Indoor Tracking Using Undirected Graphical Models

Zhuoling Xiao, Hongkai Wen, Andrew Markham, and Niki Trigoni

Abstract—Indoor tracking and navigation is a fundamental need for pervasive and context-aware smartphone applications. Although indoor maps are becoming increasingly available, there is no practical and reliable indoor map matching solution available at present. We present MapCraft, a novel, robust and responsive technique that is extremely computationally efficient (running in under 10 ms on an Android smartphone), does not require training in different sites, and tracks well even when presented with very noisy sensor data. Key to our approach is expressing the tracking problem as a conditional random field (CRF), a technique which has had great success in areas such as natural language processing. Unlike directed graphical models like Hidden Markov Models, CRFs capture arbitrary constraints that express how well observations support state transitions, given map constraints. In addition, we show how to further improve tracking accuracy, by tuning the parameters of the motion sensing model using an unsupervised EM-style optimization scheme. Extensive experiments in multiple sites show how MapCraft outperforms state-of-the art approaches, demonstrating excellent tracking error and accurate reconstruction of tortuous trajectories with zero training effort. As proof of its robustness, we also demonstrate how it is able to accurately track the position of a user from accelerometer and magnetometer measurements only (i.e., gyro- and Wi-Fi-free). We believe that such an energy-efficient approach will enable always-on background localisation, enabling a new era of location-aware applications to be developed.

Index Terms—Inertial, orientation tracking, map matching, conditional random fields

1 Introduction

Whereas GPS is the *de facto* solution for outdoor positioning, no clear solution has as yet emerged for indoor positioning despite intensive research and the commercial significance. Applications of indoor positioning include smart retail, navigation through large public spaces like transport hubs, and assisted living. The ultimate objective of an indoor positioning system is to provide continuous, reliable and accurate positioning on smartphone class devices. We identify maps as the key to providing accurate indoor location; this is a reasonable assumption given that indoor mapping is a high priority for companies, such as Google, Microsoft, Apple, Qualcomm, and so on. A map can be viewed in the broadest sense as a spatial graph which provides constraints. At the simplest level this takes the form of a floor plan of a building. This constrains the allowable motion of a user—people cannot walk through walls and can only enter a room through a door. Other maps (meta-maps essentially) provide additional constraints or features, such as the positions of access points (APs), radio fingerprints, signal strength peaks or distorted geomagnetic fields. Based on a time-series of observations, such as inertial trajectories or RF scans, the goal is to reconcile the observations with the constraints provided by the maps in order to estimate the most feasible trajectory of the user, i.e., the sequence that violates the fewest constraints.

Existing map matching techniques, based on recursive Bayesian filters, such as Hidden Markov Models (HMMs), Kalman and particle filters, have been successfully applied to the location estimation problem, but are limited in two ways: First, they are computationally expensive, and are thus typically delegated to the cloud to run. Not only does this lead to lag and service unavailability in connection-poor areas, it has the side-effect of leaking detailed sensor data and precise location to a third party. Second, they typically require high fidelity sensor data to estimate accurate trajectories, leading to power drain.

Motivated by these pressing problems, we present a fresh approach to indoor positioning that is lightweight and computationally efficient, but also robust to noisy data, allowing it to provide always-on and real-time location information to mobile device users. The goal of the proposed approach is to achieve similar or better accuracy as existing techniques, but with fewer computational, space and sensor resources. Unlike existing techniques that model the problem using directed graphical models, the proposed MapCraft algorithm uses an undirected graphical model, known as linear chain conditional random fields (CRFs). The CRF model is particularly flexible and expressive, allowing a single observation to be related with multiple states and for multiple observations to inform a single state. This allows us to capture correlations among observations over time, and to express the extent to which observations support not only states, but also state transitions.

In terms of performance, MapCraft is two to three orders of magnitude more computationally efficient than competing techniques, running in <10 msec on an Android phone, enabling real-time location computation online. The second advantage of MapCraft is that it offers high location

- *The authors are with the Department of Computer Science, University of Oxford, Oxford, United Kingdom. E-mail: zhuolingxiao@gmail.com, {hongkai.wen, andrew.markham, Niki.Trigoni}@cs.ox.ac.uk.*

Manuscript received 10 Feb. 2014; revised 4 Nov. 2014; accepted 12 Jan. 2015. Date of publication 1 Feb. 2015; date of current version 29 Sept. 2015.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TMC.2015.2398431

accuracy even when it uses ultra low power sensors (e.g., accelerometer and magnetometer), whereas existing approaches rely heavily on power hungry sensors like frequent Wi-Fi scans to monitor the local radio environment, or gyroscopes running at high sampling rates to capture turns and steps. Until recently, because of the dominant cost of the main processor, deactivating these sensors had little impact on overall power consumption. However, with the advent of motion tracking devices, we see a clear trend of low power digital motion processors (DMPs), e.g., InvenSense MPU-6000/MPU-6050, able to task and process inertial data in bursts, while the system processor remains in a low-power sleep mode. In this new regime, the dominant cost will not come from the processor, but from the power hungry sensors, such as Wi-Fi and gyroscope. Another reason for gyro-free motion sensing is the anticipated growth of the wearable device market, in which many ultra low power chips (e.g., KMX61 and LSM303C) are not equipped with gyroscopes. Thus, algorithms that can afford not to use these sensors are key to offering an always-on positioning service for a large range of low power devices. In summary, this paper's key contributions are

- Lightweight map-matching: The proposed MapCraft technique enables computationally efficient, real-time map-matching using sensor data from multiple sources and a floor plan.
- Robust and accurate indoor tracking with noisy trajectories: We demonstrate excellent performance even in the presence of bias, noise and distortions. As an extreme case, we show accurate tracking can be obtained from gyro-free dead reckoning.
- Unsupervised parameter learning: Crucial parameters of the motion sensing model are learned in an unsupervised manner. Zero user effort is required to make the tracking system work in a new environment as long as the map is available.
- Extensive real-world validation: We achieve high tracking accuracy in multiple environments (office, museum, market). MapCraft outperforms existing map matching techniques even without training.

The remainder of this paper is organized as follows: Section 2 outlines the system architecture and Section 3 overviews existing techniques. Section 4 introduces the CRF model and Section 5 proposes MapCraft, a map matching solution that uses CRFs for indoor tracking. Section 6 estimates crucial parameters in MapCraft with an unsupervised learning approach. Section 7 extensively evaluates MapCraft in three indoor settings, and compares it with competing techniques. Section 8 concludes the paper and discusses ideas for future work.

2 System Model

The system architecture is shown graphically in Fig. 1, and is described through the use of an example. When a user enters a building and launches the tracking application, the application requests a floor plan (along with other meta-data as generated by other systems, which could include fingerprint maps) from the server, if not already within the cache. Note that this is the only time that a user needs to reveal any data about their coarse position to a third party. Sensors on the user's phone collect data about the motion and (radio) environment. Motion sensors can include accelerometers, magnetometers and gyroscopes. Radio sensors can include Wi-Fi, Bluetooth (low energy), FM radio and so forth. Other sensors can also be used such as acoustic or vision. Raw sensor data is typically not immediately usable and needs to be processed. In the case of motion data, this could include dead reckoning trajectories based on counting steps and estimating heading, or using full IMU tracking in the case of foot mounted sensors. For RF data, a channel/propagation model can be used to relate RSS to physical distances. Alternatively, raw signal strengths may be directly forwarded to the CRF model, to be later combined with RF fingerprint map data if available.

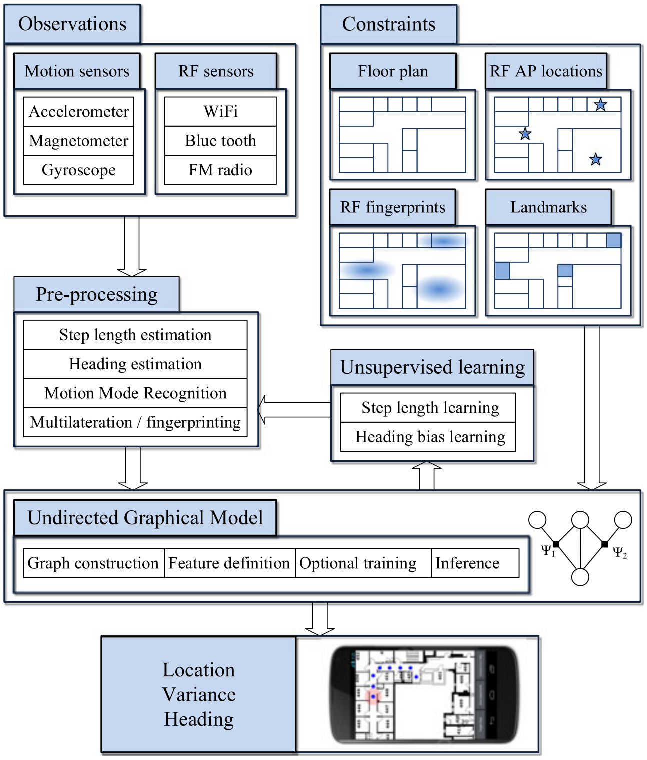


Fig. 1. System architecture.

Maps and observations are combined using conditional random fields, an undirected graphical model described in Section 4. The CRF model is particularly well suited to this sequential problem because it allows us to flexibly define *feature functions* that capture the extent to which observations support states and state transitions, given map constraints. As a user moves through the building, certain paths become unlikely, as they violate map constraints. The Viterbi algorithm is used to efficiently find the most likely sequence of states through the transition graph, culminating in an estimate of the user's location and quality thereof. Meanwhile, model parameters like step length and heading bias can be learned from the undirected graphical models using unsupervised EM-style optimization. The learned parameters are then fed into the preprocessing to correct the sensor errors/biases to form a robust long-term tracking system.

3 Background

Before presenting our novel approach, we will first position our work in the context of the literature. We focus on techniques that make use of widely available infrastructure, such as inertial sensors embedded in mobile devices and wireless access points in buildings, which we divide into three classes: 1) motion sensing techniques that use magnetometer, accelerometer and gyroscope data; 2) RSS-based techniques that make use of received signal strength readings from wireless Access Points, and 3) Bayesian fusion of inertial and RSS sensor data.

3.1 Motion Sensing

Motion sensing involves fusing data generated by inertial measuring units (IMUs) to compute the user trajectory relative to her initial position. Some techniques assume that IMUs are mounted on the foot of the person [11], [15], [37], whereas others obtain data from IMUs embedded in consumer electronic devices, such as smartphones [31].

Inertial motion sensing is performed by iteratively repeating the following three tasks:

Motion Mode Recognition which uses accelerometer and gyroscope data to distinguish between different modes of movement (e.g., static, walking and hand texting, walking with phone in a bag, etc.) [31].

Orientation Tracking which uses magnetometer, accelerometer and, optionally, gyroscope data (we utilize unscented kalman filter (UKF) to fuse these data in this study) to estimate the device orientation [14], [30].

