (e.g., inside a tunnel), we sometimes prepare only a very sparse trajectory. In this case, the reliability of the prediction may be low. In our experiment described below, we found that the phone's coordinates could not be measured for more than five minutes an average of 1.9 times an hour. Also, several applications do not assume continuous positioning. For example, assume that a user takes a photo when she is at a shop, and her coordinate point at that time is embedded in the photo. In such case, her past trajectory is not available because her phone's coordinate point is measured only when she takes a photo or tweets.

Here we assume another predictor that outputs a coordinate point simply by querying another WiFi positioning provider with WiFi scan data. This predictor may output accurate coordinates even when we have no trajectory. However, when there are few WiFi APs around the phone, the reliability of the predictor may be low. This predictor is also affected by anomalous APs. As above, predictor reliability depends on a phone's context, e.g., sparseness of trajectory and the number of APs around the phone. (In our evaluation, we confirm that we cannot accurately detect WiFi positioning errors with a single predictor in various situations.) That is, we should design a context-aware error detection method that takes context into account. In this paper, we propose a context-aware method for detecting WiFi location errors by employing the phone's context data and combining several location predictors with different characteristics, e.g., a predictor that works well even when histories (past coordinates and scans) are unavailable and a predictor that works well even when there are few APs around the phone. That is, our method detects outliers in a *context-aware manner*. With this method, we can save a phone's energy by mainly using WiFi positioning and switching to GPS only when coordinates measured with WiFi are found to include errors. However, as mentioned above, people are only covered by GPS for short period during their daily lives. Moreover, there are many devices with no GPS receivers. In this paper, we also design a context-aware method for predicting correct coordinates without GPS by employing an ensemble of the above predictors.

In the rest of this paper, we first introduce our data set and the findings it provided. Then we propose methods for detecting WiFi location errors and estimating correct coordinates. After that we evaluate our methods with the data set. To our knowledge, this is the first work to attempt to detect and correct WiFi positioning errors in a context aware manner.

DATA SET

Overview

Several participants carried mobile phones (Google Nexus One) in several situations, and the phones collected coordi-

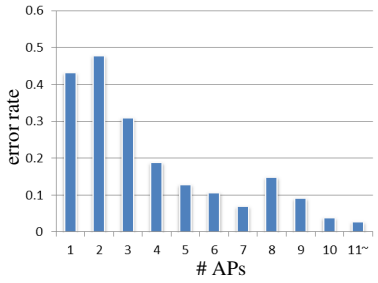

Figure 4. Relationship between error rate and # APs around phones.

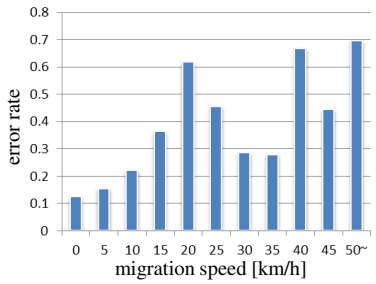

Figure 5. Relationship between error rate and phone migration speed.

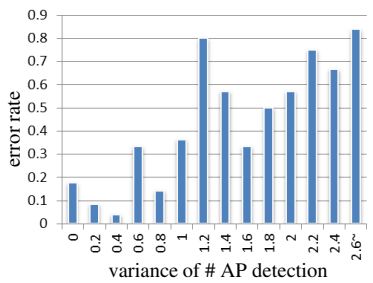

Figure 6. Relationship between error rate and variance of # AP detection in five-minute scans.

nate points (latitudes, longitudes, and timestamps) with WiFi positioning every minute. The phone simply employed native Android APIs to measure their coordinates. (The APIs measure coordinates by combining WiFi and GSM data.) At the same time, the phone saved a WiFi scan (received signal strengths and MAC addresses of APs). The phone also continuously measured GPS coordinate points, which will be used as ground truth in the evaluation section. (We judge whether or not a WiFi coordinate point is correct with the data.) Note that GPS coordinates cannot be obtained indoors. To obtain ground truth indoors, we employed the experience sampling method (ESM) [16]. The ESM was originally used to intentionally interrupt an experimental participant in order to have her make notes about her current situation. In this experiment, an application on each phone asks a participant to input her current location with a Google Maps type interface when the phone obtains a WiFi coordinate point but cannot obtain any GPS coordinates at the same time. We have used this method to collect the following three kinds of trajectory data.

- Journey data set: This set was collected when participants commuted to work or school by trains, buses, and/or on foot. This set also includes trajectories collected when the participants went out to do something. We obtained a total of 11 trajectories from five participants.
- City data set: A participant walked around an urban city. The participant usually walked through buildings and stations, and visited shops.
- Underground data set: A participant walked around the underground parts of cities.

Note that these data sets do not include any two identical trajectories, i.e., trajectories whose start and end points are not identical in the data sets. Also, in the experiment, we asked the experimental participants to carry the mobile phones freely. In many cases, male participants carried the phones in their pants pockets, and female participants carried the phones in their bags. In addition, when the phone could not obtain any GPS coordinates, we employed the ESM. The ratio of GPS outage time was about 49% because our data set includes 'underground' data, i.e., 49% of ground truth data was obtained by the ESM.

Table 1 provides an overview of the above data sets. The error rate of the journey data set is higher than that of the other sets because the phones could not capture WiFi signals sufficiently well when their users were on trains/metros. Fig. 2 also shows the distribution of WiFi positioning errors. An error corresponds to the distance between a WiFi coordinate point and a GPS coordinate point (or an ESM answer) obtained simultaneously. In addition, Fig. 3 shows the cumulative distribution function (CDF) of WiFi positioning. Small errors (smaller than about 150 meters) may correspond to basic WiFi positioning errors. We want to detect errors larger than these. (In our evaluation, we determined the smallest errors that our method could successfully detect.) The figures show the wide distribution of positioning errors greater than 150 meters. Also, the ratio of small errors (150 to 5000 meters), which may be difficult to distinguish from correct measurements, to all measurements is about 13%.

