Indoor Localization using Place and Motion Signatures

by

Jun-geun Park

B.S., Seoul National University (2004)
S.M., Massachusetts Institute of Technology (2009)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2013

Author . . . .
Department of Aeronautics and Astronautics
May 16, 2013

Certified by . . . .
Seth Teller
Professor of Computer Science and Engineering
Thesis Supervisor

Certified by . . . .
Nicholas Roy
Associate Professor of Aeronautics and Astronautics

Certified by . . . .
Tommi Jaakkola
Professor of Computer Science and Engineering

Accepted by . . . .
Eytan H. Modiano
Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Indoor Localization using Place and Motion Signatures
by
Jun-geun Park

Submitted to the Department of Aeronautics and Astronautics
on May 16, 2013, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Aeronautics and Astronautics

Abstract

Most current methods for 802.11-based indoor localization depend on either simple radio propagation models or exhaustive, costly surveys conducted by skilled technicians. These methods are not satisfactory for long-term, large-scale positioning of mobile devices in practice. This thesis describes two approaches to the indoor localization problem, which we formulate as discovering user locations using *place* and *motion signatures*.

The first approach, *organic* indoor localization, combines the idea of crowdsourcing, encouraging end-users to contribute place signatures (location RF fingerprints) in an organic fashion. Based on prior work on organic localization systems, we study algorithmic challenges associated with structuring such organic location systems: the design of localization algorithms suitable for organic localization systems, qualitative and quantitative control of user inputs to "grow" an organic system from the very beginning, and handling the device heterogeneity problem, in which different devices have different RF characteristics.

In the second approach, motion compatibility–based indoor localization, we formulate the localization problem as trajectory matching of a user motion sequence onto a prior map. Our method estimates indoor location with respect to a prior map consisting of a set of 2D floor plans linked through horizontal and vertical adjacencies. To enable the localization system, we present a motion classification algorithm that estimates user motions from the sensors available in commodity mobile devices. We also present a route network generation method, which constructs a graph representation of all user routes from legacy floor plans. Given these inputs, our HMM-based trajectory matching algorithm recovers user trajectories. The main contribution is the notion of *path compatibility*, in which the sequential output of a classifier of inertial data producing low-level motion estimates (standing still, walking straight, going upstairs, turning left etc.) is examined for metric/topological/semantic agreement with the prior map. We show that, using only proprioceptive data of the quality typically available on a modern smartphone, our method can recover the user's location to within several meters in one to two minutes after a "cold start."

Thesis Supervisor: Seth Teller
Title: Professor of Computer Science and Engineering

Acknowledgments

To my research advisor, Prof. Seth Teller, for his tremendous guidance and support throughout my graduate school life.

To the members of the BMG/OIL project, especially Dr. Jonathan Ledlie, Ms. Dorothy Curtis, Benjamin Charrow, Jonathan Battat and Ami Patel, for their collaboration and feedback.

To my thesis committee, Prof. Nick Roy and Prof. Tommi Jaakkola, for helping me shape this thesis.

To my friends and family, for their unconditional support for many years.

To Mina, for always being there for me, in my good and bad times.

Some material from Chapter 3 and 4 appeared in previous publications [1,2].

Contents

Chapter 1

Introduction

Location information is a fundamental basis for mobile computing and its applications. Nowadays people carry mobile devices wherever they go; the location of the device naturally reflects its user's context, giving valuable information to the mobile applications and services he or she is using. For example, a recommendation "app" running in a smartphone can suggest the best place for dinner, considering the user's current location and recent movements. Also, location reveals the user's other contextual information; in prior work, location traces are used for inferring the user's transportation mode [3] or frequently visited places [4]. Sometimes, the location itself is what the user wants to know, in order to avoid getting lost.

Unfortunately, most location-aware applications to date are restricted to outdoor environments, as they rely primarily on the GPS (Global Positioning System) infrastructure [5] to obtain the user's current location. Consequently, we do not observe widespread use of location-aware mobile applications in indoor environments; GPS cannot provide timely and accurate position estimates indoors.

To extend the capability of mobile applications, researchers have worked on alternatives to GPS for indoor environments. Early work explored instrumenting a lab space with specialized beacons and mobile devices with dedicated receivers. For example, ActiveBadge [6] used infra-red beacons; Cricket [7, 8] used a combination of RF (radio frequency) and ultrasound signals. However, as IEEE 802.11 standards have been established as a *de facto* standard for short-to-mid range communication between mobile devices, with nearly universal coverage in most urban environments [9], indoor positioning has converged on methods that rely on existing wireless local area network (WLAN) infrastructure.

The key idea of the WLAN infrastructure–based localization is associating location-dependent characteristics of RF signals with physical locations. Such methods can find the user's location later by matching the signals, observed by the mobile device s/he is using, to the previously constructed RF map. Prior work based on this idea can be classified into either of two categories: *radio propagation model–based* methods and *fingerprinting* methods. The radio propagation model–based methods determine user location from distance estimates to three or more known locations of nearby wireless access points (WAP) by triangulation. The path-loss signal propagation model (§ 2.2.2), describing the relationship between RF signal strength (received signal strength indicator, RSSI) and physical distance between a transmitter-receiver pair of wireless devices, is used to estimate the distances. On the other hand, fingerprinting methods create a database that associates ambient wireless signal characteristics (e.g. signal strength or response rate) with physical locations. Such associations are called *location fingerprints* (Fig. 1-1). The user's device is then localized by finding the fingerprint in the database that is most similar to the currently observed signal and returning the associated location.

While prior work using these approaches has shown acceptable performance in terms of positioning error [10–12], neither is satisfactory for long-term, large-scale positioning of mobile devices in practice. Radio propagation model–based approaches cannot achieve high positioning accuracy because complex propagation phenomena of RF signals — multipath fading and shadowing due to walls, structures and furniture [13] — are not captured well by the signal propagation model. As a result, inaccurate distance estimates may result in incorrect location estimates [10]. On the other hand, fingerprinting methods require a high deployment burden, because a fingerprint database must be constructed by a manual survey process. Expert surveyors must systematically walk from room to room, gaining

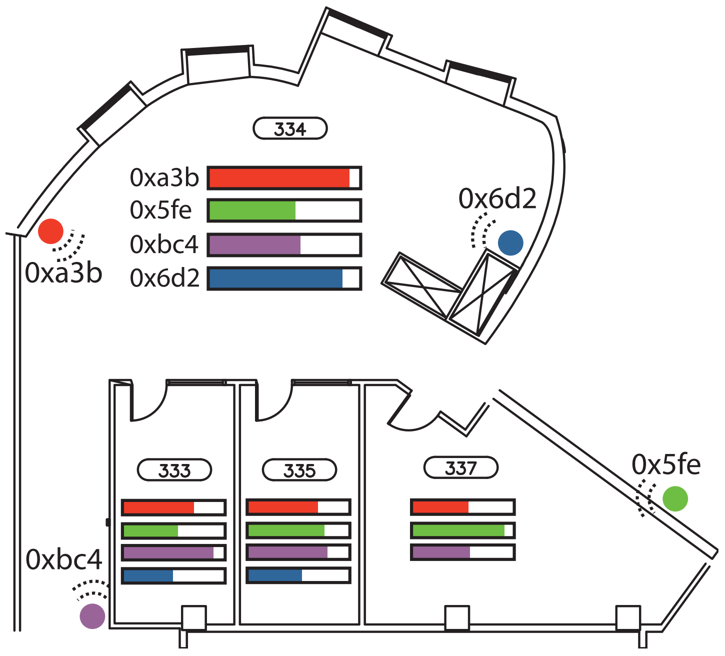


Figure 1-1: Illustration of RF fingerprints. The bars in each space illustrate the signal strength from each in-range AP.

access to all areas of a building to create the required fingerprinting database [11]. Even for a moderately-sized office building, this process can take several days and cost tens of thousands of dollars [14]. Furthermore, this process must be repeated whenever there is a significant change in the wireless environment of the target building.