Step Length Estimation which uses accelerometer to detect the step frequency (denoted with f_t at time t) which is later used to estimate step length l_t as

$$l_t = h(\alpha f_t + \beta) + \gamma, \tag{1}$$

where h is the pedestrian height, α is the step frequency coefficient, β is the step constant given pedestrian height, and γ is the step constant. A similar model can be found in [25].

Orientation tracking and step length estimation are performed only when the estimated motion mode is not static, i.e., the user has moved from her initial position. The last two tasks iteratively extend the trajectory by detecting a step, and extending the trajectory by the estimated length of that step, along the estimated orientation.

Challenges in motion sensing. Practical challenges in motion sensing, such as the sensitivity to phone position and the variability in user walking profiles, are explored in [20]. In cooperative scenarios, accuracy can further be improved by fusing inertial data with user encounter information [9], [32]. A major challenge with motion sensing is dealing with orientation errors due to magnetic field distortions and sensor biases. This problem will be exacerbated when trying to infer orientation without the gyroscope, which will be increasingly power-efficient with the advent of digital motion processors (Fig. 2). Even with the aid of gyroscope, the initial heading can only be obtained from the magnetometer. Errors due to building effects cause initial heading bias which is compensated by initial heading bias learning techniques discussed in Section 6.2.

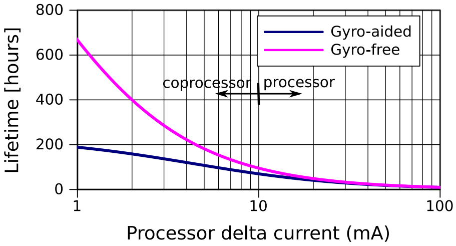


Fig. 2. The power consumption gap between gyro-free and gyro-aided systems will increase with the emergence of digital motion processors (assuming 1000 mAh battery, gyro-free current of 0.5 mA and gyro-aided 4.3 mA).

Another challenge is the cumulated error from the inaccurate step length estimation. The step length estimation model in (1) is very inaccurate for handheld devices because the acceleration signal captured with handheld devices is a mixture from the movement of the feet, the body, and the hand, which makes it difficult to build a simple model to estimate the step length. In addition, it is nearly impossible for one model to capture unique signal features of different individuals. Therefore, it is necessary that we have unsupervised online learning techniques to fine tune the step length parameters so as to improve the step length estimation accuracy, as discussed in Section 6.1 in detail.

3.2 RSS-Based Localisation

RSS-based fingerprinting is another popular method due to the wide availability of wireless APs, and the cost benefits of not having to install and maintain special-purpose infrastructure. Existing techniques, such as Radar [3], PlaceLab [19] and Horus [41], typically involve a training phase, in which a building is surveyed and the signal strengths received at each location from the various APs are recorded in a radio map. Once a map is available, people can use it to determine their own location by comparing the signal strengths that they receive from APs with those in the map. Further techniques have been developed to deal with heterogeneous wireless clients [16] or to exploit additional features of the environment, e.g., FM signals [6], sound, light and color [2], [38]. The main disadvantage of these methods is that they require labour intensive surveying of the environment to generate radio maps. To address this problem, there have been a number of simultaneous localisation and mapping efforts recently that aim to automatically build the radio map, by fusing RSS with motion sensor data, as discussed in the next section.

3.3 Bayesian Fusion of Motion and RSS Data

Existing fusion algorithms, such as Hidden Markov Models, Kalman and Particle Filters, are typically based on Bayesian estimation. They represent the conditional dependence structure between observation and state variables using directed graphical models (top of Fig. 3). It is often assumed that the sensor measurements are conditionally independent of each other given the state. This is the basis of the Naive Bayes model (top left of Fig. 3). The extension of this model to a sequence of states linked through transition probabilities leads to recursive Bayesian models, such as HMMs (top center of Fig. 3). The joint probability

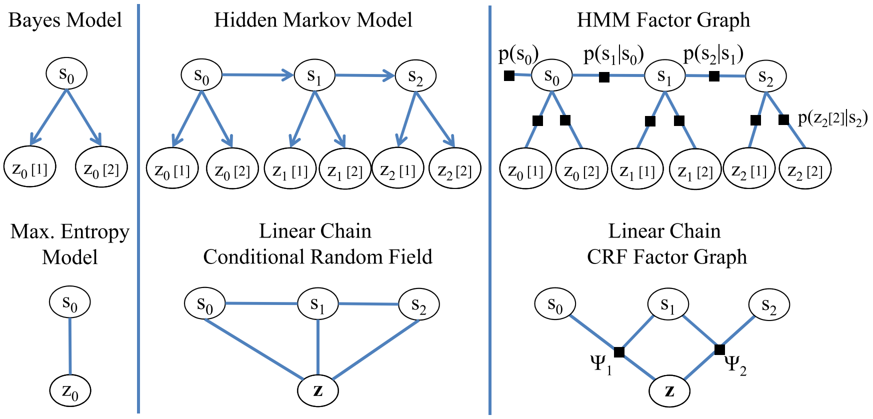


Fig. 3. Directed generative graphical models (top) versus undirected discriminative models (bottom).

distribution between all state and observation variables is decomposed into a product of conditional distributions (top right of Fig. 3)

$$p(S_{0:T}, Z_{0:T}) = p(S_0)p(Z_0|S_0)\prod_{i=1}^{T}[p(S_i|S_{i-1})p(Z_i|S_i)], \qquad (2)$$

where $S_0, \ldots, S_T$ are the variables representing the real states of a system (e.g., the locations of a person) over a time horizon $0, \ldots, T$, and $Z_0, \ldots, Z_T$ are the observation variables over the same time period.

The problem of indoor localisation is either cast as a filtering problem, or as a smoothing problem if the user can tolerate some delay, or as the inference problem of finding the most likely trajectory $S_{0:T}$ given observations $Z_{0:T}$.

Hidden Markov Models are a special class of a recursive Bayesian model where state variables are discrete, and the transition model $p(S_i|S_{i-1})$ is a matrix. HMMs have been widely used for map matching and location estimation, both outdoors using road maps [12], [21], [33] as well as indoors [28], [36], and they come in two flavours: one is the first order HMM where states represent user locations. An alternative is to model the transitions between pairs of locations as the state itself, an equivalent second order HMM. A limitation of both models is that they do not take into account correlations between nearby inertial observations, for example correlated magnetometer bias due to the metal disturbances in the earth's magnetic field. Section 7 shows how this limitation impacts the accuracy of first-order [28], [33] and second-order HMM algorithms [12].

Kalman filters. An alternative approach is to consider the location of a user as a continuous variable and resort to a Kalman Filter variant, e.g., Extended or unscented kalman filter (e.g., [5]). Kalman filters have also been widely used in outdoor environments to fuse inertial trajectories and GPS readings with map information [22]. Sensor fusion is typically performed in two steps. First, the Kalman Filter estimates the position of a moving node after taking into account its previous position and the inertial sensor data. Then, RSS data are used to update the node's position, abiding by map constraints. Similar to first order HMMs, Kalman Filters are effective when radio signal strengths are periodically sampled from various access points, and fused with inertial data; however, their performance deteriorates when we solely make use of inertial sensors and the building's floor plan.

Particle filters. Another common technique for performing online map matching is to use a particle filter [1], [4], [20], [24], [37]. The key idea of particle filters is to approximate the distribution $p(S_{t-1}|Z_1, \ldots, Z_{t-1})$ by a set of particles. In each round, these are first moved according to the transition model, their weights are then updated according to the observation model, and particles are then re-sampled according to their weights. The likelihood of each trajectory being correct is calculated and trajectories which are unlikely or impossible (for example, crossing a wall) are culled. Locations which are more likely act as seeds for the next iteration of the algorithm, through a resampling step. Over time, the particles typically converge to the most likely position of the user. One of the major issues of the particle filter is the computation time, as a large number of particles are typically required to ensure good estimation of the continuous probability distribution, especially when dealing with noisy inertial data and large maps. Section 7 shows that our particle filter implementation (inspired by [24]) often fails to converge when using gyro-free dead reckoned data due to errors in orientation estimates. When it uses the full suite of IMU data and Wi-Fi, it converges but at a much higher cost.

Wi-Fi SLAM. Other approaches such as Wi-Fi-based SLAM fuse RSS and motion sensor data to simultaneously build a map of the environment and locate the user within this map [10], [26], [27]. Recently, [40] proposed a SLAM approach that does not exploit the full power of dead reckoning but only measures walking steps. SLAM approaches can be seen as orthogonal to our work: first, we assume that basic maps, i.e., floor plans, are available and there is no need to discover them; second, we view SLAM techniques as map generators providing optional input to our map matching algorithm, e.g., radio fingerprint maps [10], or organic landmark maps [29], [35]. Once floor plans (and optionally radio maps) are available, our focus is on designing sensor fusion techniques that make the best possible use of sensor data and maps in a way that is both lightweight and robust.

In summary, existing end-to-end smartphone-based indoor positioning solutions fall into two categories: RF category where only Wi-Fi/BLE are used and fusion category where inertial data and Wi-Fi/BLE data are fused to perform positioning. Typical examples in the RF category are Horus [41], Radar [3], EZ [7], and SCPL [39]. The state-of-the-art algorithms in the fusion category are Zee [24], UnLoc [35], Wi-fiSLAM [13], and the algorithm in [20]. These approaches require either labor intensive site survey or intensive computation to yield good accuracy. In this paper, we propose a novel approach to resource-efficient localisation based on conditional random fields.

4 Conditional Random Fields

CRFs are undirected probabilistic graphical models introduced by Lafferty et al. [18]. They have been successfully applied to a number of tasks in computer vision (e.g., classifying regions of an image), bioinformatics (e.g., segmenting genes in a strand of DNA), and natural language processing (e.g., extracting syntax from natural-language text). In the context of indoor localization, they have only been used for subtasks of localization like motion recognition [23]. In all of these applications, the input is a vector of observations $Z = \{Z_0, \ldots, Z_T\}$, and the task is to predict a vector of latent variables $S = \{S_0, \ldots, S_T\}$ given input Z.