Table 1. Overview of our data set.

	journey	city	underground	overall
# trajectories	11	10	10	31
average trajectory duration (min)	84.8	31.6	26.7	48.9
# WiFi measurements per hour	24.8	44.8	39.1	31.5
error rate (> 150 m)	32.8%	9.8%	20.3%	22.4%

Characteristics of Data

Here we investigate how a phone's context affects WiFi positioning errors. Fig. 4 shows the relationship between the number of WiFi APs the phones detected when their coordinates were measured with WiFi positioning and the ratio of incorrect measurements. (Here we regard measurements whose errors are larger than 150 meters as being incorrect.) As shown in Fig. 4, when there are small numbers of WiFi APs around a phone, WiFi positioning cannot measure the phone's coordinates accurately.

Fig. 5 shows the relationship between the migration speed of a user (phone) and the ratio of incorrect measurements. (GPS measurements include the phone's speed.) As shown in the graph, when a phone moves fast, the WiFi positioning method cannot measure the phone's precise coordinates. This may be because the phone sometimes cannot successfully receive signals from APs when it moves fast.

As mentioned in the *Introduction* section, mobile WiFi hotspots cause WiFi positioning errors. If a person who has a mobile hotspot rides the same bus or in the same train carriage as a user, for example, the hotspot may affect the user's

WiFi positioning results. In this case, the user's phone continuously captures signals from the mobile hotspot. On the other hand, because the phone moves quickly, it may not continuously capture signals from a certain static AP (i.e., AP placed at a house or office). We focus on five minutes of WiFi scan data and count the number of scans that include each AP. The number may be large for mobile APs and small for static APs. Fig. 6 shows the relationship between the variance of the number and the ratio of incorrect measurements. When the variance is large, a mobile hotspot may be close to the user. And then, the ratio of incorrect measurements increases.

METHOD: DETECTING WIFI POSITIONING ERRORS

We try to detect WiFi positioning errors that have the above characteristics. As mentioned in the *Introduction* section, we try to achieve context-aware error detection by employing an ensemble of predictors that have different characteristics. To realize context awareness, we also utilize a phone's context data that may affect the performance of the predictors and WiFi positioning itself (e.g., # APs around the phone).

Problem formulation

We employ supervised machine learning techniques to detect WiFi positioning errors. Assume that, at current time T, a phone's WiFi coordinate point (x_T, y_T) is obtained, and the phone obtains a WiFi scan ws_T at the same time (time T). The phone also has WiFi coordinate histories (x_1, y_1),$\cdots$, (x_{T-1}, y_{T-1}) and WiFi scans obtained at each time slice $ws_1, \cdots, ws_{T-1}$. With the coordinates and WiFi scans from time 1 to time T, we classify the coordinate point (x_T, y_T) in an appropriate class, i.e., 'correct' or 'incorrect' class. Note that, when we want to employ supervised machine learning techniques, we should prepare a teaching signal for each measurement. We obtain a teaching signal from the ground truth (GPS coordinate point or ESM answer), by computing the distance between a WiFi coordinate point and the ground truth. When the distance is larger than a given threshold ϵ, we assume that the WiFi coordinate point is incorrect. We investigate the threshold value in the evaluation section. Note that we assume that a WiFi positioning provider only provides a service that returns (x_T, y_T) for a query (a WiFi scan). As with existing providers, the provider does not allow us to freely browse its very expensive WiFi fingerprint database.

Our ensemble error detection method

Assume that a predictor (e.g., a Kalman filter) estimates a coordinate point at time T (xn_T, yn_T). With the estimated coordinate point obtained from the predictor, we can construct a weak error detector that judges whether or not (x_T, y_T) is an error measurement, i.e., whether or not the distance between (x_T, y_T) and (xn_T, yn_T) is larger than the threshold ϵ, by employing the following equation.

$$h_n\big(p_T, pn_T\big) = \frac{1 - d\big(p_T, pn_T\big)/\epsilon}{1 + d\big(p_T, pn_T\big)/\epsilon},$$

where p_T is (x_T, y_T), pn_T is (xn_T, yn_T), and $d\big(p_T, pn_T\big)$ shows the distance between (x_T, y_T) and (xn_T, yn_T). We show some examples. When the distance is zero, the $h_n()$ value becomes 1. When the distance is equal to ϵ, the $h_n()$ value becomes 0. Also, when the distance approaches infinity, the $h_n()$ value approaches -1. We can judge whether or not (x_T, y_T) is an error measurement with $C_n^{WiFi} = \mathrm{sign}\big[h_n(p_T, pn_T)\big]$. That is, when C_n^{WiFi} is -1, (x_T, y_T) is determined as an error measurement. On the other hand, when C_n^{WiFi} is $+1$, (x_T, y_T) is determined as a correct measurement. Note that, when we want to use a small threshold value as the ϵ value in the above equation, we confirmed that many error detectors always output -1. In practice, using $\alpha\epsilon$ ($\alpha > 1$. In our implementation, $\alpha = 3$ and $\epsilon = 150$.) on behalf of ϵ in the above equation works well because many errors are much larger than the threshold.