Another approach that has been investigated recently is to make extensive use of sensors equipped in modern mobile devices. Smartphones and tablets these days have a multitude of sensors, which enrich mobile applications in a variety of ways. Since many of the mobile sensors, such as accelerometers, gyroscopes and magnetometers, capture physical properties that are directly or indirectly related to the user's motion and orientation, researchers have proposed localization methods that fuse low-level sensor measurement to estimate the user's current location. These methods rely on dead-reckoning, Kalman filters or particle filters [15–17]. However, the low-cost MEMS sensors in commodity devices are not well-calibrated and can drift quickly over time. Therefore, these methods typically require the device to be mounted in a specific position (waist or foot) [18] or they are aided by frequent external position fixes from a different source (e.g. WiFi) [15], having limited utility in practice.

Moreover, these methods have failed to recognize human as a central component of the localization model. Mobile devices *per se* do not represent how humans move in indoor environments and interact with the localization system. For example, in an area where the coverage of positioning systems is poor, a user may want to improve the system by actively contributing WiFi data, so that s/he or others can benefit from the improved coverage next time. Also, user paths in indoor environments can be concisely described using abstract terms describing human motions, such as "walking", "turning", or "riding an elevator", even without precise metric values. These aspects have not been well captured in the design of localization algorithms to date; the previous methods mainly concerned electromechanical properties of the mobile devices only, without modeling of end-users of the localization system. We argue that, localization algorithms, especially those for indoor environments in which motion patterns are shaped by human perception and activity, must place users as a central piece of their models.

In this thesis, we explore two approaches for the indoor localization problem, influenced by this idea. Specifically, we develop algorithms and systems that utilize "user aspects" in modeling indoor localization: *organic indoor localization* and *motion compatibility–based indoor localization*. These methods start from recognizing the localization problem as an inverse problem, in which we estimate user locations from observed "measurements" by sensors commonly available in modern mobile devices, including RF, acoustic/light [19], proximity and/or physical measurements such as acceleration and angular velocity, which convey information about the user's location and movements. These distinguishing sensor measurements, which are potentially unique to each space, comprise a *signature* for that space. Unique characteristics of an indoor space, shaped primarily by its position, shape, and walls and fixtures among others, form unique *place* and *motion signatures* for that space. As a specific example, RF fingerprints are place signatures as they are largely determined by the relative location of each space to nearby base stations as well as the material and the shape of walls and obstacles around the area. In another example, inertial sensor measurements induced by motions of the user carrying the device produce motion signatures that the spaces impose on his/her movements.

However, modeling the association between signatures and locations, and collecting required to capture these associations data present challenges to be addressed before a localization system can be used in practice. Assuming a localization system must deliver room granularity, the number of "examples" required for associating signatures (e.g. RF fingerprints) to locations (i.e. "training" a localization model) can be enormous if the target corpus is large. Also, user motions, which are better described in abstract terms than in raw metric sensor measurements, are not sufficiently utilized in previous research. The two threads of work that we present in this thesis attempt to improve the state-of-the-art of indoor localization methods by addressing these aspects.

1.1 Organization

In describing organic indoor localization (Part I), we address challenges associated with building and maintaining a crowdsourcing-based indoor localization system. Prior work on RF fingerprint–based indoor localization requires complete fingerprints to be collected in advance. This survey process is burdensome and costly, preventing RF fingerprinting–based methods from being widely used in practice. However, some later work [14,20,21] recognized that a localization system can incorporate users in its survey–use cycle, combining the idea of crowdsourcing with the fingerprinting method. In particular, the Organic Indoor Location (OIL) system [14] had individual end-users contribute WiFi location fingerprints, which were rapidly shared across end-users, improving coverage and quality of the localization service. While this approach can effectively reduce the burden of location surveying, it introduces its own set of challenges. On top of the basic OIL architecture, we address the challenges for effective and accurate positioning of users in an organic location system.

In the second part of the thesis, motion compatibility–based indoor localization (Part II), we explicitly model the way humans describe indoor routes within typical indoor environments. Because of metric, topological, or semantic constraints imposed by indoor maps, indoor navigation by humans is highly structured and can be described well with a small "vocabulary" of motions. This vocabulary, or a set of natural descriptions of human motions, describes a path of the user on the move. When a sequence of such user motions and a prior map are given, we show how to recover the originating user path on the map. To further increase the utility of the method, we describe how user motions and the matching representation of the prior map, route networks, can be generated automatically from smartphone sensor data and legacy CAD drawings, repectively.

1.2 Contributions

The following list summarizes the contributions made by this thesis:

1. Algorithms for organic indoor location systems:
    - Challenges in designing sustainable organic location systems
    - Design of a localization algorithm for organic location systems
    - Method for efficient collection of user inputs
    - Method for identifying and filtering erroneous user inputs
    - Analysis of, and a solution to the device heterogeneity problem
2. Motion compatibility–based indoor localization:
    - Automatic construction of route networks for indoor spaces
    - Notion of path compatibility between user motions and indoor paths
    - Accurate motion classification using mobile sensor data
    - Trajectory matching algorithm based on motion compatibility
    - Demonstration of the method with real user data using a handheld device

Part I

Algorithms for Organic Indoor Location Systems

Part I of the thesis presents algorithmic solutions to the problems arising in organic indoor location systems. We address issues that emerge when building a working organic localization system: collecting place signatures from end-users of the system efficiently and correctly, and using place signatures gathered from diverse mobile devices with heterogeneous RF characteristics.

We start by explaining the background that has motivated our work in Chapter 2. As our work builds on previous work on OIL systems, we give a brief overview of the OIL architecture. We also review related work here.

In Chapter 3, we describe algorithms to build an organic location system efficiently from scratch, when only a small amount or no data required for accurate localization are available. To build a working system quickly under this circumstance, it is important to guide users to contribute data for poorly covered locations to improve coverage, as well as to selectively employ user fingerprint contributions that are correctly associated with the right location. We present two approaches that deal with these problems.

Chapter 4 covers another important problem for organic indoor localization: device diversity. Because mobile devices can exhibit dissimilar RF characteristics due to differences in hardware and software, organic systems that take input from any end-users must account for such discrepancies. We analyze different localization models and present guidelines for designing a localization algorithm to work organically.

Chapter 2

Organic Indoor Location System: Background

In this chapter, we start by giving a motivation for our work. Then, we review prior work on RF-based localization algorithms for mobile devices. Next, we introduce *organic indoor location* (OIL), a crowdsourcing-based localization system that removes the initial survey burden required for traditional RF-based systems, and the associated challenges, for which solutions are presented in the following chapters.

2.1 Introduction

Incorporation of information about a user's location can enhance a variety of applications, including calendars, reminders, navigation assistants, and communication tools. For example, the Locale application automatically adjusts mobile phone behavior based on location [22]. However, most current location-aware applications are restricted to outdoor operation; they depend upon GPS, which requires clear sky visibility and may need minutes to provide an accurate location estimate.

Much of the research into alternatives to GPS has converged on methods that rely on existing wireless and cellular infrastructure (e.g., [10, 11, 23]). These methods share underlying elements: first, create a database that associates ambient wireless or cellular signals, or *fingerprints*, with physical locations; next, to localize, find the most similar fingerprint in the database to what one's device currently observes, and report the associated location. While such methods can localize indoors to within a few meters in regions with high infrastructure coverage [11], they have a high deployment burden. Surveyors must methodically walk from room to room, gaining access to all areas of a building to create the required fingerprint database [24]. For a moderately-sized office building, this process can take several days and cost tens of thousands of dollars, and must be repeated when the wireless infrastructure or building layout changes.

Because this deployment cost is prohibitive for all but the most managed environments (e.g., airports), researchers have developed systems in which users perform the required surveying activity [14, 20, 25, 26]. While these *organic* location systems reduce deployment and management burden significantly, they also introduce a new set of challenges. For example, if the fingerprint database is initially empty and grows in a piecemeal fashion, thus providing location estimates of spatially-varying quality, how can the system meaningfully convey to users both the need for more data, and the relative accuracy of its current location estimate? How can the system determine when to prompt user-surveyors for input? Insufficient prompting will not produce enough fingerprint data for a useful system, while too much prompting will annoy users. Additionally, user-surveyors will provide data of varying quality; how can the system sift through users' contributions to retain accurate contributions while discarding stale, erroneous or even malicious data? We study these questions in Chapter 3.