Maximum entropy model (MEM). In order to introduce CRFs, we must first introduce the *maximum entropy model*. A chain CRF is an extension of MEM for state sequences, in the same way that a HMM extends a naive Bayes model, as illustrated in Fig. 3. The maximum entropy model assumes that given incomplete knowledge of the probability distribution $p(S_0|Z_0)$, the only unbiased estimate is a distribution that is as uniform as possible given training data (consisting of several (Z_0, S_0) values). This implies finding the model that has the largest possible conditional entropy

$$p^*(S_0|Z_0) = \underset{p(S_0|Z_0)\in P}{\operatorname{argmax}} H(S_0|Z_0), \tag{3}$$

where P is the set of all models consistent with the training material. To explain the meaning of consistency, let's consider a set of m features $f_1, \ldots, f_m$, each one of which is a function of observation and state variables. A model is consistent with the training material when the expected value of each feature in the empirical distribution (training dataset) is equal to its expected value in the model's distribution. Each feature thus introduces a constraint, and finding $p^*(S_0|Z_0)$ becomes a constrained optimisation problem. For each constraint a Lagrange multiplier λ_i is introduced; the optimal solution in the maximum entropy sense is log linear [17], [18]

$$p^*_\lambda(S_0|Z_0) \propto \exp\left(\sum_{i=1}^{m} \lambda_i * f_i(S_0, Z_0)\right). \tag{4}$$

The conditional probability distribution of states given observations is thus proportional to the exponentiated sum of weighted features.

Linear Chain Conditional Random Fields can be viewed as the sequence version of maximum entropy models, in the same way that HMMs is an extension of the naive Bayes classifier model. In linear chain CRFs, the conditional probability of states given observations is proportional to the product of potential functions that link observations to consecutive states, as expressed in the equation below and shown in Fig. 3 (bottom right)

$$p^*_\lambda(S|Z) \propto \prod_{j=1}^{T} \Psi(S_{j-1}, S_j, Z, j), \tag{5}$$

where j denotes the position in the observation sequence, and Ψ are potential functions. A potential function is composed of multiple feature functions f_i, each of which reflects in a different way how well the two states S_{j-1} and S_j are supported by the observations Z

$$\Psi_j(S_{j-1}, S_j, Z, j) = \exp\left(\sum_{i=1}^{m} \lambda_i * f_i(S_{j-1}, S_j, Z, j)\right). \tag{6}$$

Differences between HMMs and CRFs. HMMs are directed generative graphical models: they are trained to maximize the joint probability distribution of observation and state variables, which they compute as a product of state priors and conditional probabilities of observations given states (top right of Fig. 3). In contrast to HMMs, CRFs are undirected discriminative models: they are trained to directly maximise the conditional probability of state variables given observation variables, which they compute as a product of potential functions (bottom right of Fig. 3). Thus, unlike HMMs, in CRFs there is no need to model the exact conditional probability distributions of observations given states (or state transitions). Instead one only has to define feature functions, as discussed in Section 5.2.

Furthermore, as shown in the CRF factor graph (bottom right of Fig. 3), the power of CRFs over HMMs lies in how they link observations to each other and to states. CRFs are able to model both: 1) how observations relate to individual states, as well as 2) how they relate to transitions between states. This is very convenient for tracking systems that make use of inertial sensor data, which naturally depend on the transition between two locations rather than on a single location. Using inertial observations in a HMM would complicate things by either having to use an input-output first order HMM, where transition probabilities depend on inertial data, or having to model the problem as a second-order HMM, where a state represents a transition between locations.

In HMMs observations at a given timestamp are typically considered independent of each other given the state (naive Bayes assumption), whereas in CRFs, it is possible to define features that capture these dependencies. In addition, in HMMs, observations generated at a given step only depend on that step's state. In CRFs, we are flexible to define features that link an entire chain of observations (of arbitrary length) with a state or a state transition. This is useful when nearby observations are perturbated with correlated errors, e.g., when a local distortion of the magnetic field affects several consecutive heading observations. It is further useful when using RSS landmarks for tracking; for instance, several RSS observations must be received after time t before deciding if the observation at time t is a peak value.

Finally, in CRFs, it is possible to define more than one feature function that captures the dependency between a sub-chain of observations and a state (or state transition).

New feature functions can be used to accommodate new sensing modalities in a natural way.

5 Map Matching Using CRFs

We are now in a position to describe our algorithm, called MapCraft, which makes use of linear chain CRFs to track people in indoor environments.[1] MapCraft involves four distinct steps: 1) Map pre-processing; 2) Definition of states and feature functions; 3) Training to determine feature weights; and 4) Inference to estimate location over time. The first three steps are performed once for each building by either a mobile phone or a service in the cloud.. The fourth step is performed online on the user's smartphone to track themselves.

5.1 Map Pre-Processing

This step takes a floor plan as input, and produces a graph that 1) encodes a set of discrete states (locations), and 2) represents physical constraints between discrete states imposed by the map. This information will then be fed to the second step, to help us define the CRF's states and feature functions. In our implementation, such information is obtained from maps in various image formats. The main task is to extract edges from the image needed to perform map structure recognition or reconstruction, using standard edge detection algorithms. Note that indoor maps are typically cleaner than big Google maps, which makes the extraction quite simple. Standard edge detection algorithms, including simple ones that use pixel grey scale detection, could accomplish this goal. We can then use a connectivity test to find out the reachable regions in the map. A graph is built on the reachable region of the map. We first divide the map into identical squares with edge length e^2. The size of e impacts both the position accuracy and the computational cost of the map matching algorithm. On the one hand, the larger the edge length the coarser the achievable position accuracy. In addition, to have connected vertices in narrow corridors, the edge length should not exceed the width of corridors connecting different parts of the building. On the other hand, as we decrease the edge length, we increase the computational cost of the map matching algorithm, which is quadratic on the number of vertices and bi-quadratic on the number of edges. On balance, a suitable choice of e in our system is the width of the narrowest corridor in most buildings, e.g., 0.8 m. The accuracy benefits of more fine-grained maps were observed to be negligible compared to the additional computational cost that they incurred.

The neighbours of a target vertex are vertices which have a common geographical border with the target vertex and pass the connectivity test as well. The removal of unreachable vertices is important to the system performance, because there is typically a large number of vertices in the map that cannot be reached from the legal region. The process of map generation and graph construction only happens once when a new map is used in the system.

5.2 Definition of States and Feature Functions

The output of the previous step directly allows us to define the state space of the CRF, as the set of discrete locations encoded in the vertices of the generated graph. We are now in a position to introduce the set of feature functions used by MapCraft. Recall that a feature function f_i defines the degree to which observations Z support our belief about two consecutive states (S_{t-1} and S_t); the stronger the support, the higher the value of the feature function $f_i(S_{t-1}, S_t, Z)$. Note that in CRFs, we are free to use any subset of observations, generated at a single or multiple time steps, though in most cases, the observations that matter are those temporally close to time t. In what follows, we specify for each feature the subset of observations that it uses, and how it relates them to state transitions or, in some cases, to individual states.

The first feature in our system expresses the extent to which an inertial measurement Z_t^{in} supports the transition between states S_{t-1} and S_t

$$\begin{aligned} f_1\big(S_{t-1}, S_t, Z_t^{in}\big) &= I(S_{t-1}, S_t) \\ &\times \big(f_1^\theta\big(S_{t-1}, S_t, Z_t^\theta\big) + f_1^l\big(S_{t-1}, S_t, Z_t^l\big)\big), \end{aligned} \tag{7}$$

where $I(S_{t-1}, S_t)$ is an indicator function equal to 1 when states S_{t-1} and S_t are connected and 0 otherwise. The inertial observation Z_t^{in} has two components: the measured length Z_t^l and angle (heading) Z_t^θ of displacement, which are assumed to be independent. f_1^θ and f_1^l are the functions to relate the angle and length to the underlying graph, respectively. The function f_1^θ is given as

$$f_1^\theta\big(S_{t-1}, S_t, Z_t^\theta\big) = \ln\frac{1}{\sigma_\theta\sqrt{2\pi}} - \frac{\big(Z_t^\theta - \theta(S_{t-1}, S_t)\big)^2}{2\sigma_\theta^2}, \tag{8}$$

where $\theta(S_{t-1}, S_t)$ is the orientation of the edge between states S_{t-1} and S_t, and σ_θ^2 is the heading variance of the observation Z_t^θ. The feature function f_1^l is defined likewise.

The purpose of the second feature is to handle correlations in heading errors in a recent time window. It does so by measuring how well a *corrected* inertial measurement $Z_t^{in}(\hat\theta)$, derived by rotating Z_t^{in} by angle $\hat\theta$, supports the transition between states S_{t-1} and S_t:

$$f_2(S_{t-1}, S_t, Z_{0:t}) = f_1\big(S_{t-1}, S_t, Z_t^{\text{in}}(\hat\theta)\big), \tag{9}$$

The rotation angle $\hat\theta$ is estimated as the average heading difference between the estimated and measured headings from time stamp $t-w$ to t, where w is a window size parameter. The estimated headings are based on MLE state estimates generated by MapCraft's inference step (Section 5.4). More specifically

$$\hat\theta = \sum_{i=1}^{w} \Theta\big(S_{t-i}^{\text{MLE}} - S_{t-i-1}^{\text{MLE}}, Z_{t-i}^\theta\big)/w, \tag{10}$$

where Θ represents the angle between two vectors, and S_t^{MLE} is the maximum likelihood estimate (MLE) of the state at time t taking into account all measurement from 0 to t. MLE state estimates are computed efficiently using the Viterbi algorithm as explained in Section 5.4.

[1] Since we assume that a floor plan (and optionally a radio map) is readily available, we interchangeably refer to the tracking problem as the map matching problem. Note however that CRFs as such do not require the presence of a map; they could be used to infer location without a map, using as input location sensor data from various modalities.

The third feature function is optional, and it takes into account the signal strength observations in conjunction with a radio fingerprint map, if it is available in a building. Unlike the previous two feature functions, that constrain state transitions, this feature constraints individual states

$$f_3(S_t, Z_t^{\mathrm{RSS}}) = -(S_t - \boldsymbol{\mu_t})^T (\Sigma_t)^{-1} (S_t - \boldsymbol{\mu_t}), \qquad (11)$$

in which the observation Z_t^{RSS} is the estimated mean $\boldsymbol{\mu}_t$ and covariance Σ_t of current position given RSS fingerprint data. This feature measures the negative squared Mahalanobis distance between the state and the RSS-based position estimate.

The CRF model used by MapCraft combines the three features above into a potential function $\Psi_j(S_{j-1}, S_j, Z, j)$, which is computed as the exponentiated function of their weighted sum as shown in 6. The way that weights λ_i are determined is explained in the next section (Training step). However, as we will show in Section 7, in typical indoor environments, the training step is not strictly required, as using equal weights typically yields comparable location accuracy to that obtained after careful weight training. Hence, in practice the following training step can be skipped and equal weights can be assigned to the three features above.