In this paper, we propose an ensemble error detection architecture that can take a phone's context into consideration as shown in Fig. 7. We assume that the method runs on a server computer. We will discuss it in *Related work* section. The ensemble error detection is defined as the following equation.

$$H\big(p_T, s_T\big) = \mathrm{sign}\Big[\sum_{n=1}^{N} \alpha_n p(C^n = 1|s_T) h_n\big(p_T, pn_T\big)\Big],$$

where s_T is the phone's context features computed from $ws_1, \cdots, ws_T$ and (x_1,y_1), $\cdots$, (x_T,y_T), N is the number of weak error detectors, and $C^n \in \{-1, +1\}$ shows whether the nth predictor's measurement (pn_T) is correct or not. Also, α_n is the weight of the nth weak detector. The weight corresponds to the error detection accuracy of the nth detector computed from training data with a cross validation approach. In addition, with $p(C^n = 1|s_T)$, we can incorporate the phone's context into the nth error detector. Note that $p(C^n = 1|s_T)$ is the probability with which the nth predictor's estimation (xn_T, yn_T) is correct when the phone's context feature vector s_T is given. That is, we estimate the reliability of the nth predictor by using the phone's context and incorporate it into our ensemble error detection approach. We can compute the probability as follows by using Bayes' theorem.

$$p(C^n = 1|s_T) = \frac{p(s_T|C^n=1)p(C^n=1)}{p(s_T)}$$
$$= \frac{p(s_T|C^n=1)p(C^n=1)}{p(s_T|C^n=1)p(C^n=1)+p(s_T|C^n=-1)p(C^n=-1)},$$

where $p(C^n)$ is the prior density of the nth detector's reliability computed from its predictor's estimations and the GPS ground truth in the training data, $p(s_T)$ is the prior density of the phone's context data s_T (evidence), and $p(s_T|C^n)$ is the likelihood of C^n for s_T. We employ a Gaussian Mixture Model (GMM) to model C^n. That is, we prepare two models (one for $C^n = 1$ and another for $C^n = -1$), and each model shows the distribution of context features s_t corresponding to its class (1 or -1). To construct the model, we estimate the parameters of the Gaussians by employing expectation maximization (EM) [4]. We can compute the likelihood of the GMM of $C^n = 1$ (or $C^n = -1$) for s_t as follows.

$$p(s_T|C^n) = \sum_i \pi_i \mathcal{N}(s_T, \mu_i, \Sigma_i),$$

where π_i is the mixture weight of the ith multivariate Gaussian distribution of the GMM, and μ_i and Σ_i are the mean

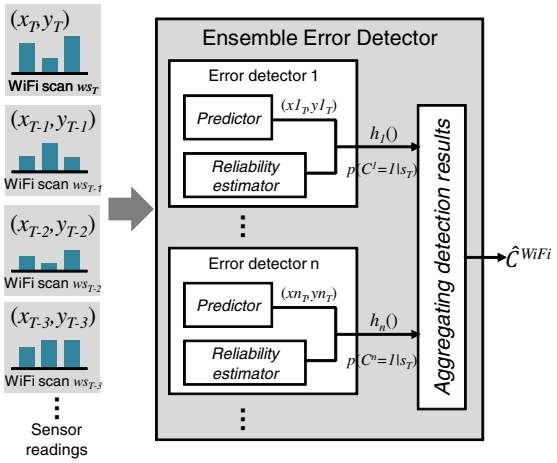

Figure 7. Overview of ensemble error detection.

vector and covariance matrix of the Gaussian distribution, respectively.

As above, we detect WiFi positioning errors by fusing each weak error detector's vote associated with its reliability computed from the context. When $H(p_T, s_T)$ is -1, we can say that p_T is an error measurement ('incorrect' class). Here we consider that s_T (esp. ws_T) also relates directly to p_T. So, in addition to the normal error detectors described in the next subsection, we prepare a special error detector that employs the following equation, and include that detector into our ensemble detector $H(p_T, s_T)$ in the same manner as the normal detectors.

$$h_{WiFi}(p_T, s_T) = 2p(C^{WiFi} = 1|s_T) - 1,$$

where $C^{WiFi} \in \{-1, +1\}$ shows whether the WiFi measurement p_T is correct or not. We estimate $p(C^{WiFi} = 1|s_T)$ based on the GMM approach in the same way as computing $p(C^n = 1|s_T)$. (When we train GMMs, we can know whether C^{WiFi} in the training data is 1 or -1 from the GPS ground truth.) This weak detector permits us to estimate the reliability of p_T solely from the phone's context.

Next, we describe the prepared (normal) predictors and context features.

Predictors

We prepare several predictors that have various characteristics to cope with a phone's various contexts. We introduce the predictors we prepared in this study.

Kalman filter

This predictor employs the Kalman filter [10]. The Kalman filter is usually used to track the trajectory of a moving object [19, 11]. The Kalman filter is based on a linear system and its algorithm works in a two-step process: prediction and correction. In the prediction phase, the filter estimates the current state variables ((xn_T, yn_T) in our case) based on state variables at the previous time step (time $T-1$). In this implementation, we simply assume that a phone moves at the same velocity as at the previous time step. In the correction phase, the estimated variables are updated by using an actual measurement (x_T, y_T). We employ the estimated variables (xn_T, yn_T) in the prediction phase as the predicted coordinate point of this predictor. This predictor may work well when the phone has a sufficient number of past coordinates.

Kalman filter (improved)

Our data sometimes include very large errors. That is, the distance between a measurement and an actual coordinate point is sometimes very large. Because the Kalman filter works recursively, i.e., the two prediction and correction processes are repeated every time step, a large error can greatly affect its future predictions. We cope with this problem simply by ignoring a measurement that is very far from its previous corrected coordinate point. That is, the filter does not take such measurements into account when it corrects the prediction.