Another key problem for organic indoor localization is that *device diversity*, differing in RF characteristics between different types of mobile devices, introduces a new, complicating variable. Where expert surveys may use an expensive RF-scanner whose properties, e.g. dynamic range, have been rigorously calibrated, this level of standardization and equipment cannot be expected in organic surveying with typical consumer-grade laptops and cellphones. The "device heterogeneity" problem occurs when a user/surveyor's and a standard user's devices are different, which is the common case in organic location systems. Chapter 4 addresses the device heterogeneity problem, significantly expanding the potential of organic localization systems for real-world deployment.

2.2 Related Work

In this section, we review existing RF-based indoor positioning algorithms and systems for mobile devices and sensor networks, including prior work on reducing the survey burden of localization systems.

2.2.1 Positioning Systems with Dedicated Infrastructure

Early indoor positioning systems use dedicated hardware for locating people in indoor environments. A signal is transmitted either from the client hardware or from beacons installed in the area where positioning is required. The ActiveBadge [6] system by Want et al. tracks people and objects in an environment by attaching a badge, which periodically emits infrared signals. The Cricket location support system [7] uses a combination of RF and ultrasonic signals in order for the mobile client to infer the closest beacon unambiguously.

The Cricket system was later extended to provide the orientation of a mobile device [8]. The SpotOn system [27] used a custom RF transceiver with which the final location is computed by trilateration using RF signal strength measurements. While these systems demonstrated feasibility of mobile indoor positioning with reasonable accuracy, the need for instrumenting an area and for a client with dedicated hardware may be considered impractical in many settings.

2.2.2 RF Propagation Model–Based Methods

RF-based indoor positioning methods can be classified into two categories: radio propagation model–based methods and fingerprint methods. The radio propagation model methods first estimate the distance to the base station (802.11 wireless access point) from a measured received signal strength (RSS) using a path loss model. The positions of the base stations are assumed to be known and fixed. A most widely used path-loss signal propagation model (log-distance path-loss model, Fig. 2-1) is given by

$$P_i = P_0 - 10\gamma \log_{10} \frac{d_i}{d_0} + \epsilon \tag{2.1}$$

where P_i is the RSS for the base station i, P_0 is the received signal strength at a reference distance d_0, d_i is the distance to the base station i, γ is the path-loss exponent, and ϵ is a random variable modeling signal fading. The final position of the mobile is computed by

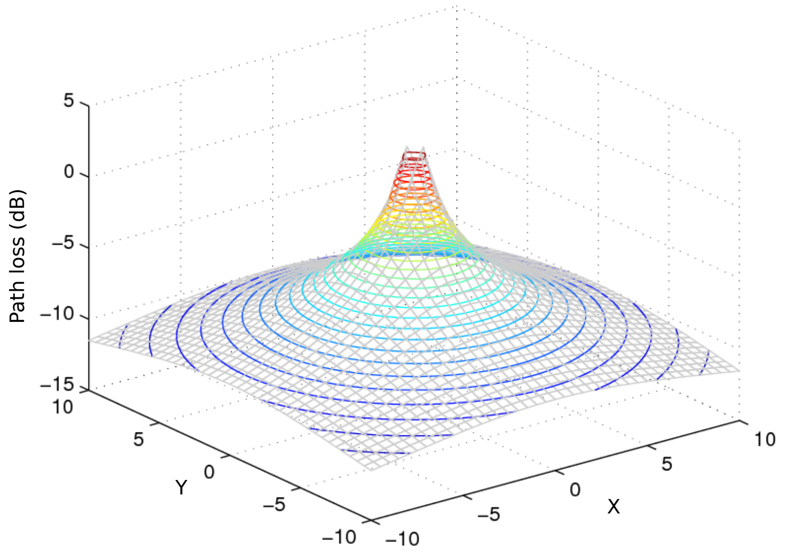


Figure 2-1: RF propagation model (path-loss model) with a base station located at origin (Equation 2.1).

triangulation or by setting an over-determined system of equations [10, 28]. Lim et al. [29] proposed an automatic calibration of model parameters in the signal propagation model by measuring RSS between the base stations.

A major drawback of the radio propagation model–based methods is that the path-loss model in Equation (2.1) cannot capture complex propagation phenomena of RF signal in indoor environments. The existence of walls, structures, and furniture in typical residential and office buildings makes the distance estimate from RSS inaccurate.

2.2.3 RF Fingerprint–Based Methods

Fingerprinting methods do not rely on the unreliable path-loss model, but rather construct an empirical database of RF signals at each location. They exploit the spatial variation in available radio frequency (RF) signals, such as 802.11 and cellular broadcasts, compiling this information into a map ("fingerprint") [10, 23]. Due to walls, distance, and other factors, the signals observed within a particular space differ substantially from those observed in other spaces, even those that are directly adjacent. The location of a mobile device can then be estimated by identifying the space within the map whose fingerprint best matches the fingerprint recently observed by the device.

RADAR [10] is one of the first fingerprinting-based RF localization systems. It used k-nearest-neighbors (k-NN) on training signal data captured at each location in the area in order to find the best matching location among the surveyed locations. It also compared the fingerprinting methods with a radio propagation model–based method, demonstrating that the former (k-NN with training points) performed better at the expense of survey efforts. The RADAR system was later extended to consider temporal continuity in user motion [30].

Researchers have investigated the use of various supervised learning algorithms on the RF-based indoor positioning problem, such as neural networks [31] or support vector machines [32]. In particular, Bayesian localization methods, which have been used mainly for robot localization [33], have been adopted to the mobile indoor positioning problem. Haeberlen et al. [11] modeled the signal strength distribution as the Gaussian density function and used a Bayesian classifier for localization. The Horus system [12, 34] also estimated the location of the mobile using a Bayesian classifier with a signal strength distribution at each location. The system was further extended to handle signal correlation across successive samples and small-scale variations in WiFi signals [12, 35].

When applied to continuous tracking problem using 802.11 WLAN signals, sampling-based approaches such as particle filters, have been used in order to represent the state space (physical coordinates) in continuous coordinates. For instance, Berna et al. [36] proposed a Monte Carlo Localization, a particle filter applied to the Bayesian localization problem, that uses wireless signal strength for updating the importance weight of particles. Biswas and

Veloso [37] pursued a similar approach for localizing autonomous indoor robots.

More sophisticated Bayesian statistical approaches have been adopted for indoor positioning problem. For example, Madigan et al. [38] proposed hierarchical Bayesian graphical models for WiFi-based indoor location estimation problem. Bayesian networks were constructed upon the path-loss model and the positions of the access points were assumed to be known. While the system is able to incorporate domain-dependent prior knowledge because of the flexibility of the hierarchical Bayesian models, it is not adequate for the real-time positioning because it uses Monte Carlo Markov Chain simulation for inference. Letchner et al. [39] developed a similar hierarchical Bayesian model for WiFi signal distributions.

2.2.4 Localization in Sensor Networks

The localization problem has also been studied in the context of sensor networks. Localization for sensor networks and RF-based mobile positioning share many common characteristics; they both assume a small mobile sensing platform which has a communication capability based on radio frequency or an equivalent technology. However, they are different in many of their basic assumptions. For example, mobile localization usually involves positioning of individual mobile computers, whereas localization in sensor networks tackles positioning of a group of sensors; mobiles in mobile localization communicates with base stations, whereas sensors in sensor networks usually communicates with each other from which proximity or distance measurements are obtained. In the review below, we refer to anchor nodes, or *anchors*, as the sensors with positions known from other means, i.e., fixed at known coordinates or equipped with GPS.

Many simple yet effective localization algorithms are based on proximity information. Proximity information is used for bounding the maximum distance within which a pair of sensors can be located from each other. In order to provide global position estimates, proximity-based approaches often assume existence of large number of anchors in the region so that each node can approximate its location from single-hop proximity information. Examples of proximity-based methods include work by Bulusu et al. [40], APIT (Approximate Point In Triangulation) [41], and bounding box–based algorithms [42, 43]. Proximity-based approaches are simple and cost-effective because they do not require accurate ranging capability, and thus can operate on simple low-cost devices. However, the localization error is relatively large and they can require high anchor density.