The power of the CRF model is that it does not constrain us to only use the features above. Depending on the sensor data available and the maps available, it might be useful to extend the list of features. For example, suppose that we are in possession of a radio map, denoted with PeakPointInMap(S_t), that contains the locations where the RSSI from an access point takes a local peak value (e.g., provided by [29]). We could then define a fourth feature as follows:

$$f_4(S_t, Z_{t-w:t+w}) = \begin{cases} 1, & \text{if } \; Z_t = \max(Z_{t-w:t+w}) \\ & \text{and } \; \mathrm{PeakPointInMap}(S_t), \\ 0, & \qquad \text{otherwise.} \end{cases} \qquad (12)$$

In case the building is equipped with cameras or other sensors, additional features could be added to easily incorporate visual and other sensor data. Since the computational complexity of CRFs is sub-quadratic with respect to the number of features [8], MapCraft can still work well and fast even if we double/triple the number of features. In general, CRFs provide a flexible model where a number of different observations can be fused into the model. Recall that since we use a CRF model, we do not need know the exact observation probabilities given states, and need not restrict ourselves to assuming conditional independence of observations given state. Finally we are free to associate a state at a particular time step (or a state transition) with observations that go beyond this time step.

5.3 Training to Determine Feature Weights

In many scenarios where CRFs are applied, the freedom of being able to define a number of different features comes at the cost of needing to estimate their weights. This step requires training material T, which consists of one or more true trajectories, paired with respective sequences of sensor observations. Training the CRFs to estimate weights is performed by maximising the log-likelihood on the training material T, i.e., the log of the conditional probability of states given observations

$$L_\lambda(T) = \sum_{((S,Z) \in T)} \log p^*_\lambda(S|Z). \qquad (13)$$

By taking the partial derivative of the log likelihood function with respect to each feature λ_j and setting it to 0, we get the maximum entropy model constraint

$$E_T(f_i) - E(f_i) = 0 \qquad (14)$$

that is, the expected value of the *i*th feature under the empirical distribution $E_T(f_i)$ is equal to its expected value under the model distribution $E(f_i)$, where the two expected values are

$$E_T(f_i) = \sum_{(S,Z) \in T} \sum_{j=1}^{T} f_i(S_{j-1}, S_j, Z, j) \qquad (15)$$

$$E(f_i) = \sum_{(S,Z) \in T} \sum_{S' \in StateSeq} \sum_{j=1}^{T} f_i(S'_{j-1}, S'_j, Z, j). \qquad (16)$$

Setting the gradient to zero does not always give us an analytical solution for the weights λ_i. This requires resorting to iterative methods, such as iterative scaling or gradient-based methods. Independent of the method used, one needs to be able to compute $E(f_i)$ and $E_T(f_i)$ efficiently. This is easily done for $E_T(f_i)$, since all we have to do is go over each training sequence, sum up the weighted sum of feature functions over all time steps, and sum up the result for all training sequences.

Computing $E(f_i)$, however, is slightly more complicated and requires the use of a dynamic programming approach, known as the Forward-Backward algorithm, similar to the one typically used for Hidden Markov Models. The details of how this algorithm is applied to CRFs can be found in [17]. The time complexity of the Forward-Backward algorithm is $O(|S|^2 T)$, where T is the length of the sequence and $|S|$ is the number of discrete states.

The goal of the training is to tune the feature weights λ_i in Eq. (6) in order to make the features best support the training data. The weight actually reflects how much we trust the corresponding feature. For instance, if we set a large weight, e.g., 5, for feature $f_1(S_{t-1}, S_t, Z_t^{in})$, we can see from Eq. (8) that it is equivalent to decreasing the length variance σ_l^2 and angle variance σ_θ^2, which means we consider the length and angle measurements to be more accurate than indicated by their variances (in Eq. (8)). Therefore, it is not necessary to tune the weights from training data if the variances of measurements are estimated accurately.

It is also worth noting that large feature weights should be avoided, because they amplify slight differences in the feature values of two tracking solutions into significant differences in the potential function ψ, due to the exponential function in Eq. (6). We suggest that all feature weights should be chosen from $[0.5, 2]$, which works well in all our experiments in different environments. It is also demonstrated in Section 7 with our empirical results from three different indoor environments that the training step only has slight impact ($< 10\%$) on localisation accuracy. The default setting of weights equal to 1 for all features yields similar performance to weights derived from training. Hence, in practice we do not need training material (ground truth trajectories); we can directly proceed to the location inference step described below.

5.4 Inference to Estimate Location over Time

The final step is finding the most likely sequence of hidden states, i.e., the most likely trajectory S^*. This requires solving the following optimisation problem:

$$S^* = \underset{S}{\operatorname{argmax}}\, p(S|Z). \tag{17}$$

The Viterbi algorithm, a dynamic programming algorithm, offers an iterative solution. The Viterbi algorithm is often used for error correction in convolutional coding and is computationally efficient with a worst case time complexity of $O(|S|^2 T)$, where T is the length of the trajectory in steps and $|S|$ the number of states. It is similar to the Forward-Backward algorithm used in each learning iteration, with the subtle difference that it applies a maximisation instead of a summing operation in each induction step. More specifically, in each step, it evaluates the highest score $\delta_j(s)$ along a path at position j that ends in each possible value s for state S_j, as follows, and gradually fills a lattice with these values

$$\delta_j(s|Z) = \max_{s' \in S} \delta_{j-1}(s')\Psi_j(s', s, Z, j). \tag{18}$$

In the case of on line *real-time tracking*, the most recently filled column of the lattice, which represents a discrete distribution $p(S_i|Z_{1:i})$ is normalised and converted into a 2D Gaussian distribution and displayed on the user's map. If MapCraft is extended to employ features that use a few observations after the current step (e.g., feature f_4), a slight delay will be introduced in displaying a user's location. In the case of delay tolerant off line tracking, it is possible to wait until the location accuracy is high before performing the path backtracking step of the Viterbi algorithm, and computing the optimal path form the lattice. As an alternative, we can combine the online and offline approaches above and refresh the users map to show at each step the most up-to-date previous and current location estimates. Then the accuracy of the position estimate is determined by the 2σ or 3σ rule.

6 Parameter Learning

The accuracy of MapCraft, the proposed CRF-based algorithm for indoor tracking, does not only depend on the design of feature functions and the weights used to combine them. It also very much depends on the quality of observations that it takes as input. In this section, we focus on motion observations. We propose an unsupervised approach to tuning the parameters of the motion sensing algorithm (discussed in Section 3.1), including the step length parameters (α, β, and γ in (1)), the heading estimation parameters especially the initial heading bias, and the feature weights which we have discussed in Section 5.3. To make the tracking system practical, we develop unsupervised learning techniques discussed below to estimate the aforementioned parameters.

6.1 Step Length Parameter Learning

To make the step length parameter learning simple and efficient, we only learn the step constant γ in (1) for different individuals because 1) the average step length plays a crucial role in the tracking accuracy; and 2) the parameters α and β are very similar for different individuals in our experiments.

Conceptually, the parameter learning could be done with a series of observations (including both the step frequency f_t and heading) given the map constraints because there is only one state sequence that can best support the observations if the observation sequence is sufficiently unique in the given map.

Therefore, the key idea of the step length parameter learning is that only when the estimated step length is the same as (or very close to) the real step length can we get the maximum conditional probability of the state sequence given the observation. Therefore, the step constant learning actually becomes the following optimization problem:

$$\gamma^* = \underset{\gamma}{\operatorname{argmax}} \ln\, p(S|Z), \tag{19}$$

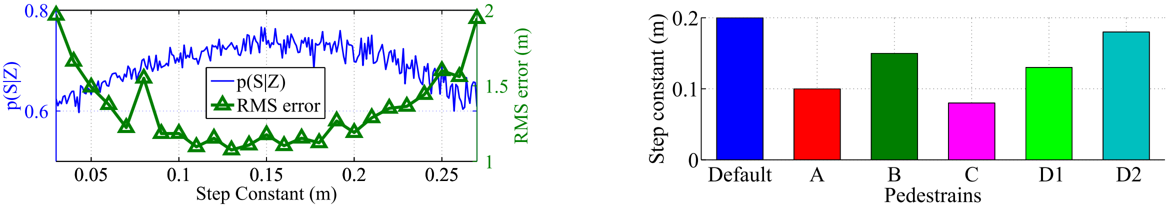
where γ is the step constant in (1).

However, the feature functions shown in Section 5.2 cannot be used in the parameter learning process. We take the first feature function in (7) as an example to explain this in detail. In essence, this feature function is the product of the probability of the estimated step length given the graph edge length $p(Z_t^l|S_{t-1}, S_t)$ and the probability of the estimated heading given the heading of the map graph edges $p(Z_t^\theta|S_{t-1}, S_t)$. The log-likelihood in (19) is maximized when the estimated step length is the same as the graph edge length because the item $p(Z_t^l|S_{t-1}, S_t)$ keeps the maximum for each state transition during the whole state sequence while the item $p(Z_t^\theta|S_{t-1}, S_t)$ is not significantly affected.

As a result, we must change the feature functions for the step constant learning. The essence of the step length parameter estimation is to find a γ that best supports the heading observations given the underlying graph of the map. To find this optimum γ, we start with an initial γ and equally divide the whole training trajectory into segments with the same length as the edge of the graph. Then a new feature function taking only the heading observation into account is defined as

$$f_t\big(S_{t-1}, S_t, Z_t^\theta\big) = \mathcal{N}\big(Z_t^\theta, \theta(S_{t-1}, S_t), \sigma_\theta^2\big), \tag{20}$$

where Z_t^θ is the heading observation at time t, $\theta(S_{t-1}, S_t)$ is the orientation of the edge from state S_{t-1} to state S_t, and σ_θ^2 is the heading observation variance.



(a) The step constant which maximizes $p(\boldsymbol{S}|\boldsymbol{Z})$ in (22) can minimize the tracking error.

(b) Different step constants of people with different heights, gender, ages, etc., showing the necessity of unsupervised online learning.

Fig. 4. Experiments results showing (a) the effectiveness and (b) the necessity of the proposed unsupervised online learning. The data were collected in the office environment from different people. Similar trajectories are used to learn the step constants shown in (b).

To solve the optimization problem in (19), we first consider the (soft) conditional EM approaches which optimize the model parameters in two steps

$$\begin{aligned} &\text{E-step}: \quad p(\boldsymbol{S})^i = \operatorname*{argmax}_{\boldsymbol{S}} p(\boldsymbol{S}|\boldsymbol{Z}, \gamma^{i-1}),\\ &\text{M-step}: \quad \gamma^i = \operatorname*{argmax}_{\gamma} E_{p(\boldsymbol{S})^i}[\ln\, p(\boldsymbol{S}|\boldsymbol{Z}), \gamma], \end{aligned} \tag{21}$$

where λ is the parameter to be estimated and the superscript i indicates the current iteration step.

Unfortunately the (soft) EM approach in (21) does not fit our model formulation. It is apparent that the expectation in the M-step can only be evaluated when the state space from the E-step $p(\boldsymbol{S})^i$ remains the same during the optimization process. However, the goal of the M-step is to find a γ^i that maximizes the log-likelihood. Therefore, γ^i experiences many different values during the optimization process in the M-step. From our training model formulation described above, different γ lead to training sequences of different lengths and thereby different state spaces for the state sequence $\boldsymbol{S}$. This contradicts the condition that the state space must remain the same for the expectation maximization and thus the algorithm fails.