Particle filter

Assume that a person gets out of a car and walks. Her migration speed may suddenly change when she leaves the car. To track such a trajectory, we employ a particle filter [5] that is usually used to estimate the states of non-linear systems. Its algorithm works in a three-step process: sampling, weight calculation, and resampling. In the sampling process, new particles are generated from particles at the previous time slice $(T-1)$ and are moved based on a motion model. The generated particles show estimations of coordinates at time T. In the weight calculation process, the particle weights are computed based on an importance function and a measurement at time T. Basically, particles that are close to the measurements and match the importance function have heavy weights. In the resampling process, particles with small weights are eliminated. The three processes are iterated. This predictor outputs an average over particles obtained in the sampling process as an estimated coordinate point. In the sampling process, we employ a bivariate Gaussian distribution as a motion model. Its mean corresponds to an extrapolated coordinate point at time T simply computed from a particle's speed and coordinates at time $T-1$. Also, its standard deviations correspond to the distance between the mean coordinate point and the particle's coordinate point at time $T-1$, and its covariances are zero. With this filter, we attempt to track the movement of a phone whose speed changes suddenly.

Second opinion

This predictor simply queries another WiFi location provider by using ws_T. In our implementation, we employ the Skyhook location service. This predictor works well even when the phone does not have any past coordinates. However, when there are few APs around the phone, the predictor may not output accurate coordinates. Moreover, this predictor is affected by anomalous APs.

Query modification

When a WiFi scan includes signals from mobile APs, the APs may have a negative effect on the location estimation with WiFi positioning. This predictor attempts to remove mobile

AP signals from WiFi scan data ws_T and queries the WiFi positioning service (Google location service) by using the modified scan data. This predictor is designed based on our assumption that small numbers of mobile APs are included in a WiFi scan. This predictor consists of four steps. We describe the method using an example. The algorithm is also given as Algorithm 1. Assume that WiFi scan ws_T includes (signals from) six WiFi APs one of which is a mobile AP.

(1) This method randomly splits the six APs into g sub-groups. (g is an odd number.) In this example, we assume g to be 3. Note that the number of elements in a sub-group does not exceed $\left\lceil \frac{\text{\# APs in } ws_T}{g} \right\rceil$.
(2) This method queries the WiFi positioning provider (Google Geolocation API) by using WiFi scan data in each sub-group. That is, when AP1 and AP2 are included in a sub-group, for example, the method queries this by using signal scan data of AP1 and AP2.
(3) Three data points (coordinates) are obtained from the three sub-groups. Because we assume that one mobile AP is included in ws_T, one data point of the three may have a large error. (The data point is obtained by using a sub-group that includes the mobile AP.) That is, the data point may be far from the other two points. This method clusters the three data points simply by using an agglomerative hierarchical clustering (bottom-up clustering) [9]. So, we first place each data point (coordinates) in a cluster. We then iteratively merge the two closest clusters. To define inter-cluster similarity, we employ the single-linkage method: $d(c_1, c_2) = \min_{(x_1,y_1)\in c_1,(x_2,y_2)\in c_2} d((x_1, y_1), (x_2, y_2))$, where c_1 and c_2 correspond to clusters, (x_1, y_1) and (x_2, y_2) correspond to data points, and $d((x_1, y_1), (x_2, y_2))$ shows the Euclidean distance between two data points. Note that we stop combining clusters when the number of data points in the largest cluster exceeds $\lfloor \frac{g}{2} \rfloor$. By doing so, we can group static APs that may be located close to each other. (That is, the largest cluster may not include mobile APs.)
(4) This method focuses on the largest cluster and queries the WiFi positioning provider by using WiFi scan data consisting of APs that are used to obtain the coordinates in the cluster. This predictor outputs the data point (coordinate point) obtained by using the query.

Note that the Google location service (Google Geolocation API), which is used in steps 2 and 4 of our implementation, does not accept WiFi scan data that consist of only one AP. That is, when g is three and the number of APs included in ws_T is smaller than six, for example, several sub-groups of APs created in the first step may include only one AP. In such cases, we should increase the number of APs included in the groups. That is, we add (signal data of) APs that are randomly selected from ws_T to each group until there are two APs in the group. Note that we do not select an AP that is already included in the group. Also, when the number of APs included in ws_T is smaller than four, the predictor does not output any results. This predictor may work well when large numbers of APs are included in ws_T even if small numbers of mobile APs are included in ws_T. However, when the numbers of APs included in ws_T is small, this predictor may not work well. We assume that values of g are 3 and 5, and prepare this type of predictor for each g value.

Algorithm 1 Prediction in 'Query modification' method

Input: WiFi scan ws, # sub-groups g
Output: estimated coordinate point (x, y)
1: $SG = Split(ws, g)$ {Split APs in ws into g sub-groups.}
2: $C = \emptyset$ {Initialize a set of clusters C.}
3: **for** each sub-group of APs $sg_i \in SG$ **do**
4: $(x_i, y_i) = GetCoordinates(sg_i)$ {Query a WiFi positioning service using sg_i.}
5: Turn (x_i, y_i) into a singleton cluster c_i and add it to C
6: **end for**
7: **while** # elements of the largest cluster $\leq \lfloor \frac{g}{2} \rfloor$ **do**
8: **for** each pair of clusters c_1 and $c_2 \in C$ **do**
9: Compute distance between clusters $d(c_1, c_2)$
10: **end for**
11: Merge the pair of clusters with the smallest distance
12: **end while**
13: Make sg_{max} from the largest cluster c_{max} {Find sgs that are used to obtain coordinates in c_{max}, and construct a new group of APs sg_{max} from the sgs. }
14: $(x, y) = GetCoordinates(sg_{max})$
15: **return** (x, y)

Context features

We prepare the following context features to capture context information of the phone, and construct a vector (s_T) concatenating the feature values.