Lateration-based algorithms refer to localization algorithms based on lateration, which is a class of geometric methods to locate an object given distance measurements from multiple reference positions [44]. The simplest case is when a position of a node in a two dimensional space can be uniquely determined from three non-collinear node positions and distances, which is called trilateration. If only a small number of anchor nodes are available in the network, direct distances to anchor nodes are unavailable for sensor nodes; therefore they are often approximated [45, 46]. Another approach is to build a network localization graph incrementally [47–50], while incorporating error-control mechanisms in order to prevent error accumulation during the incremental reconstruction process. Several previous works extended the basic lateration method to mobile sensor networks [51, 52].

Another approach to the sensor network localization problem is to formulate it as a global optimization problem, for example, by a second-order cone programming given the known anchor coordinates [53], or as semi-definite programming [54]. Although this global optimization formulation is mathematically elegant, it has two major problems: it is unavoidably centralized, and it becomes computationally intractable as the size of the networks grows. Researchers have used the Cramér-Rao bound, a statistical lower bound on the covariance of an unbiased estimator, to characterize error behavior [55, 56]. The sensor network localization problem has also been formulated as an instance of learning methods. Multidimensional scaling can be naturally applied to the sensor network localization as pairwise distances between sensor nodes are given [57–59].

Finally, Bayesian inference techniques have been applied to the sensor network localization problem. Bayesian inference methods provide a disciplined mechanism to incorporate measurement uncertainty into the positioning estimates. Ihler et al. [60] model sensor networks as Markov random fields and apply nonparametric belief propagation which is a sample-based variant of the belief propagation method. For mobile sensor networks, sequential Bayesian estimation methods have been applied by considering the limited sensing capability of individual sensor nodes [61, 62].

2.2.5 Mobile Positioning with Minimal Survey Effort

Researchers have studied positioning methods that require only minimal or no survey effort. Krumm and Platt [63] formulated the positioning problem as a kernel regression problem, interpolating a signal strength vector into a location that is not surveyed. Their work demonstrated that the indoor positioning with reduced survey effort was feasible where only 40% of the locations were surveyed and the positioning error increased only about 20% in their deployment.

Some previous work approached the problem as a graph reconstruction problem where the locations of only a small fraction of nodes are known *a priori*. It is often formulated as an optimization problem that minimizes a certain kind of error function, which models mismatch between the measured sensor data such as RSS and the predicted sensor measurements given the configuration of nodes. The path-loss model or its equivalent is used in order to predict an RF signal measurement given a distance. As described in Section 2.2.4, this approach is conceptually very similar to the localization algorithms in sensor networks where pairwise distances are given.

LaMarca et al. [64] introduced an iterative algorithm that constructs a graph from a small number of known anchor locations. Pan et al. [65] presented a "co-localization" method, in which mobile positions and AP positions are computed simultaneously. It is formulated as a semi-supervised learning problem using a graph Laplacian, which is an approximation of the data manifold in the signal space.

There also exist wireless positioning algorithms that avoid pre-deployment survey effort altogether. As in the graph reconstruction approaches presented above, they seek to find a relative configuration of measurement locations that is consistent with the associated WiFi signal measurements. The resulting configuration can be followed by an anchoring step if the absolute locations for some measurements are known by other means, e.g. GPS. Chintalapudi et al. [66] formulated it as an optimization problem for finding AP and mobile locations. Huang et al. [67] adopted GraphSLAM [68], a simultaneous localization and mapping paradigm for robotics applications, for the WiFi-based positioning problem. GraphSLAM can also be understood as a graph reconstruction method. On the other hand, Ferris et al. [69] applied Gaussian process latent variable models [70], a dimensionality reduction method based on latent variable modeling with Gaussian processes. This work approached the problem from the perspective of dimensionality reduction of a high-dimensional WiFi signal data. While these methods have demonstrated that it is feasible to compute the relative locations of WiFi measurements on users' mobile device, they did not provide a satisfying solution to how to ground the relative configuration on a physical representation of the world, such as location names or floor plans.

2.2.6 Crowdsourcing-Based Positioning Methods

RF-fingerprinting–based methods (§ 2.2.3) have been the most popular approach to indoor localization using mobile devices because of their ability to capture unique RF characteristics per location without having to relying on existing infrastructure (§ 2.2.1) or inaccurate propagation models (§ 2.2.2). However, they require expert surveying to collect fingerprints necessary for localization. Methods based on expert surveying have practical, cultural and technical downsides preventing them from achieving widespread use. On the practical side, such methods have a high fixed cost, as they require an initial site survey to build and populate the signal strength map. This deployment burden typically requires a few person-days of careful and spatially comprehensive survey effort by skilled technicians. This approach faces a cultural barrier as well, as members of a community may feel reluctant to allow strangers into certain areas such as private offices. A technical challenge is that site survey data may become outdated over time, e.g. through access point reconfiguration, re-positioning or replacement, each of which may degrade or invalidate subsequent location estimates.

These factors led to the development of user-generated, or *organic* localization systems, where the initial, comprehensive site survey is replaced with *ad hoc*, incremental collection of data by individual users [14, 20, 25, 26]. Organic localization merges the "survey" and "use" phases that were distinct in earlier work [10, 11] into a single phase, where the users of the system are also prompted to construct the signal strength map. After a handful of early users populate the map for a building environment, a typical user will enjoy high-quality location discovery with minimal individual effort.

Outdoors, GPS coordinates can be used to annotate user input and build the fingerprint-to-place mapping. This process, called *war-driving* [23], generates fingerprints that can later be used for location determination by devices that lack GPS but have WiFi [71]. War-driving can be considered a form of organic data collection, but its dependence on GPS limits it to outdoor use.

User input has also been employed in indoor positioning systems. ActiveCampus [21, 25] uses a prediction-correction mechanism: first, the system builds a coarse-grained fingerprint, and then users can correct a location estimate by providing a "virtual AP". Bolliger [20] developed the RedPin localization system which uses WiFi, GSM, and Bluetooth as sensing devices. Like OIL system presented in this work, RedPin does not require an expensive training phase and generates location fingerprints from user input. Krumm and Hinckley's NearMe infers proximity of users by comparing user-generated WiFi signatures [72]. Barry et al. [26] conducted a year-long study of a user-trained localization system and showed its utility. None of these systems addressed challenges associated with organic input, such as spatial uncertainty, labeling errors, and device diversity.

In the next section, we present our implementation of the crowdsourcing-based localization, the Organic Indoor Location (OIL) system [1, 14], which aims to provide scalable localization services for context-aware mobile applications.

2.3 Organic Indoor Localization: Overview

We have developed the Organic Indoor Location (OIL) system for user-supported localization system [1, 14]. The key idea of the OIL system is to combine the power of a community of users with an indoor positioning system, by encouraging individual users to contribute location fingerprints with location labels in an organic fashion (Fig. 2-2). Once a user selects the current location on a graphical user interface, OIL automatically associates the location with RF signal scans collected around the same time, forming a *bind*. Location fingerprints collected in this manner constitute a shared fingerprint, which is used for training the positioning algorithm at a shared server. This trained model can be shared by all users, contributors and non-contributing users alike, who use the shared model for later positioning.

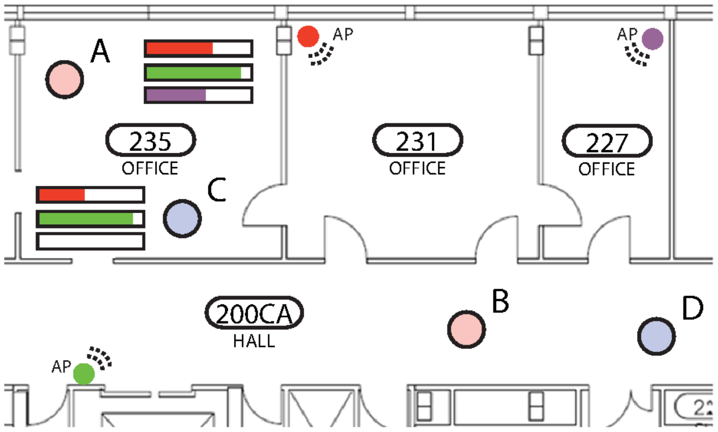


Figure 2-2: In an "organic" indoor location system, a small fraction of users contribute RF signatures for each space, while most users' devices simply use the resulting database for positioning. For example, users A and B could be contributors to the office environment shown, whereas C and D are non-contributing users. The bars next to A and C represent the set of signal strengths (the fingerprint) seen by their devices when they are in room 235. If A and C have different devices, the fingerprints observed may differ, even if they are in the same room at the same time.