Fortunately, it is justified in [34] that the most likely value of the state sequences given the model and the observed data, which is the result of the inference step, also maximizes the conditional likelihood. Therefore, by replacing the probability distribution in (21) with maximization, we transform the (soft) conditional EM to hard conditional EM, also known as Viterbi training. Then the hard EM is the process of iterating over the following two steps:

$$\begin{aligned} &\text{E-step}: \quad \boldsymbol{S}^i = \operatorname*{argmax}_{\boldsymbol{S}} p(\boldsymbol{S}|\boldsymbol{Z}, \boldsymbol{\lambda}^{i-1}),\\ &\text{M-step}: \quad \boldsymbol{\lambda}^i = \operatorname*{argmax}_{\boldsymbol{\lambda}} \ln\, p(\boldsymbol{S}|\boldsymbol{Z}). \end{aligned} \tag{22}$$

Experiments were conducted in an office environments (see Fig. 9b for test site map) to show the effectiveness of the unsupervised training algorithm. As shown in Fig. 4a, the step constant which maximizes the conditional probability $p(\boldsymbol{S}|\boldsymbol{Z})$ in (22) also minimizes the RMS error of tracking, which demonstrates the effectiveness of the proposed algorithm in estimating the step constants. Furthermore, as shown in Fig. 4b, we also find that the step constants are quite different not just for different pedestrians, but also for the same pedestrian in different environments as well, e.g., pedestrian D in an office environment ($D1$) and in a museum environment ($D2$). It is observed that the step constants of different people differ significantly from each other and from a default value which is derived from supervised off-line training. As shown later in Section 7.3, these differences of step constants have great impact on the tracking accuracy. Specifically, the step constant learned online can improve the RMS error of the indoor tracking by up to 30 percent compared to tracking with a default step constant. Therefore, unsupervised online learning of the step constant is critical for high-accuracy indoor tracking based on step length estimations.

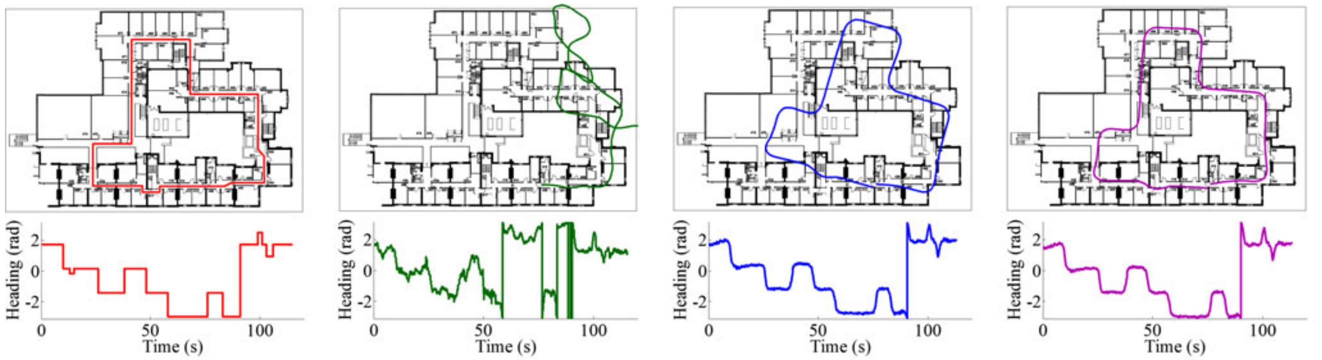
6.2 Heading Parameter Learning

The heading parameter learning, or specifically the initial heading bias (denoted with Δh) learning is much simpler than the learning of step length parameters, by fusing gyroscope and magnetometer data. Gyro sensors have high accuracy within a short time period while suffer significantly from long-term bias because of thermo-mechanical events. Magnetometers are the converse without long-term drift but lack of short-term accuracy due to the soft or hard iron effect. Therefore, the two types of sensors can compensate each other to offer highly accurate heading estimations.

However, the initial heading can only be estimated from the magnetometer and thus a large initial bias might be introduced. To make things worse, we have no other information from the map constraints before map matching algorithm converges, e.g., at the beginning, to correct the initial heading estimation. The initial heading is crucial because it impacts on all heading estimations in the future. Therefore, it is necessary to learn the initial heading bias as soon as possible and use this learned value to improve the performance of the CRFs-based tracking algorithm proposed in Section 4.

The proposed approach to learning the initial heading bias is to minimize the difference between headings estimated from gyroscope and magnetometers. After the system has worked for a period of time, we have both magnetometer and gyroscope readings. Then the initial heading bias can be easily learned by solving the following optimization problem:

$$\Delta h^* = \operatorname*{argmax}_{\Delta h} \sum_{t=1}^{t'} \left(\theta_t^{\text{gyro}} - \theta_t^{\text{mag}} - \Delta h\right)^2, \tag{23}$$



(a) Ground truth (b) Magnetometer (c) Gyroscope (d) Initial heading bias compensation

Fig. 5. Estimated trajectories and headings from a) ground truth, b) gyroscope, c) magnetometer, and d) initial heading bias compensation.

where Δh is the heading bias at time $t = 0$, t' is the time the pedestrian has walked before the bias estimation,[2] θ_t^{gyro} and θ_t^{mag} are the heading estimated from gyro sensors and magnetometers at time t, respectively.

The estimated initial heading bias is fed into the unscented Kalman filter mentioned in Section 3.1 to make accurate heading estimations. We have also conducted experiments in the office test site to show the effectiveness of the initial heading bias compensation mechanism. Starting from an arbitrary location in the office test site, a pedestrian walked a trajectory shown in Fig. 5a. Figs. 5b and 5c show the resulting heading estimations and trajectories with heading derived from magnetometer only and gyroscope only, respectively. It is observed that the trajectory is extremely noisy but immune from long-term drift with only magnetometer data. In comparison, the trajectory with gyroscope data is very clean during the experiment period but suffers significantly from the initial heading bias. Besides, the accuracy of this trajectory is also weakened by long-term drift near the end of the trajectory. Fig. 5d shows the headings and trajectories after the initial heading bias has been learned. It is observed that after the initial heading bias learning and correction, the accuracy of the trajectory is greatly improved. It is later shown in Section 7.3 that the initial heading bias learning mechanism can not only improve the tracking accuracy, but also greatly speed up the convergence of the map matching algorithm even in the absence of Wi-Fi data.

7 Evaluation

Sites. To demonstrate the real world applicability of the tracking system, MapCraft is evaluated and compared against competing approaches in three real-world settings, namely an office building, a market, and a museum. All of these have different floor plans as shown in Fig. 9 and methods of construction which affect the obtained sensor data. The office environment ($65 \times 35\ m^2$, where the majority of the tests have been conducted) is a multi-storey office building with a stone and brick construction, reinforced with metal rebars - testing was conducted on the fourth floor. The market ($108 \times 53\ m^2$) consists of a number of small shops, laid out over a single floor. Construction is brick and mortar, with a metal roof. The museum ($109 \times 89\ m^2$) is a multi-storey stone building with large, open spaces. Testing was conducted on the ground floor. Overall, 500 trajectories of average length 200 m were collected over 15 days. Error is expressed in [m] RMS.

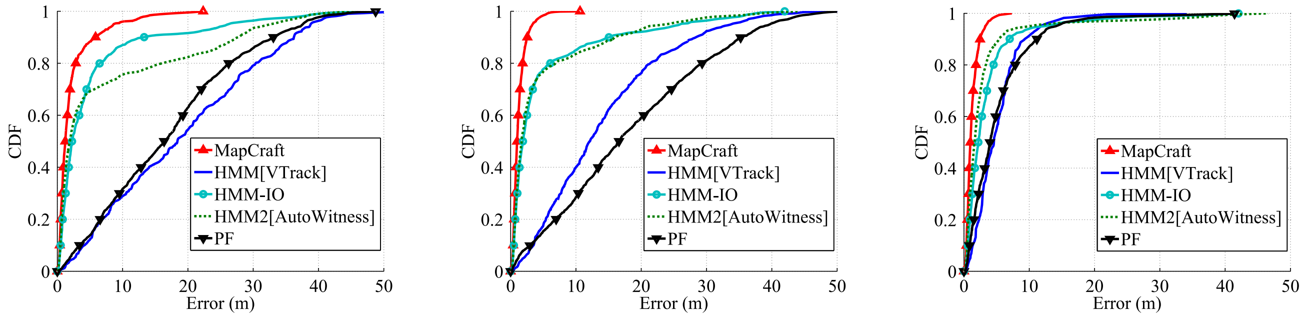
Participants. The variations between different people are taken into account by acquiring data from 20 people of different genders, heights, and ages. They may not appear in all three sites, but each one of them has participated in the experiments in at least two different sites. During the experiments, the subjects hold the mobile phones in their hands and then walk anywhere in the building without planned routes, to realistically capture real pedestrian motion, rather than artificial, constant speed trajectories.

Devices and implementation. Different types of mobile phones and pads are involved in experiments, including LG Nexus 4, Asus Nexus 7, Samsung Nexus S, Samsung Galaxy S IV, Samsung Galaxy S III, Samsung I9100G Galaxy S II, HTC Hero S and Huawei U8160. These mobile phones differ greatly in terms of sensors, functionality and price. But one thing in common is that they all run the Android operating system no earlier than version 2.3.6. A snapshot of our application prototype has been shown in Fig. 1.

Ground truth. To provide accurate ground truth, numbered labels were placed along corridors and within rooms on a 3 m grid. Using the device's camera, these were filmed at the same time experiments were conducted. The time-synchronized video streams were then mapped to locations on the floorplan, and intermediate locations interpolated using footstep timing, also obtained from the video.

Training. Fig. 8c shows the impact of training on the performance of MapCraft. Training alters weights for the various input features, away from the nominal case of weights 1 for all features (when the training iteration is 0). Several iterations of training were run on a set of training trajectories obtained from the office environment. The weights from each training iteration were then applied to each of the three data sets (trajectories from the office, museum, and market) for cross validation. Note that the RMS error of the trajectories in the office environment, where training was performed, is decreased with the number of training iterations, but only slightly (up to 9 percent). The RMS error of market and museum trajectories also do not vary much; it can even slightly increase because the training is performed in a different environment than testing. This implies that training is not critical to good performance and in practice, it can be skipped. The remainder of experiments, which are

[2] Empirically 30 $\sim$ 90 seconds in our experiments.



(a) Accelerometer + magnetometer

(b) Accelerometer + magnetometer + gyroscope

(c) Accelerometer + magnetometer + gyroscope + WiFi

Fig. 6. Error CDFs for various map matching approaches with increasing number of sensors a) gyro-free and b) gyro-aided and c) wi-fi- and gyro-aided.

conducted with weights 1 for all features, show that MapCraft is accurate without requiring careful training to a particular environment.