- Time interval: This feature corresponds to the time interval between ws_{T-1} and ws_T. The time interval is very large when the phone cannot obtain its coordinates for a long time. And then, the performance of several predictors (Kalman and particle filters) becomes poor.
- Migration speed and distance: We employ the phone's migration speed computed from (x_{T-1}, y_{T-1}) and (x_T, y_T) as a feature. We also employ the distance between (x_{T-1}, y_{T-1}) and (x_T, y_T). When these values are very large, the coordinate point at time T may be incorrect.
- Variance of speed and average speed: We focus on the n latest coordinates $(x_{T-n-1}, y_{T-n-1}), \cdots, (x_T, y_T)$. We compute the phone's speed by using pairs of adjacent coordinate points (i.e., (x_t, y_t) and (x_{t+1}, y_{t+1})). We employ the variance of the speed as a feature. This value becomes large when the phone's migration speed changes. And then, the performance of the Kalman filter based predictors worsens. We also employ an average speed over the n latest coordinates.
- Number of APs: This feature corresponds to the number of APs included in ws_T. The performance of several predictors ('Second opinion' and 'Query modification' in addition to 'WiFi positioning' that employs Google Android APIs) is related to this feature value.
- Average signal strength and variance of signal strength: We employ an average signal strength over the APs included in ws_T as a feature. The feature may relate to signal quality around the phone. We also employ the variance of the signal strength.
- Features related to mobile hotspot: If a person who has a mobile hotspot rides the same bus or in the same train carriage as the phone user, for example, the hotspot may affect

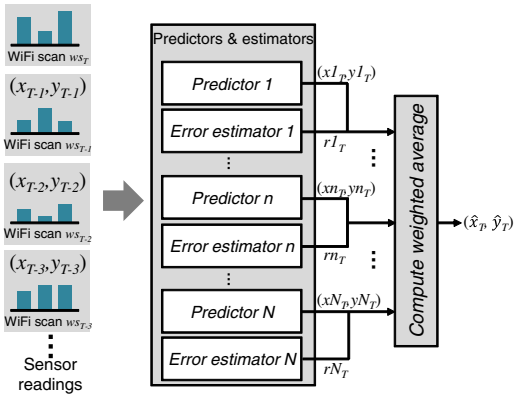

Figure 8. Overview of correct coordinate prediction.

the phone's WiFi positioning results. In this case, the phone continuously captures signals from the mobile hotspot. On the other hand, because the phone moves quickly, it cannot detect signals from a static AP (i.e., AP placed in a house or office) at time T that was detected at time $T-1$. We compute the ratio of the number of APs that are detected at *both* $T-1$ and T to the total number of APs detected at $T-1$ and T. And we employ the computed value as a feature. We also compute a feature related to mobile hotspots from $ws_{T-n-1}, \cdots, ws_T$. For each AP, we count the number of scans that include the AP. We employ the variance of the number as a feature. When a mobile hotspot user rides the same bus or in the same train carriage as the phone user, the value becomes large as shown in Fig. 6.

In our implementation related to context feature extraction, we set n as five. Note that WiFi positioning providers also provide an estimated positioning error. However, because it was not reliable in our investigation, we did not use it as a feature.

METHOD: CORRECTING WIFI COORDINATES

When we determine a phone's coordinate point to be incorrect, we can obtain a correct value by turning on a GPS receiver if the phone is outdoors. When the phone remains indoors, we should estimate its correct coordinates without GPS. Here we briefly explain our error correction method because this paper mainly focuses on detecting WiFi positioning errors. Fig. 8 shows an overview of our method. With this method, we employ the predictors prepared for our error detection method. We also prepare an error estimator for each predictor. The estimator estimates an error (the distance between an actual coordinate point and coordinate estimated by the predictor) of the corresponding predictor by using the context features computed in the *Context features* section because we consider predictor reliability to be related to the phone's context. We construct each error estimator by using the sequential minimal optimization (SMO) algorithm for regression [20]. As with the error detection, we train the estimator by using GPS and ESM ground truth. Note that, when we train the estimator, an error is capped at 1000 meters to prevent the regression model from (over) fitting to large errors.

Each predictor outputs the phone's coordinates (xn_T, yn_T). The estimator corresponding to the predictor outputs the error of the coordinates at time T. We assume that the inverse of the error shows the reliability of the predicted coordinates rn_T. By using the predicted coordinates and their reliabilities, we compute the weighted average of the coordinates with $(\hat{x}_T, \hat{y}_T) = \frac{\sum_{n=1}^{N} w_n rn_T(xn_T, yn_T)}{\sum_{n=1}^{N} w_n rn_T}$ [14]. Note that w_n is the weight of the nth predictor. The weight corresponds to the average accuracy (precision within 150 meters) for the nth predictor computed from training data. The computed coordinates $(\hat{x}_T, \hat{y}_T)$ correspond to the output of this method.

EVALUATION

Evaluation Methodology: Detecting WiFi Positioning Errors

We employ a leave-one-session-out cross validation to evaluate our error detection method. (A session corresponds to a trajectory obtained in the *Data set* section.) That is, we detect errors included in a trajectory by using an error detection model (classifier) trained with the other thirty trajectories. To investigate the effectiveness of our proposed method, we test the following naive methods in addition to our method.

- Kalman filter: This method simply constructs a classifier (C4.5 decision tree implemented in [21]) that employs $h_n()$ computed from (x_T, y_T) and coordinates at time T estimated by the Kalman filter as a feature. It classifies the feature into the *correct* or *incorrect* class. The following methods also employ the predictors prepared in the *Predictors* section.
- Kalman filter (improved)
- Particle filter
- Second opinion
- Query modification (g **=3 and 5)**
- Context: This method simply constructs a C4.5 decision tree classifier by using context features.