The idea is that as more and more contributions populate the system, the normal users will also benefit from them with higher quality and less interface burden. The architecture and the user interface of the client software of the OIL system are shown in Figures 2-3 and 2-4, respectively.

The client software was implemented using Python on Nokia N810 Internet Tablets. Users are presented with a clickable map (Fig. 2-4), where polygonal space contours and space names have been automatically extracted from AutoCAD files [73].

Given this outline of our organic localization system, we now describe the system design and user interface related issues that were handled in prior work [14] in Section 2.3.1, for completeness. Next, we describe the Bayesian localization algorithm, which estimates the location of a user device by identifying the space whose fingerprint best matches WiFi signal strength observed by the device (§ 2.3.2). Then, we explain challenges of the organic localization (§ 2.4), for which we present algorithmic solutions (Chs. 3 & 4) to make the system practical in real environments.

2.3.1 Design and User Interface

The main thrust of the OIL system is seamless integration of data collection into the use phase of the location service. On the other hand, the organic data collection process should not overburden users, i.e., the system architecture should provide a fluent user experience so

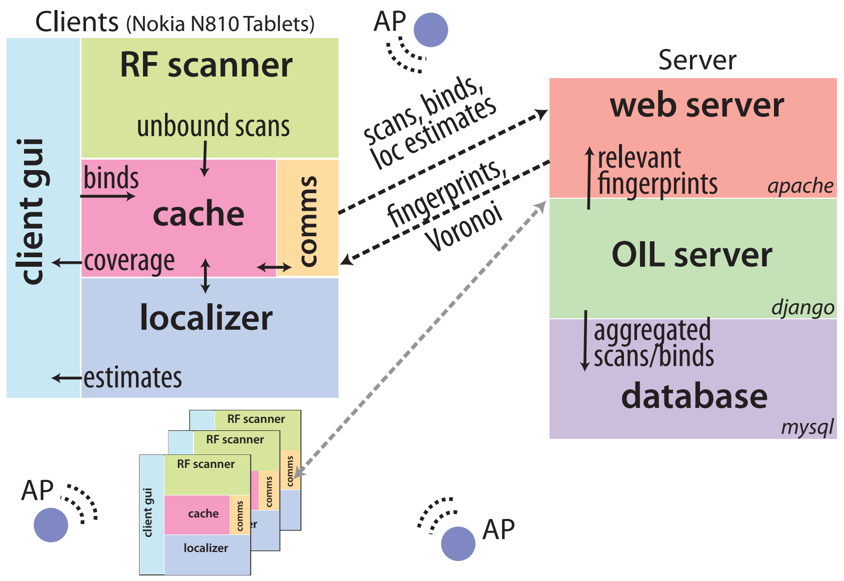


Figure 2-3: OIL system architecture. OIL client software on the user's mobile device collects signal strengths from nearby APs. When the user indicates the mobile's current location, the OIL client associates the location with RF scans collected around the same time and send the bind to OIL server. The server aggregates it with other binds from the same location, forming a WiFi fingerprint for that location. The WiFi fingerprints are transferred to the user clients and used for real-time positioning on the client.

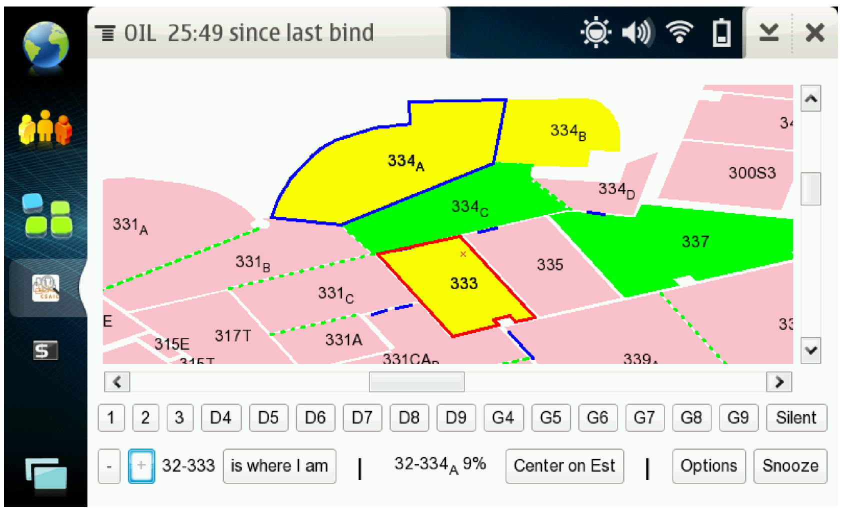


Figure 2-4: OIL user interface. The user interface of OIL system displays the current location estimate to the user on a graphical map, and provides an interface to collect user binds in an efficient way. Users can select the currently-occupied space on a floor plan and contribute the location fingerprint to the system.

that the user only needs to invest minimal effort in the process. The OIL system satisfies these considerations from various perspectives — from the user interface to the architectural level. Below, we summarize the key aspects of the OIL system from this viewpoint.

Data Collection and User Prompting

User-collected fingerprint data are essential in realizing organic location systems. The OIL system takes user binds, an association between a RF fingerprint and a location label, either from user-initiated contribution or by user prompting. Users can upload fingerprint data for the current location anytime they want to do so. The system also prompts for user input when it deems appropriate for improving localization performance (§ 3.1.2).

Interval Binds

As opposed to snapshot-based data collection methods [11], which take the present location of the mobile as input, the OIL system solicits user's past and future duration in that space. This *interval bind* mechanism has several advantages. First, the system can collect far more training examples for the input of the localization model, as users often spend a long time in each given location. Second, the system can suppress other UI notifications for user prompting during this period, reducing user burden.

Fingerprint Caching and Pre-Fetching

The central goal of the OIL system is to provide location estimates to the user in a timely manner. That is, the client must be able to produce location estimates without necessarily connecting to the fingerprint server. To support this seamless location service, OIL implemented fingerprint pre-fetching and local-caching scheme on the client device. The OIL client maintains a set of fingerprints adaptively, by receiving new fingerprints from the server based on similarity to the fingerprint at the current location, and by removing stale cached entries. Another advantage of this adaptive scheme is that it effectively reduces the computation of the localization algorithm.

2.3.2 Localization Algorithm

In principle, any ad-hoc or principled supervised learning algorithm can be used for WiFi-based localization. However, a localization algorithm for practical organic location systems must be efficient in two ways. First, the localization model computed from organic location fingerprints must be able to be updated frequently, because the location fingerprints will be updated frequently in a piecemeal fashion as users contribute a small amount of WiFi data to the system. Second, since the system computes user position at the client side in order to preserve user privacy, the location inference must be simple enough to run on consumer-grade mobile devices without hampering other user applications.

Based on these considerations, the OIL system uses a naïve Bayes classifier, which has a number of properties that make it suitable for user-contribution–based localization systems for mobile devices: low overhead of model update, simple computation on the client, and good performance despite its simplicity.