Competing approaches. We compare MapCraft with several state-of-the-art map matching algorithms. For readability, their names reveal the underpinning Bayesian estimation technique with references that point to source papers with implementation details: 1) **HMM** [33]: This algorithm uses a first order HMM, where states represent discrete positions, with equal transition probabilities between neighboring states (regardless of the vertex degree). Like VTrack [33], it uses ground truth position estimates as observations (but from RSS rather than GPS). It extends VTrack, by also exploiting position estimates from the inertial trajectory as observations; 2) **HMM-IO** [28]: This is an Input-Output first order HMM; the difference from the first order HMM is that inertial data is not used to generate observations at each step, but to calculate the transition probability between consecutive states; 3) **HMM2** [12]: In the second-order HMM, states are path segments connecting two locations in the map. Observations encompass both inertial displacement vectors and RSS-based position estimates. Transition and observation models are defined as in AutoWitness [12]; 4) **PF**: This algorithm uses a particle filter implementation similar to Zee [24]. Specifically, A total number of 2,000 particles are used. We also implement similar step counting algorithm and exactly the same particle update as Eqns. (3) and (4) in [24] with the stride length variation δ_i uniformly distributed within $\pm 30\%$ of the estimated stride length and heading perturbation $\beta_i \sim \mathcal{N}(0, 10^\circ)$.

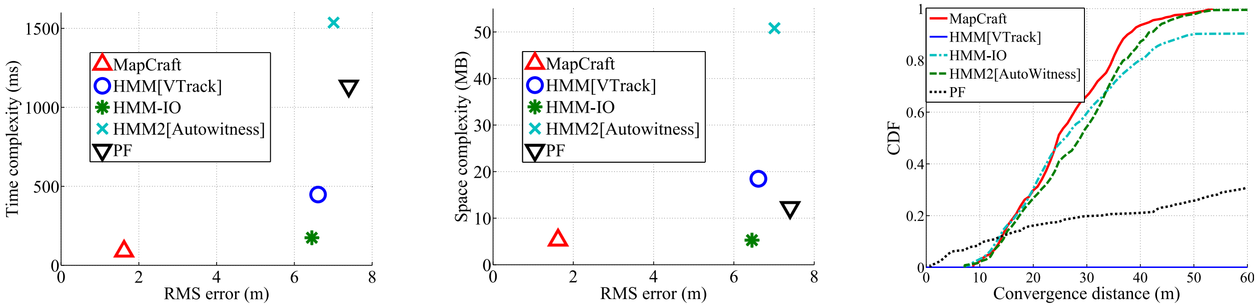
7.1 Performance Comparison

Accuracy/Sensor Tradeoff. The goal of the first experiment, which was conducted in the office site, is to explore the tradeoff between sensor usage and position accuracy. Different sensors provide different accuracy in location or heading estimations. For instance, gyro sensors help improve heading estimates, esp. in the presence of magnetic field distortions, as are typically encountered in indoor settings. Wi-Fi scans provide helpful information about absolute location. Fig. 6 shows three different regimes of map matching with increasing levels of sensor usage: 1) accelerometer and magnetometer only; 2) full inertial sensing: accelerometer, magnetometer, and gyroscope (no Wi-Fi); 3) full inertial sensing with periodic Wi-Fi RSS measurements. Notice that MapCraft significantly outperforms competing algorithms under all three regimes, typically resulting in errors two to three-fold lower than the next best approach.

Specifically, Fig. 6a shows the error CDF with only accelerometer and magnetometer measurements, such as would be used in a system aiming to consume minimal energy. The lack of gyro readings makes the heading estimation very inaccurate, especially in areas with high concentrations of metal. As a consequence, the whole raw trajectory is very noisy. These distorted trajectories greatly deteriorate the performance of existing methods whilst our approach remains robust to noisy sensors. One of the key reasons is the use of feature f_2 that handles heading error correlations. Fig. 6b shows the error distributions with full inertial measurements. The gyroscope provides more accurate heading estimation over a short period of time. However, over time, the accumulated error becomes excessively large. However, due to the features in our system, the accumulated error is gradually reduced in each step, which guarantees the accuracy of the heading estimation and yields accurate matching results. Fig. 6c shows the error CDF with periodic Wi-Fi measurements, taken every 16 seconds. These are used, in conjunction with a radio fingerprint map, to provide absolute position estimates. With these measurements, the performance of HMM and PF improves significantly. However, the performance of MapCraft increases further still with the combination of relative and absolute measurements. This is because it captures more features of the measurements across both time and space.

Generally speaking, different phone models involved in our experiments only have slight impact on the tracking accuracy (with a standard deviation of less than 0.2 m) except the one without gyro sensors (Huawei U8160). The reason lies in the fact that without gyro sensors the user's headings can only be estimated from earth's magnetic field which is significantly distorted in indoor environments. The noisy raw trajectories from only accelerometer and magnetometer would degrade the performance of MapCraft, as we can observe from Fig. 6.

The underlying reason for the superiority of MapCraft in tracking accuracy is the ability to model displacements of inertial trajectories without prior assumptions, as we have discussed in Section 4. The first-order HMM matches the locations rather than the displacements of the raw trajectory to possible location sequences in the map, and thus its performance can be easily degraded with a very noisy raw trajectory



(a) Execution time (WiFi-aided) (b) RAM usage (WiFi-aided) (c) Convergence distances (WiFi-free)

Fig. 7. Comparisons of (a) execution time, (b) memory usage, and (c) convergence distances of the various map matching approaches.

due to the increasingly accumulated location error from the inertial measurements. We have to make prior assumptions on the transition and observation probability distributions for HMM and HMM2, which are only approximations of the real world scenarios. In addition, the HMM category (HMM, HMM2, and HMM-IO) all assume the independence of observations given the state but, in reality, different observations are likely to be correlated, both spatially and temporally. As explained in Section 4, a CRF makes no similar assumptions, allowing us to capture these dependences and accurately model the tracking process as it is. As a result, CRFs offer superior performance, especially with tortuous trajectories and when there is a need for long-term tracking.

The PF approach is sensitive to motion sensing errors, especially the heading bias caused by magnetic disturbance of metallic objects, which is ubiquitous in indoor environments. In our experiments, the magnetic disturbance from one big metallic object usually lasts for 5 to 15 meters and can be as large as 30 degrees, which is very likely to mislead all particles into the wrong room, especially with very tortuous trajectories generated in our experiments. Since the PF approach only has local information about the region covered by particles, if all particles enter a wrong room it is hard to recover to the correct position. Furthermore, if the heading bias comes at the beginning of the tracking process, it is difficult for the particle filter to converge without knowledge of the user's initial pose.

Execution time and memory use. Next, we investigate the cost of MapCraft on a mobile phone (LG Nexus 4). We show that, not only it is more accurate, but it also offers significant computational and memory savings compared to existing approaches. This makes it lightweight and practical to run on resource-constrained mobile devices such as phones and wearable sensors. For a highly responsive pedestrian tracking application, the processing time for each step should be less than 300 ms. Meanwhile, the memory each application can use is quite limited in Android, e.g., less than 24 MB for earlier Android devices. Any algorithm with requirements exceeding these limits is not suitable for real-time pedestrian tracking with existing hardware. Fig. 7 shows the execution time and RAM usage of the various map matching algorithms.[3] Time (ms) in Fig. 7a is estimated as the average time to process a single step over the trajectories generated in the office site. Approaches using Viterbi decoding (CRFs, HMM, HMM2) have a worst case time complexity of $O(N^2T)$, where N is the number of vertices of the graph. In practice, different graphs vary in terms of connectivity resulting in varying run time costs. Note that MapCraft outperforms the other approaches, with an execution time of 10 ms, whilst obtaining the lowest RMS error.

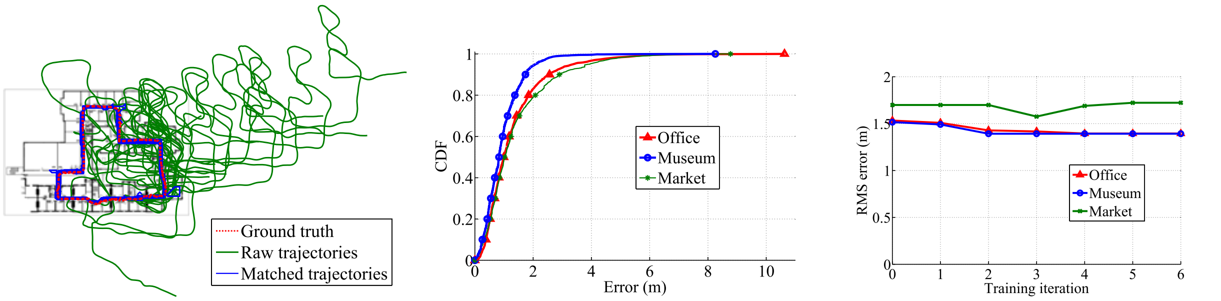
The memory requirements of the various approaches are shown in Fig. 7b. HMM2 in its current form cannot be deployed on an Android device due to the very high RAM usage of approximately 50 Mbyte. This is because it is based on a second order HMM which leads to exponential memory usage with the number of states as the entire transition matrix needs to be stored. HMM(Seitz) and PF are close to the limits of an Android application. HMM(VTrack) and MapCraft consume a similar amount of RAM (4 Mbytes), but the RMSE of VTrack is considerably higher.

The MapCraft is lightweight in both running time and memory usage because it neither stores transition or emission matrices (only several feature functions) nor performs expensive matrix operations. The HMM category, especially HMM2 whose state space is much bigger, take more running time and memory for processing transitions and emissions. The running time and memory usage of HMM-IO are comparable to MapCraft when the number of features is small, e.g., 2 or 3 in our experiments because the transition and emission probabilities are computed in a real-time manner.

The running time and memory usage of particle filter largely depends on the number of particles. The major computation cost comes from checking whether the position update of each particle violates the map constraints. The results shown in Fig. 7 are obtained with 2,000 particles.

Convergence distance. The proposed and competing algorithms are designed for online tracking. However, in the absence of Wi-Fi measurements and without knowledge of the initial pose, they initially incur a convergence cost, which we measure as the average minimum distance needed for the algorithm to find the correct location within an error of 3 m. Fig. 7c shows that for 97 percent of cases, MapCraft converges within 50 m and the next best, HMM2, within 60 m. The HMM (Seitz) and PF approach show considerably worse performance, in some cases never converging. Even with a very large number of particles (100 k), the PF approach fails to converge in many cases due to impoverishment. VTrack is unable to estimate the correct location,

[3] The execution time and memory usage of MapCraft are first tested in LG Nexus 4. All algorithms including MapCraft are implemented and tested in Matlab. Then the execution time and memory usage of other approaches are scaled relative to the values from the MapCraft real tests.



(a) Office test site: traversal of the same route 50 times.