We evaluate the classification performance of the methods by using precision, recall, and F-measure calculated based on the classification results at each time slice. ($F\text{-}measure = \frac{2 \cdot precision \cdot recall}{precision + recall}$)

Results: Detecting WiFi Positioning Errors

Results of naive methods

Fig. 9 shows average F-measures for the correct and incorrect results obtained with each method. (The average for the three data sets.) Here we assumed that WiFi coordinates whose errors are larger than 150 meters are *incorrect*. We first focus on the Kalman filter, Kalman filter (improved), and Particle filter methods that utilize past trajectories. The result with the Kalman filter method was very poor (44.7%). This may be because there are fewer 'incorrect' instances than 'correct' instances, and so the classifier may ignore the 'incorrect' instances. To cope with the problem, we performed locally weighted learning (LWL), where the *incorrect* instances were weighted as three times heavier than the *correct* instances. The results are also shown in Fig. 9. In addition Fig. 10 shows classification accuracies for each data set. As shown in the figures, the Particle filter method was the best among

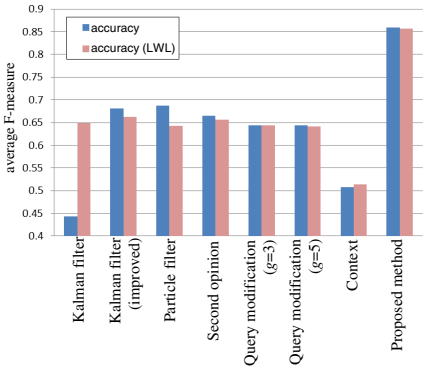

Figure 9. Accuracies (average F-measure) of naive and proposed methods.

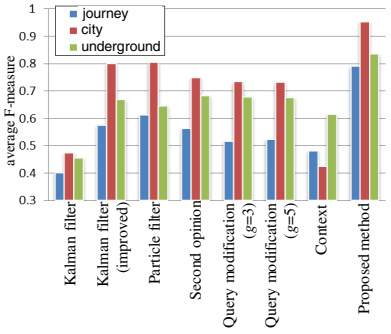

Figure 10. Accuracies of naive and proposed methods for each data set.

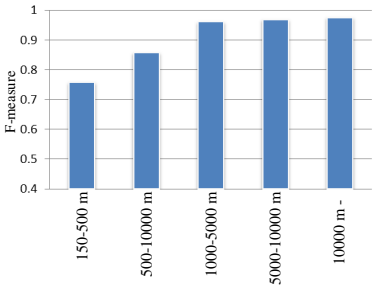

Figure 11. Accuracies for 'incorrect' measurements aggregated for their error distances.

the three methods, and achieved better results related to the journey data set. This may be because the method can track a mobile phone even if its migration speed changes frequently. (When a person rides on a train, bus, or metro, her speed changes frequently.) Here we investigate why these methods could not achieve very high accuracies with the journey and underground data sets. We found situations where the methods sometimes could not accurately track the participants. The examples are situations where a participant's coordinates were not measured for long time because she was in a tunnel or underground railway. In such situations, the difference between her actual coordinates and the estimated coordinates became large (several hundred meters). Because we assume that errors in *incorrect* WiFi measurements exceed 150 meters, it was very difficult for the three methods to distinguish true *incorrect* measurements from false *incorrect* measurements.

The Second opinion method achieved good accuracy in Fig. 9. However, this method usually could not predict a participant's coordinates accurately when she walked through a large area, e.g., a university, a park, or a station in the journey and city data sets. In the Second opinion method, we employ the Skyhook location service. Because fingerprints in its WiFi positioning database may be collected using the war-driving technique, estimated coordinates that are far from roadways are not very accurate. Therefore, the recognition results in such large areas were poor. On the other hand, this method achieved good accuracies when the participants were on trains.

Contrary to our expectation, the results we obtained with the Query modification methods related to the journey data set were poor. This is because WiFi positioning errors usually occurred when the number of APs in ws_T was small as shown in Fig. 4. This situation was frequently found in the journey data set. The Query modification methods cannot predict coordinates accurately without sufficient numbers of APs. However, we confirmed that this method could predict coordinates very accurately when sufficient numbers of APs were included in ws_T. This may reflect the results related to the underground data set. The Query modification methods achieved relatively good accuracies related to the data set. The underground parts of cities where our experiment was conducted were crowded and mobile APs carried by people may affect WiFi positioning. On the other hand, underground shops, cafes, and restaurants have many WiFi APs that can be utilized by the Query modification methods.

Also, it was very difficult to detect positioning errors with only context data as shown in the results of the Context method. Note that we confirmed that the Context method achieved about 70% accuracy when the error threshold was 500 meters. So, we can roughly detect positioning errors with only context data.

Results of our method

Figs. 9 and 10 show the results we obtained with our method. Our method greatly outperformed the other methods (86.0% w/o LWL). The average F-measure of our method was about 15% better than that of the Particle filter method even though the Particle filter method employs state-of-the-art tracking techniques. As mentioned above, each naive method has its strengths and weaknesses, and they depend on the phone's context. In our approach, the prepared predictors may cover for each other's weaknesses according to the phone's context. For example, as mentioned above, the Second opinion method did not work well when a participant was walking in a large area. On the other hand, the Kalman filter and Particle filter methods can successfully track the participant in such cases. Because our method can know whether the participant is walking or not by employing her context features (e.g., migration speed), our method can adaptively employ the opinions of the predictors. We consider that this is reflected by the results in Fig. 10 where our method achieves good results for every data set. Also, our method achieved good accuracies related to the underground data set where GPS is not available. This result indicates that we can detect WiFi positioning errors at places where we cannot use GPS polling.