Given a set of fingerprinted locations (*the training data*) and a WiFi scan observation (*the test data*), the Bayesian localization method infers the most likely location $\hat{l}$ of the mobile device using Bayes' rule. Let L and O denote the random variables for location and observation respectively. Given a WiFi scan observation $o \in O$, the posterior probability of being in location $l \in L$ is given by Bayes' rule as:

$$p_{L|O}(l|o) = \frac{p_{O|L}(o|l)\; p_L(l)}{p_O(o)}. \tag{2.2}$$

Note that the observation likelihood $p_O(o)$ is fixed and can be ignored in what follows. If we assume that the prior probability, $p_L(l)$, is uniform, the *maximum a posteriori* (MAP) estimate, with which the posterior probability is maximized, is given as follows:

$$\hat{l} = l_{MAP} = \underset{l \in L}{\operatorname{argmax}} \left[p_{O|L}(o|l) \right]. \tag{2.3}$$

Our basic localization model uses WiFi signal strength measurements as observations. In the signal strength–based localization, each observation consists of a vector of signal strengths $O = (S_1, S_2, \ldots, S_k)$ for k access points. Suppose, at the positioning phase, only $m \leq k$ access points are observed, each with a signal strength value. Let M denote the indices of the observed access points. Then the decision rule of the naïve Bayes classifier for the most likely location $\hat{l}$, with a uniform prior, becomes:

$$\hat{l} = \underset{l \in L}{\operatorname{argmax}} \left[\prod_{i \in M} p_{S_i|L}(s_i|l) \right]. \tag{2.4}$$

The training stage for the localization algorithm in Equation (2.4) is to estimate the *class-conditional probability* $p_{S_i|L}(\cdot|\cdot)$ for each location from user-contributed signal strength data. We model it as a histogram, or a Parzen window estimator (§ 4.2.1). The model update is very simple and efficient, as it requires only updating the histograms associated with a specific location and access points, given a user input. The computed class-conditional probability table is shared among end-users of the system. Client software running on each mobile device computes the most-likely location with Equation (2.4).

We apply m-estimate smoothing to avoid the zero-probability problem in Bayesian classification as well as to provide a regularization effect [74]. Given a discretized probability mass function $\hat{p}_X(x)$ computed from either a histogram or a kernel estimator, our final probability estimate $p_X(x)$ using the m-estimator is:

$$p_X(x) = \frac{N\hat{p}_X(x) + \Phi\bar{p}_X(x)}{N+\Phi} \tag{2.5}$$

where N is the weight of the observed histogram, $\bar{p}_X(x)$ is a "prior" probability, and Φ determines how much weight we attribute to the prior $\bar{p}_X(x)$. This regularization is particularly essential for organic location systems where the number of training inputs for many locations is very small and uneven because of the organic nature of the system. Our implementation set $p_k(s_i)$ to a uniform distribution for simplicity, and m to the number of effective histogram bins (we used $m = 70$ since signal strength typically ranges from -94 to -25 dBm).

2.4 Challenges for OIL

While this approach can effectively mitigate the burden of an exhaustive site survey, it introduces its own new set of challenges, ranging from design and user interface to algorithmic challenges. In particular, there are algorithmic challenges associated with the efficient and accurate operation of localization algorithms in organic location systems.

The first challenge stems from the fact that the signal strength map, constructed from user-provided information, may be incomplete and may include erroneous data. Therefore, the system must be able to control the quantity and quality of user inputs for the localization system to bootstrap properly from scratch. Also, organic localization systems are expected to encounter some level of noisy user contributions. In particular, users will not always indicate the right room when they are prompted to make a bind. These erroneous user inputs may pollute the organic fingerprint database with scans acquired in the wrong spaces. Therefore, identifying erroneous contributions is a key problem in organic localization. In Chapter 3, we propose algorithmic solutions to tackle these challenges: a mechanism for evaluating *spatial confidence* when inferring a specific location, and prompting for additional binds to improve coverage; and a method to identify erroneous binds in order to improve localization accuracy.

The second complication is that users of the organic localization system carry different types of mobile devices. Unlike conventional RF-fingerprinting methods for which signal characteristics of the survey device are known and calibrated, varying RF characteristics due to various factors, such as different size, operating system, driver or calibration, can make the localization algorithm vulnerable to even small differences across devices, potentially limiting interoperability between different fingerprint producers and consumers. Hence, the localization algorithm must be robust against the diversity of user devices in organic location systems. Chapter 4 presents an analysis of, and design considerations for a localization system in this scenario.

Chapter 3

Growing an Organic Indoor Location System

To "grow" an organic indoor location system from scratch efficiently, the system must incorporate user input for locations without sufficient binds. Also, for the system to be "healthy", it should be able to figure out which inputs arise from faulty contributions, and must not incorporate those erroneous inputs into the fingerprints. This chapter proposes algorithmic solutions for these problems. Section 3.1 proposes a Voronoi diagram–based spatial uncertainty metric and a corresponding policy to prompt a new user input. Section 3.2 presents a clustering-based algorithm to handle erroneous user input. Evaluations from simulation and live deployments are described in Section 3.3.

3.1 Voronoi-based User Prompting

In survey-based positioning, the survey provides a snapshot of ambient RF for all spaces where client positioning is desired. Based on the obtained signal strength map, a standard localizer can then find the space that matches the fingerprint observed by a client device with the highest probability. Organic positioning, however, begins with an empty database which is gradually populated with user-provided fingerprints. If the fingerprint database is empty or if the database does not include any of the RF sources that a client sees, then the organic localizer outputs an "unknown location" response. However, when the database is only partially populated, the localizer's use of incomplete information may bias its predictions. For example, consider the extreme case where only a single bind has been made; if a database of known fingerprints contains a single location, the localizer will output that space as its prediction, even if the wireless scan observed by the device overlaps only slightly with the fingerprint associated with that location. We therefore require a method for estimating and conveying the localizer's spatial *confidence* in its output prediction. This confidence measure can be displayed along with the location estimate in a way that is intuitive to contributors and non-contributors alike. We can also use the confidence measure as the basis of a policy for requesting user input. For organic systems, in order to increase coverage, it is useful to occasionally prompt users for their location. However, there is a trade-off between providing imprecise estimates due to lack of coverage, and irritating users with too many bind requests, especially when the fingerprint database is only partially populated. When should a user be prompted with an explicit location request? Our system prompts whenever localizer confidence falls below a threshold.

During the development of OIL, we considered several prompting policies and their implications for coverage. The simplest policy is to prompt all users at regular intervals, regardless of their location or estimate confidence. However, this method was intrusive and conflicted with our goal of having knowledgeable "locals" being the primary data generators. An alternative policy was to prompt with frequency inversely proportional to coverage: as more spaces in a building are associated with fingerprints, user prompting decreases. However, we found via simulation that this approach resulted in a high false prompting rate, i.e., users were too often prompted in spaces that did not require it. A third policy – inserting interpolated, artificial fingerprints for unbound spaces – requires good coverage of nearby spaces to produce meaningful results. We therefore arrived at a user prompting policy based on spatial uncertainty.

In the next section we outline our approach for evaluating spatial uncertainty. A corresponding policy for user prompting is described in Section 3.1.2.

3.1.1 Spatial Uncertainty

In an organic localization system, if a user is in an unbound space, the most likely space to be selected by the localizer is the nearest bound space. This is because the RF fingerprint of the unbound space is likely to be similar to those of physically nearby spaces (Fig. 3-1). If the localizer determines the user to be in a space with unbound neighbors, the user's true location may be one of the surrounding spaces and *not* the bound space. To convey this spatial uncertainty, we can display to the user the set of possible spaces, including unbound ones. We can also prompt the user to bind in one of the unbound neighbors, as s/he is likely to be nearby.

In order to estimate uncertainty, we employ discrete Voronoi diagrams. In a standard, continuous two-dimensional Voronoi diagram [75], a set of point *sites* exists on a plane. Each site is associated with a *Voronoi cell* containing all points closer to that site than to any other site.

In our solution, bound *spaces* are sites, and unbound spaces become members of the cell associated with the nearest bound space (Fig. 3-2). As a space shifts from being unbound

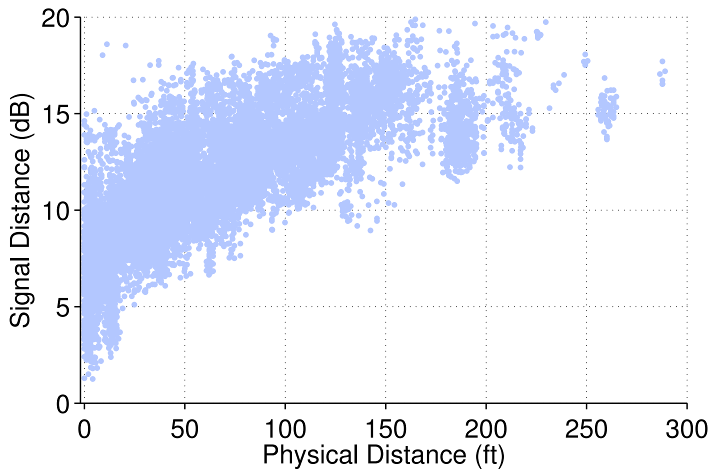


Figure 3-1: Signal distance vs. physical distance. As users add binds from more physically distant spaces, signal distance (Eq. 3.2) between binds increases.

to bound, it becomes a site and adds nearby unbound spaces to its newly-formed cell. The underlying intuition is that if a user is in an unbound space, the space most likely to be selected by the localizer will be the nearest bound space – the Voronoi "site" associated with the user's true location. Therefore, the size of the bound space's Voronoi cell naturally captures the spatial uncertainty associated with prediction of the bound space.