(b) Location errors in office, museum, and market sites.

(c) Training in the office site and cross-testing in all three sites.

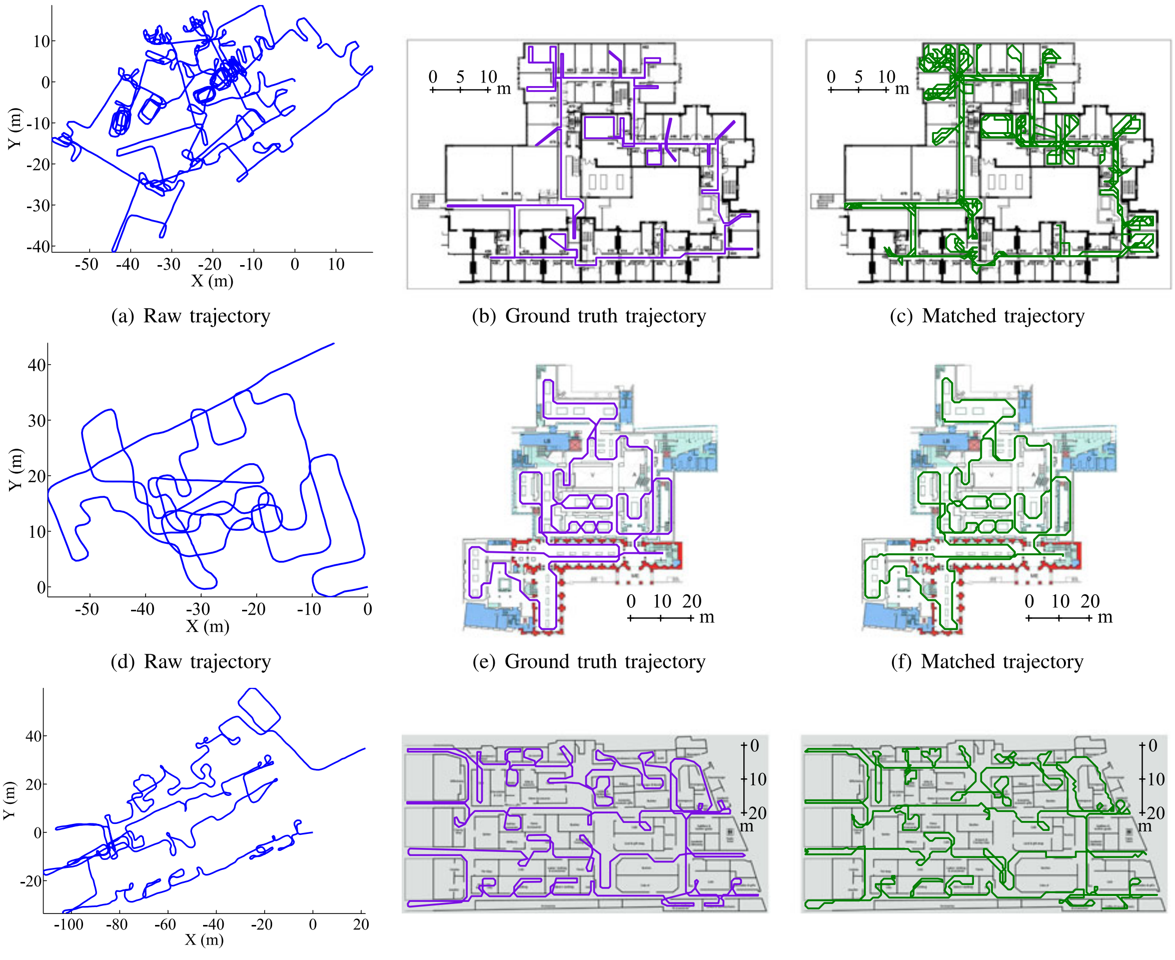
Fig. 8. Robustness of MapCraft in long-running and multi-site scenarios.

as it requires a good initial starting point estimation. Note that when Wi-Fi-aiding is used, all algorithms converges after 2-4 m.

7.2 Robustness

The next set of experiments is performed to examine the robustness of MapCraft and its applicability to real world scenarios. It must be emphasized that all these results are for the Wi-Fi-free case, i.e., the only input to the system is a floor plan and IMU data. The results that we had with Wi-Fi were even better, but we do not show them for space reasons.

Long-term tracking. First, we studied the repeatability of accuracy results as we run MapCraft for long time periods, resulting in inertial trajectories that are increasingly distorted with respect to the true trajectory. Fig. 8a shows the results of an experiment where the same route was followed 50 times in the office environment by different people with different mobile phones and the resulting trajectories calculated. Note that although the raw inertial measurements are significantly different each time, the map-matched trajectories are very similar. This shows that MapCraft is able to accurately reconstruct the correct trajectory, in spite of



(a) Raw trajectory

(b) Ground truth trajectory

(c) Matched trajectory

(d) Raw trajectory

(e) Ground truth trajectory

(f) Matched trajectory

(g) Raw trajectory

(h) Ground truth trajectory

(i) Matched trajectory

Fig. 9. Experiments in office (top), museum (middle), and market (bottom) site, showing raw, ground-truth, and matched trajectories.

TABLE 1
RMS Error, 97 Percentile Accuracy and Room-Level Accuracy of MapCraft in Three Sites

Site	office	museum	market
RMS error (m)	1.69	1.14	1.83
97 percentile (m)	4.10	2.37	4.53
Number of rooms	20	15	29
Room entry-events	357	360	82
Room identification (%)	93.0	100	96.3

excessive and varying metal-induced distortions to the heading estimation.

Multi-site performance. We then studied the robustness of MapCraft in a variety of environments, namely an office building, a museum and a supermarket. All of these have different floor plans and methods of construction which affect the obtained sensor data. The supermarket consists of a number of small shops, laid out over a single floor. Construction is mainly brick and mortar, with a metal roof. The museum is a multi-storey stone building with large, open spaces. Testing was conducted on the ground floor. The office environment (where the majority of the tests have been conducted) is a multi-storey office building with a stone and brick construction, reinforced with metal rebars—testing was conducted on the fourth floor.

Very complex and tortuous trajectories typically are the weakness of inertial tracking systems, due to drift and the absence of absolute anchor measurements. However, by using the map matching approach, very accurate reconstruction can be provided even when the user executes a complex trajectory. This is shown in Fig. 9. The CDF of location errors in the three environments are shown in Fig. 8b.

The room-level accuracy, defined as the percentage of matched trajectories entering the correct room, and overall 97 percentile accuracy is shown in Table 1. The room level accuracy is above 90 percent in all environments, which demonstrates the applicability of our approach to tackling real-world navigation problems. This compares well with other approaches based on extensive and multimodal radio fingerprinting [6].

The experiments in multiple environments have shown that the more constrained floor plans could lead to more accurate tracking performance, the reason being that the constraints of the environments are very informative of the location in that environment. An extreme example is the open space where no map constraints can be applied and hence the tracking accuracy is the worst—the same as the raw trajectories. In addition, we also found that the number of Wi-Fi access points does not really matter in terms of tracking accuracy. In the museum environment where we could detect less than 20 access points, we could achieve a higher accuracy than in the office environment where signals from over 100 access points were collected. This is because the Wi-Fi signals are not sufficiently stable to significantly improve a tracking system with RMS error less than 2 meters.

7.3 Parameter Tuning

We also evaluate the performance of our parameter learning algorithms. The office environment is the major test site where participants with different heights, genders, ages, etc. walk similar trajectories and then their RMS errors are evaluated with and without the parameter learning algorithm.

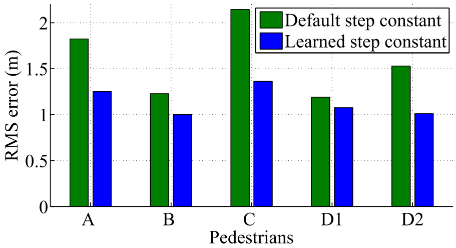


Fig. 10. Tracking accuracy (Wi-Fi-free) before and after the unsupervised step constant learning, discussed in Section 6.

Fig. 10 shows the Wi-Fi-free tracking accuracy improvement after the step constant learning, discussed in Section 6. It is observed that the tracking error with a default step constant can be almost twice as big as the tracking error with the step constant from the learning algorithm (Pedestrian C).

Wi-Fi-free experiments have also been conducted in the office environment to test the performance of the proposed initial heading bias learning algorithm. We have taken trajectories with an average length of 100 meters starting from 1,000 different locations randomly selected from the office environment. Then the RMS errors of these trajectories were calculated with and without the initial heading bias learning. The RMS errors along with 15 and 85 percentiles are shown in Fig. 11. It is observed that the initial heading bias learning algorithm is proved to improve the RMS error of the tracking by over 20 percent.

8 Conclusion

We demonstrated the merit of a novel map matching technique, based on the application of conditional random fields. We have shown how it is robust, being able to operate with very noisy sensor data; lightweight, running in under 10 ms on a smartphone; and accurate, achieving the lowest RMS errors compared with other state-of-the-art approaches. It does not require per-site training, which will allow for easy and widespread adoption, as the only information that is required to use our approach is a floorplan. We have also demonstrated that the proposed unsupervised parameter learning algorithms can significantly improve the tracking accuracy and convergence performance.

In the future, our system has the potential to make crowd-sourcing of Wi-Fi fingerprints practical, without requiring time-consuming manual scans. This is because MapCraft is able to establish a user's position using only dead-reckoned trajectories and a floorplan, without any

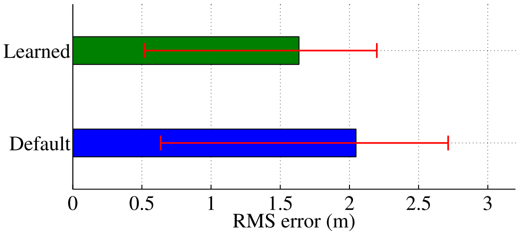


Fig. 11. The RMS error (Wi-Fi-free) with initial heading bias learning.

external information such as a starting location or knowledge of Wi-Fi access point locations. We believe that MapCraft has widespread application to a number of domains, as this single approach can be used with a wide variety of sensors and map information. One particularly relevant area is estimating location online and in real-time in resource-constrained body-worn sensors. In summary, we have presented a system that addresses the very pressing problem of providing accurate, low power, indoor tracking, that is responsive, robust and scalable.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their comments and suggestions to improve the paper. They also acknowledge the support of the EPSRC through grants EP/L00416X/1. Zhuoling Xiao is the corresponding author.

REFERENCES

[1] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, “A tutorial on particle filters for online nonlinear / non-gaussian Bayesian tracking,” *IEEE Trans. Signal Process.*, vol. 50, no. 2, pp. 174–188, Feb. 2002.