Here we investigate the effect of incorporating context data in the WiFi error detection. That is, we computed the classification accuracy when we did not use GMMs, which deal with context data, in our method. The accuracy was 79.3% and it was about 7% poorer than our standard method. In particular, the accuracies related to the city and underground data sets decreased. We employed information gain based feature analysis [21] and found that the migration speed fea-

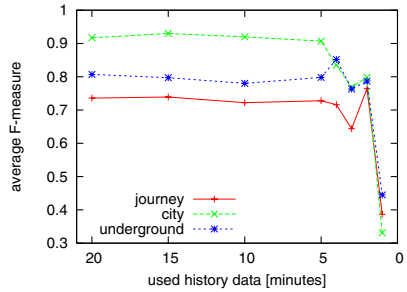

Figure 12. Relationship between average F-measure and quantity of history data to be used.

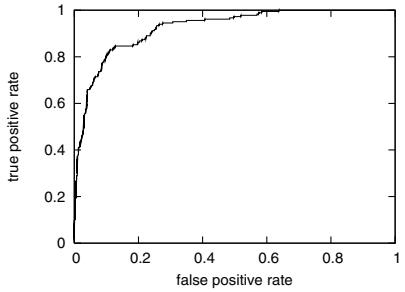

Figure 13. ROC curve for proposed method (locally weighted learning results).

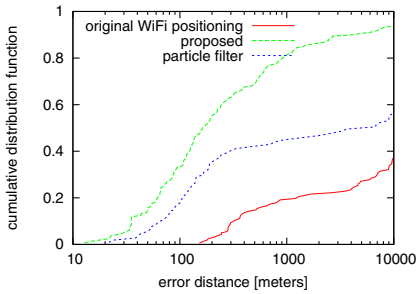

Figure 14. CDF of our error correction method.

ture in context data contributed to the classification task very much. So, we consider that, because participants in the data sets moved with an almost constant velocity, the feature could capture outlying situations (positioning errors) easily.

Also, we investigate how we can correctly detect *incorrect* measurements according to their error distances. Fig. 11 shows classification accuracies related to *incorrect* measurements aggregated for their error distances. Although it may be very difficult to find small errors (150-500 meters) with only WiFi-related data, our method could detect (classify) the errors with about 75% accuracy.

Classification with few histories

The above results were computed by using all available histories (all past coordinates and WiFi scans in a session). Here we investigate the degree to which histories are required for detecting errors. Fig. 12 shows the transition of average F-measure when we change the quantities of history data to be used (20 coordinate points to 1). When we used only one of history data, the accuracy was very poor. This may be because we cannot use several predictors (Kalman and particle filters) and context features with only one history measurement. On the other hand, when we used more than one coordinate point, we could achieve good accuracies because we could employ powerful predictors. Note that the accuracies were unstable when there were few history data because the estimations of the predictors were also unstable.

Sensitivity of error detection

The F-measure related to the *incorrect* class (77.0%) was poorer than that related to the *correct* class (92.4%). This may be because there were fewer *incorrect* instances than *correct* instances in the training data. On the other hand, a binary classifier computes the score of an instance that exhibits the degree to which the instance is a member of a class. We can threshold this score to make a binary decision. By varying the threshold, we obtain an ROC curve as shown in Fig. 13. The true positive rate corresponds to the ratio of the number of *incorrect* instances correctly classified in the total number of *incorrect* instances. Also, the false positive rate is the ratio of the number of *correct* instances misclassified in the total number of *correct* instances. By changing the threshold, we can control the sensitivity of *incorrect* detection. As shown in the graph, when we want to find *incorrect* instances with a true positive rate of about 95%, the false positive rate becomes about 30%.

Evaluation Methodology: Correcting WiFi Coordinates

As with the above evaluation, we employ a leave-one-session-out cross validation to evaluate our error correction method. We compare estimated coordinates obtained with our method with the original coordinates obtained with WiFi positioning. Note that we apply our method to WiFi coordinates whose errors are larger than 150 meters.

Results: Correcting WiFi Positioning Errors

Fig. 14 shows the CDFs of the original WiFi positioning and our method. Because we focus only on WiFi coordinates whose errors are larger than 150 meters, the cumulative probability of the original WiFi positioning was zero when the error was smaller than 150 meters. Our method greatly outperformed the original WiFi positioning, and achieved 47% precision within 150 meters. Also, our method had a precision of 50% within 163 meters. On the other hand, the original WiFi positioning had a precision of 50% within 26786 meters. Fig. 14 also shows the CDF of the particle filter estimator. The estimator had a precision of 50% within 6733 meters. The particle filter estimator outputs very large errors when WiFi positioning errors are continuously observed or positioning errors are observed at the beginning of a session (trajectory).

In the results we obtained with our method, the Kalman filter (improved) and Particle filter predictors contributed greatly (esp. on the city data set). On the other hand, we found that the Kalman filter predictor usually did not contributed because the predictor is strongly affected by incorrect measurements as mentioned in the *Predictors* section. Also, we found that the Second opinion predictor, which employs the Skyhook location provider, contributed. We consider that the areas covered by Skyhook and Google are somewhat different and Skyhook compensated for Google. However, we found that, in many cases, this predictor did not output any results because of the coverage problem. Furthermore, we found that the error estimator for the Query modification predictor sometimes performed poorly. When the number of APs in ws_T that are included in Google's WiFi positioning database is small, the predictor (and Google location service used in the predictor) seems to output highly incorrect coordinates. However, it was very difficult to detect the situation by using our prepared context features.

RELATED WORK

Here we introduce related work that we had no room to include in the above sections. Many studies have attempted to reduce a mobile phone's battery consumption by combining GPS and WiFi positioning. Lin et al. [17] implemented a system that automatically selects an adequate positioning method based on the estimated accuracy requirement for mobile search-based applications. When a user does not require highly accurate positioning, the system employs WiFi positioning to measure the user's coordinates. Zhuang et al. [23] implemented a positioning framework consisting of four design principles; accelerometer-based suppression, piggybacking of positioning queries, substitution of positioning methods, and adaptation (adjustment) of sensing parameters when a battery is low. According to the substitution principle, when GPS is being used by another application, for example, WiFi positioning is used in place of GPS. On the other hand, this paper focused on the reliability of WiFi positioning. When coordinates measured with WiFi have the potential for a large error, a phone can automatically switch to GPS positioning.