More formally, let L denote the set of all locations in a given floor, and B be the set of bound locations. Let L_c and B_c be sets of centroid coordinates of L and B, respectively. The *Voronoi diagram* for B_c is a planar subdivision of $\mathbb{R}^2$ in which every point x in the plane is assigned to $p \in B_c$ if $d(x,p) \leq d(x,p') \;\; \forall p' \in B_c,\; p' \neq p$. The set of points that are assigned to p is denoted as $V(p)$, the Voronoi cell of p.

For every bound location $b \in B$ with centroid p, we define a *spatial uncertainty region* $\mathcal{U}(b)$ to be a subset of L, as follows: every location $l \in L$ is assigned to one of the uncertainty regions, $\mathcal{U}(b)$, if the Euclidean distance from its centroid l_c is smaller to p than to any other $p' \in B_c$; equivalently, l_c belongs to the Voronoi region of p, $l_c \in V(p)$. In essence, we maintain a generalized Voronoi diagram as a collection of mutually disjoint regions.

For each region $\mathcal{U}(b)$ for a bound space b and its centroid p, we define two spatial uncertainty metrics: the *number of unbound locations*, $n(b)$, and the *maximum uncertainty radius* $r(b)$ defined as:

$$r(b) = \max_{l_c \in V(p)} d(l_c, p) \tag{3.1}$$

which is the maximum distance from the Voronoi site to the farthest unbound location in $\mathcal{U}(b)$. The number of unbound locations is used for the user prompting algorithm (Algorithm 1), giving the spatial uncertainty metric. The maximum uncertainty radius $r(b)$, used when drawing a circle centered on the corresponding bound space b, conveys

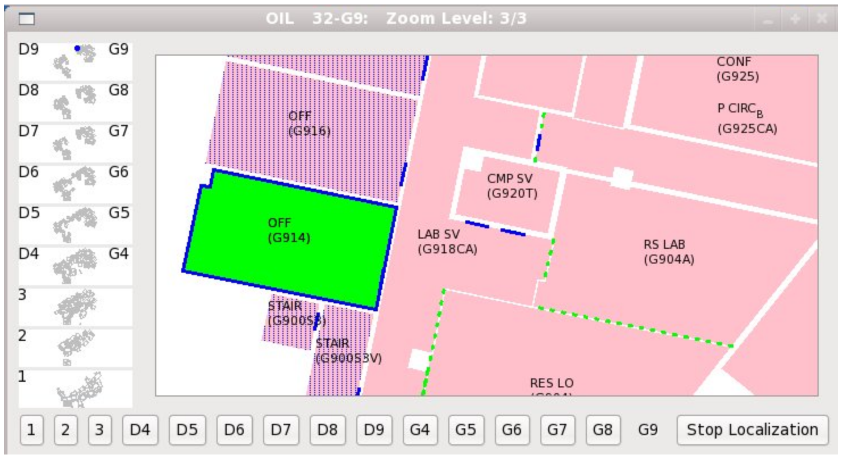


Figure 3-2: Spatial uncertainty. The user sees the currently selected bound space (green) as the Voronoi site, along with spaces belonging to its Voronoi cell (stippled). This helps contributing users know which binds would improve coverage and/or accuracy, and helps non-contributing users understand localization precision

uncertainty to the user.

As noted above, when a space changes from being unbound to bound, the floor's Voronoi diagram must be updated. The update operation is efficient; it is linear in the number of spaces held by any cells adjacent to the newly bound space. This update is performed on the server, then propagated to all clients.

3.1.2 User Prompting Algorithm

OIL requests user input in order to improve either coverage or accuracy. To this end, each client monitors a pair of hypotheses in determining whether further user input will improve the fingerprint database. Specifically, each time a location estimate is produced, the device evaluates the following questions:

1. If the user binds a nearby location, will the system's *coverage* increase?

2. If the user binds his/her current location, will the system's *accuracy* increase for this location?

The first question is answered by considering the spatial uncertainty of the current location estimate $n(b)$. High spatial uncertainty means that many nearby locations remain unbound; thus adding user input for nearby spaces will enhance the overall coverage of the fingerprint database. If the spatial uncertainty metric exceeds a threshold, the user is prompted for input.

Algorithm 1 User prompting algorithm. C_s^{max} and C_i^{max} are thresholds to determine (in)stability, and n^* is a predefined spatial uncertainty threshold. Note that prompting based on high spatial uncertainty occurs only when the location estimate is stable.

```
1: Input: location estimate $l$, uncertainty region $\mathcal{U}(l)$
2: Output: prompt = { true, false }
3: States: stability counter $C_s$, instability counter $C_i$, previous location estimate $l_p$
4: Initialization: $C_s \leftarrow 0$, $C_i \leftarrow 0$, $l_p \leftarrow$ Nil
5:
6: if $l_p$ = Nil then
7:     $l_p \leftarrow l$, prompt ← false, return
8: else
9:     if $l_p = l$ then
10:        $C_s \leftarrow C_s + 1$, $C_i \leftarrow \max\{C_i - 1, 0\}$
11:    else
12:        $C_i \leftarrow C_i + 1$, $C_s \leftarrow \max\{C_s - 1, 0\}$
13:    end if
14:    if $C_s > C_s^{max}$ and $n(l) > n^*$ then
15:        prompt ← true, $C_s \leftarrow 0$, $C_i \leftarrow 0$
16:    else if $C_i > C_i^{max}$ then
17:        prompt ← true, $C_s \leftarrow 0$, $C_i \leftarrow 0$
18:    else
19:        prompt ← false, $l_p \leftarrow l$
20:    end if
21: end if
22: return
```

The second question, concerning accuracy, is answered by checking whether recent location estimates for the user's current location have been stable. Because the duration of each user contribution can be short and wireless signal strength can vary rapidly, a user in a space with a sparse fingerprint might experience unstable and inaccurate localization results. The user is also prompted in this case.

Algorithm 1 shows the method used in OIL to answer these two questions. If the decision is to prompt the user, the user can see local coverage rates on the UI (Fig. 3-2), and (a) decide whether to bind in the current space, (b) bind in an adjacent space, or (c) request not to be bothered again for a short or long duration (5 minutes and 4 hours respectively in our current implementation).

3.2 Filtering Erroneous User Input

An organic localization system is expected to encounter some level of noisy user contributions. In particular, users will not always indicate the correct room when they are prompted to make a "bind." Early tests of our system showed that both ordinary and skilled users did indeed make mistakes.

Across organic location systems, mistaken contributions fall roughly into three categories: (1) when selecting the location from a map – as in OIL – the user may select the wrong room or floor; (2) when entering a user-defined space name – as in RedPin [20], for example – the user may type an incorrect or atypical name; and (3) while making a long interval bind, a user may move, polluting a bind with scans acquired in distinct spaces (see § 2.3). Identifying erroneous contributions is a key problem in organic localization because, without high-quality binds, database and positioning accuracy will suffer.

While we focus on the first type of error in our algorithm description and evaluation, variations on our method would also identify the other two types of errors. While we believe our method would also filter out uncoordinated malicious input, combating a pervasive attack – with many spoofed APs [76], for example – is beyond the scope of this thesis.