[2] M. Azizyan, I. Constandache, and R. R. Choudhury, “Surroundsense: Mobile phone localization via ambience fingerprinting,” in *Proc. Annu. ACM Int. Conf. Mobile Comput. Netw.*, 2009, pp. 261–272.

[3] P. Bahl and V. Padmanabhan, “Radar: An in-building RF-based user location and tracking system,” in *Proc. IEEE Conf. Comput. Commun.*, 2000, vol. 2, pp. 775–784.

[4] S. Beauregard, Widyawan, and M. Klepal, “Indoor PDR performance enhancement using minimal map information and particle filters,” in *Proc. Position, Location Navig. Symp.*, 2008, pp. 141–147.

[5] W. Chai, C. Chen, E. Edwan, J. Zhang, and O. Loffeld, “Ins/wi-fi based indoor navigation using adaptive kalman filtering and vehicle constraints,” in *Proc. 9th Workshop Positioning Navig. Commun.*, 2012, pp. 36–41.

[6] Y. Chen, D. Lymberopoulos, J. Liu, and B. Priyantha, “Fm-based indoor localization,” in *Proc. 10th Annu. Int. Conf. Mobile Syst., Appl. Services*, 2012, pp. 169–182.

[7] K. Chintalapudi, A. Padmanabha, and V. Padmanabhan, “Indoor localization without the pain,” in *Proc. 16th Annu. ACM Int. Conf. Mobile Comput. Netw.*, 2010, pp. 173–184.

[8] T. Cohn, “Efficient inference in large conditional random fields,” in *Proc. 17th Eur. Conf. Mach. Learn.*, 2006, pp. 606–613.

[9] I. Constandache, X. Bao, M. Azizyan, and R. R. Choudhury, “Did you see bob?: Human localization using mobile phones,” in *Proc. 16th Annu. ACM Int. Conf. Mobile Comput. Netw.*, 2010, pp. 149–160.

[10] B. Ferris, D. Fox, and N. Lawrence, “Wifi-slam using gaussian process latent variable models,” in *Proc. 20th Joint Int. Conf. Artif. Intell.*, 2007, pp. 2480–2485.

[11] E. Foxlin, “Pedestrian tracking with shoe-mounted inertial sensors,” *IEEE Comput. Graph. Appl.*, vol. 25, no. 6, pp. 38–46, Nov./Dec. 2005.

[12] S. Guha, K. Plarre, D. Lissner, S. Mitra, B. Krishna, P. Dutta, and S. Kumar, “Autowitness: Locating and tracking stolen property while tolerating GPS and radio outages,” in *Proc. 8th ACM Conf. Embedded Netw. Sensor Syst.*, 2010, pp. 29–42.

[13] J. Huang, D. Millman, M. Quigley, D. Stavens, S. Thrun, and A. Aggarwal, “Efficient, generalized indoor WiFi GraphSLAM,” in *Proc. Int. Conf. Robot. Autom.*, Shanghai, China, May 2011, pp. 1038–1043.

[14] B. Huyghe, J. Doutreloigne, and J. Vanfleteren, “3d orientation tracking based on unscented Kalman filtering of accelerometer and magnetometer data,” in *Proc. Sensors Appl. Symp.*, 2009, pp. 148–152.

[15] A. Jimenez, F. Seco, C. Prieto, and J. Guevara, “A comparison of pedestrian dead-reckoning algorithms using a low-cost mems imu,” in *Proc. Int. Symp. Intell. Signal Process.*, 2009, pp. 37–42.

[16] M. B. Kjaergaard, “Indoor location fingerprinting with heterogeneous clients,” *Pervasive Mobile Comput.*, vol. 7, no. 1, pp. 31–43, Feb. 2011.

[17] R. Klinger and K. Tomanek, “Classical probabilistic models and conditional random fields,” Dept. Comput. Sci., Technische Universitat Dortmund, Dortmund, Germany, Tech. Rep. TR07-2-013, 2007.

[18] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, “Conditional random fields: Probabilistic models for segmenting and labeling sequence data,” in *Proc. Int. Conf. Mach. Learn.*, 2001, pp. 282–289.

[19] A. LaMarca, Y. Chawathe, S. Consolvo, J. Hightower, I. Smith, J. Scott, T. Sohn, J. Howard, J. Hughes, F. Potter, J. Tabert, P. Powledge, G. Borriello, and B. Schilit, “Place lab: Device positioning using radio beacons in the wild,” in *Proc. 3rd Int. Conf. Pervasive Comput.*, 2005, pp. 116–133.

[20] F. Li, C. Zhao, G. Ding, J. Gong, C. Liu, and F. Zhao, “A reliable and accurate indoor localization method using phone inertial sensors,” in *Proc. ACM Conf. Ubiquitous Comput.*, 2012, pp. 421–430.

[21] P. Newson and J. Krumm, “Hidden Markov map matching through noise and sparseness,” in *Proc. 17th SIGSPATIAL Int. Conf. Adv. Geograph. Inf. Syst.*, 2009, pp. 336–343.

[22] H. Lenz, D. Obradovic, and M. Schupfner, “Fusion of map and sensor data in a modern car navigation system,” *J. VLSI Signal Process. Syst.*, vol. 45, nos. 1/2, pp. 111–122, 2006.

[23] J.-G. Park, “Indoor localization using place and motion signatures,” Ph.D. dissertation, Dept. Aeronautics Astronautics, Massachusetts Inst. Technol., Cambridge, MA, USA, 2013.

[24] A. Rai, K. K. Chintalapudi, V. N. Padmanabhan, and R. Sen, “Zee: Zero-effort crowdsourcing for indoor localization,” in *Proc. 18th Annu. ACM Int. Conf. Mobile Comput. Netw.*, 2012, pp. 293–304.

[25] V. Renaudin, M. Susi, and G. Lachapelle, “Step length estimation using handheld inertial sensors,” *Sensors*, vol. 12, no. 7, pp. 8507–8525, 2012.

[26] P. Robertson, M. Angermann, and M. Khider, “Improving simultaneous localization and mapping for pedestrian navigation and automatic mapping of buildings by using online human-based feature labeling,” in *Proc. Position, Location Navig. Symp.*, 2010, pp. 365–374.

[27] P. Robertson, M. Angermann, B. Krach, and M. Khider, “Slam dance: Inertial-based joint mapping and positioning for pedestrian navigation,” in *Proc. InsideGNSS*, 2010, pp. 48–59.

[28] J. Seitz, T. Vaupel, J. Jahn, S. Meyer, J. G. Boronat, and J. Thielecke, “A hidden markov model for urban navigation based on fingerprinting and pedestrian dead reckoning,” in *Proc. 13th Int. Conf. Inf. Fusion*, 2010, pp. 1–8.

[29] G. Shen, Z. Chen, P. Zhang, T. Moscibroda, and Y. Zhang, “Walkie-markie: Indoor pathway mapping made easy,” in *Proc. 10th USENIX Conf. Netw. Syst. Design Implementation*, 2013, pp. 85–98.

[30] Y. S. Suh, “Orientation estimation using a quaternion-based indirect kalman filter with adaptive estimation of external acceleration,” *IEEE Trans. Instrum. Meas.*, vol. 59, no. 12, pp. 3296–3305, Dec. 2010.

[31] M. Susi, V. Renaudin, and G. Lachapelle, “Motion mode recognition and step detection algorithms for mobile phone users,” *Sensors*, vol. 13, no. 2, pp. 1539–1562, 2013.

[32] A. Symington and N. Trigoni, “Encounter based sensor tracking,” in *Proc. 13th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2012, pp. 15–24.

[33] 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 *Proc. ACM Conf. Embedded Netw. Sensor Syst.*, 2009, pp. 85–98.

[34] H. Trinh, “A machine learning approach to recovery of scene geometry from images,” Ph.D dissertation, Dept. Comput. Sci., Toyota Technol. Inst. Chicago, Chicago, IL, USA, 2010.

[35] H. Wang, S. Sen, A. Elgohary, M. Farid, M. Youssef, and R. R. Choudhury, “No need to war-drive: Unsupervised indoor localization,” in *Proc. 10th Annu. Int. Conf. Mobile Syst., Appl. Services*, 2012, pp. 197–210.

[36] H. Wen, Z. Xiao, N. Trigoni, and P. Blunsom, “On assessing the accuracy of positioning systems in indoor environments,” in *Proc. 10th Eur. Conf. Wireless Sensor Netw.*, 2013, pp. 1–17.

[37] O. Woodman and R. Harle, “Pedestrian localisation for indoor environments,” in *Proc. 10th Int. Conf. Ubiquitous Comput.*, 2008, pp. 114–123.

[38] D. Wu, L. Bao, and R. Li, “Uwb-based localization in wireless sensor networks,” *Int. J. Commun., Netw. Syst. Sci.*, vol. 5, pp. 407–421, 2009.

[39] C. Xu, B. Firner, R. S. Moore, Y. Zhang, W. Trappe, R. Howard, F. Zhang, and N. An, “SCPL: Indoor device-free multi-subject counting and localization using radio signal strength,” in *Proc. 12th Int. Conf. Inf. Process. Sensor Netw.*, 2013, pp. 79–90.

[40] Z. Yang, C. Wu, and Y. Liu, “Locating in fingerprint space: Wireless indoor localization with little human intervention,” in *Proc. Annu. ACM Int. Conf. Mobile Comput. Netw.*, 2012, pp. 269–280.

[41] M. Youssef and A. Agrawala, “The horus WLAN location determination system,” in *Proc. Annu. Int. Conf. Mobile Syst., Appl. Services*, 2005, pp. 205–218.

Zhuoling Xiao is currently working toward the PhD degree in the Department of Computer Science, University of Oxford. His research interests focus on sensor networks, including localization, communication and coordination protocols for networked sensor nodes, and machine learning techniques for sensor networks and localization.

Hongkai Wen received the PhD degree in computer science from the University of Oxford. He is currently a postdoctoral researcher in the Department of Computer Science, University of Oxford. His research interests are in sensor networks, localization and navigation, and probabilistic machine learning.

Andrew Markham received the bachelor’s and PhD degrees in electrical engineering from the University of Cape Town, South Africa, in 2004 and 2008, respectively. He is currently an associate professor in the Department of Computer Science, University of Oxford, working in the Sensor Networks Group. His research interests include low-power sensing, embedded systems, and magneto-inductive techniques for positioning and communication.

Niki Trigoni received the PhD degree from the University of Cambridge in 2001. She is an associate professor in the Department of Computer Science, University of Oxford. She was a postdoctoral researcher at Cornell University during 2002-2004, and a lecturer at Birkbeck College during 2004-2007. Since she moved to Oxford in 2007, she established the Sensor Networks Group, and has conducted research in communication, localization and in-network processing algorithms for sensor networks. Her recent and ongoing projects span a wide variety of sensor networks applications, including indoor/underground localization, wildlife sensing, road traffic monitoring, autonomous (aerial and ground) vehicles, and sensor networks for industrial processes.