Kim et al. [12] developed a system that uses a low-power accelerometer to infer user's movement and automatically determines WiFi sensing frequency. Hopfner et al. [7] discuss the idea of postponing position measures until a phone's semantic location could have changed. Coelho et al. [3] discussed a delay-tolerant positioning technique that stores the WiFi scan data needed to calculate a phone's position, sends the data to a server when appropriate, and calculates the positions on the server. This technique can save energy related to communication and computation costs. Here we assume that our error detection method also works on a server. So, an end user can obtain a WiFi location and its reliability by simply sending WiFi signal strength data to the server. When we obtain a WiFi location from a WiFi location service (such as Google location service), we also send WiFi strength data to the service. That is, our approach does not require additional computation costs and data transmission costs.

S. Chen et al. [1] employed environmental properties including temperature, humidity, and ambient noise obtained by sensor networks to improve indoor positioning accuracies. Y.C. Chen et al. [2] also investigated the effect of environmental factors (people, doors, and humidity) on indoor positioning with WiFi. On the other hand, we employ a phone's context information (e.g., # APs and speed) to estimate the reliability of WiFi positioning outdoors.

CONCLUSION

In this paper we proposed methods for detecting and correcting WiFi positioning errors. Because the WiFi positioning errors occur in various situations, we cope with them by employing an ensemble of location predictors that have various characteristics. In our experimental evaluation with real WiFi coordinate data, our method achieved highly accurate error detection and greatly outperformed naive methods.

REFERENCES

1. Chen, S., Chen, Y., and Trappe, W. Exploiting environmental properties for wireless localization and location aware applications. In *PerCom 2008* (2008), 90–99.
2. Chen, Y., Chiang, J., Chu, H., Huang, P., and Tsui, A. Sensor-assisted Wi-Fi indoor location system for adapting to environmental dynamics. In *ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems* (2005), 118–125.
3. Coelho, A., Meneses, F., and José, R. Delay-tolerant positioning for location-based logging with mobile devices. In *Int'l Conf. on Ubiquitous Computing and Ambient Intelligence* (2012), 129–136.
4. Dempster, A., Laird, N., Rubin, D., et al. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological) 39*, 1 (1977), 1–38.
5. Doucet, A., De Freitas, N., and Gordon, N. *Sequential Monte Carlo methods in practice*. Springer Verlag, 2001.
6. Hodge, V., and Austin, J. A survey of outlier detection methodologies. *Artificial Intelligence Review 22*, 2 (2004), 85–126.
7. Höpfner, H., and Schirmer, M. Energy efficient continuous location determination for pedestrian information systems. In *Int'l Workshop on Data Engineering for Wireless and Mobile Access (MobiDE)* (2012), 58–65.
8. Jain, A., Chang, E., and Wang, Y. Adaptive stream resource management using Kalman filters. In *SIGMOD Conference 2004* (2004), 11–22.
9. Johnson, S. Hierarchical clustering schemes. *Psychometrika 32*, 3 (1967), 241–254.
10. Kalman, R., et al. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering 82*, 1 (1960), 35–45.
11. Kang, J., Cohen, I., and Medioni, G. Continuous tracking within and across camera streams. In *CVPR 2003*, vol. 1 (2003), 267–272.
12. Kim, K., Min, A., Gupta, D., Mohapatra, P., and Singh, J. Improving energy efficiency of Wi-Fi sensing on smartphones. In *INFOCOM 2011* (2011), 2930–2938.
13. Kjærgaard, M., Langdal, J., Godsk, T., and Toftkjær, T. Entracked: energy-efficient robust position tracking for mobile devices. In *MobiSys 2009* (2009), 221–234.
14. Kontkanen, P., Myllymaki, P., Roos, T., Tirri, H., Valtonen, K., and Wettig, H. Topics in probabilistic location estimation in wireless networks. In *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2004)*, vol. 2 (2004), 1052–1056.
15. LaMarca, A., Chawathe, Y., Consolvo, S., Hightower, J., Smith, I., Scott, J., Sohn, T., Howard, J., Hughes, J., Potter, F., et al. Place lab: Device positioning using radio beacons in the wild. In *Pervasive 2005* (2005), 116–133.
16. Larson, R., and Csikszentmihalyi, M. The experience sampling method. *New Directions for Methodology of Social & Behavioral Science 15* (1983).
17. Lin, K., Kansal, A., Lymberopoulos, D., and Zhao, F. Energy-accuracy trade-off for continuous mobile device location. In *MobiSys 2010* (2010), 285–298.
18. Liu, H., Shah, S., and Jiang, W. On-line outlier detection and data cleaning. *Computers & Chemical Engineering 28*, 9 (2004), 1635–1647.
19. Maybeck, P. The Kalman filter: An introduction to concepts. *Autonomous Robot Vehicles* (1990), 194–204.
20. Shevade, S., Keerthi, S., Bhattacharyya, C., and Murthy, K. Improvements to the SMO algorithm for SVM regression. *IEEE Trans. on Neural Networks 11*, 5 (2002), 1188–1193.
21. Witten, I., and Frank, E. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2004.
22. Zandbergen, P. Accuracy of iPhone locations: A comparison of assisted GPS, WiFi and cellular positioning. *Transactions in GIS 13* (2009), 5–25.
23. Zhuang, Z., Kim, K., and Singh, J. Improving energy efficiency of location sensing on smartphones. In *MobiSys 2010* (2010), 315–330.