Since location fingerprints are generated organically, there is no *a priori* model available for use in identifying correct binds. Error detection should therefore be managed in an unsupervised fashion. Our approach for handling erroneous user inputs hinges on outlier detection in *signal* space – the observed RF signal strengths for each access point (Fig. 3-3). We rely on the fact that independent correct binds made at the same location are similar, and

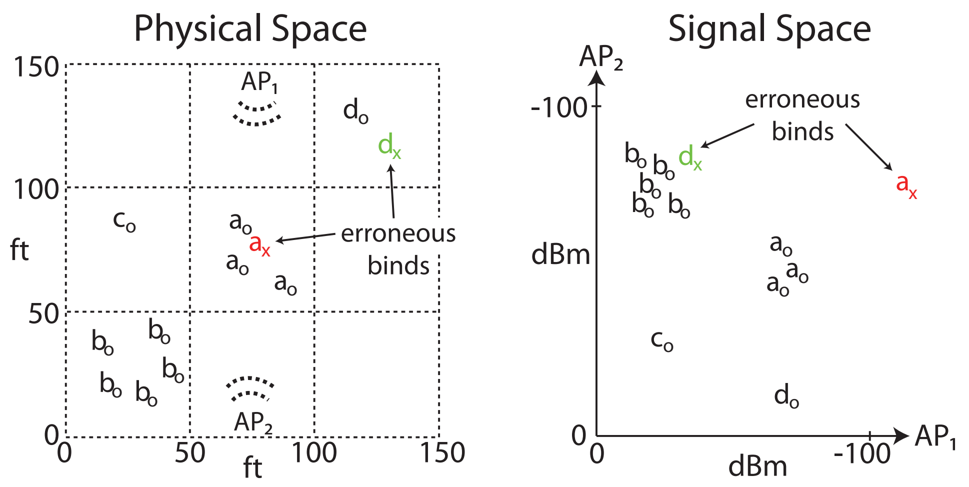


Figure 3-3: Correct binds made in the same physical space tend to cluster in signal space. By observing outliers in the signal space, we can detect and eliminate erroneous binds. Correct binds are denoted as e.g. a_o; incorrect binds are denoted as e.g. a_x.

tend to cluster. We apply a clustering algorithm to detect outlier binds, which are suspected to be erroneous.

When a new user bind is received, it is processed in two steps. First, a clustering algorithm identifies, for the annotated location, a group of binds that are similar in signal space. Then, our proposed erroneous bind detection method tags the new bind as correct or erroneous. Later, localization incorporates only those binds tagged as correct. The fingerprint for each location, however, maintains all binds assigned to it (regardless of correctness), so that all data can be used to periodically reclassify clusters and outliers for that location. Sections 3.2.1 and 3.2.2 describe our approach in detail.

While we focus on erroneous bind detection in this section, we anticipate using a similar approach to address other problems in organic localization, such as detecting AP addition, deletion, and movement or replacement. Currently, we consider a collection of binds from a limited time window, to detect outliers within that window. The same approach can be employed *across* time windows of different granularity to detect true changes in the RF environment, such as AP movement. If clusters of correct binds formed in consecutive windows for the same location vary substantially, this is indicative of a change in the environment. In this case, localization accuracy may be increased by discarding old binds. We leave exploration of this issue to future work.

3.2.1 Erroneous Bind Detection

We represent a *bind* as a signal-strength vector in a k-dimensional signal space, with k the number of observed APs. Given multiple scans per bind, the bind's i-th dimension is populated with the mean RSSI per AP_i. APs for which no signal is observed in the input scans (due to range or channel collision) are assigned a fixed value of -100 dBm.

Given multiple binds made at location l, our goal is to arrange these binds into meaningful clusters. We apply agglomerative hierarchical clustering [77] to group binds by similarity. In this approach, clusters are successively merged in a bottom-up fashion, based on a similarity metric, until no clusters are similar enough to be merged. We define the distance (dissimilarity) metric between two bind vectors $\mathbf{b^s} = (b_1^s, ..., b_k^s)$ and $\mathbf{b^t} = (b_1^t, ..., b_k^t)$ as the normalized signal-space Euclidean distance:

$$d_s(\mathbf{b^s}, \mathbf{b^t}) = \left[\frac{1}{M} \sum_{i=1}^{k} (b_i^s - b_i^t)^2 \right]^{1/2} \tag{3.2}$$

where $M \leq k$ is the number of APs for which a signal has been detected in either bind $\mathbf{b^s}$ or $\mathbf{b^t}$. The normalization term yields proper “per-AP” distances because, in any real setting, each bind will involve only a small subset of the observed APs.

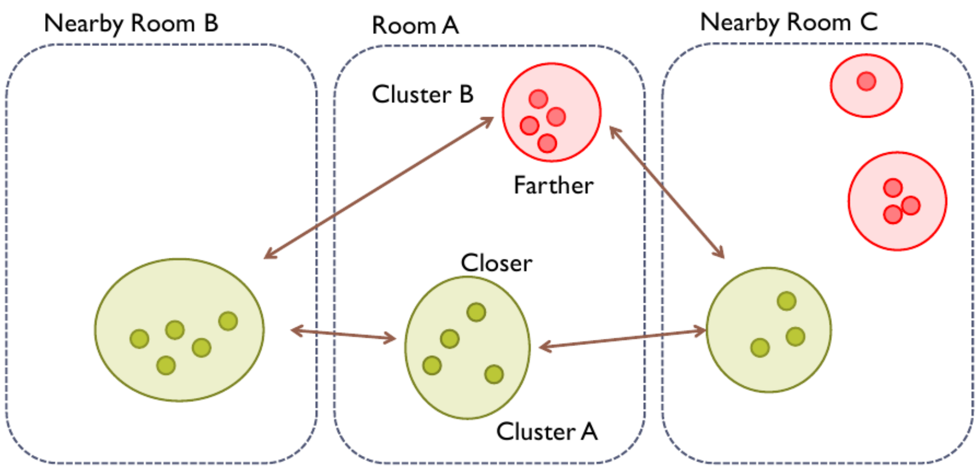


Figure 3-4: Filtering erroneous user input. User binds in Room A are first grouped into two clusters by hierarchical clustering. The correct group of binds is found by comparing both clusters with those in nearby locations.

Further, the distance between two clusters C_s and C_t, referred to as the *linkage function*, is defined as the average distance between inter-cluster bind pairs as follows:

$$D_S(C_s, C_t) = \frac{1}{|C_s||C_t|} \sum_{\substack{(\mathbf{b}^s,\mathbf{b}^t)\in \\ (C_s,C_t)}} d_s(\mathbf{b}^s, \mathbf{b}^t) \tag{3.3}$$

Clustering continues until the linkage function between all pairs of clusters falls below a pre-defined cut-off distance d^*. We use independent labeled data to obtain *a priori* knowledge about intra- and inter-location signal distance to set d^*. This procedure is described in detail in Section 3.2.2.

Once binds are grouped into clusters, the system must identify which cluster includes the correct binds (the rest of the clusters are assumed to contain erroneous binds). If we assume that most users make correct binds, it is natural to take the largest cluster as the correct one. However, in a previous deployment we found that more than 75% of the spaces had three or fewer associated binds [14]. When the organic system has not yet obtained good coverage, majority voting is not feasible. Instead, we use the observation that signal distance between two locations is positively correlated with physical distance between them (Fig. 3-4). Therefore, we identify the correct cluster of binds C_l^* given a set of bind clusters at location l, $\mathscr{C}_l$, according to the following criterion:

$$C_l^* = \underset{C\in\mathscr{C}_l}{\operatorname{argmin}} \sum_{m\in\mathcal{N}(l)} D_s(C, C_m^*) \tag{3.4}$$

Algorithm 2 Erroneous bind detection method.

```
1: Input: new bind $b^N$ to location $l$; set $\mathscr{B}_l$ of all binds for location $l$; neighbor locations $\mathscr{N}_l$
2: Output: set of correct binds $C_l^*$
3:
4: Add $b^N$ to $\mathscr{B}_l$.
5: if $|\mathscr{B}_l| = 1$ or $\mathscr{N}_l = \emptyset$ then
6:     $C_l^* \leftarrow \mathscr{B}_l$ {No information for detection is available.}
7:     return
8: else
9:     $\mathscr{C}_l \leftarrow$ Hierarchical clustering of $\mathscr{B}_l$ (Eq. 3.2 and Eq. 3.3).
10:    if $|\mathscr{C}_l| > 1$ then
11:        $C_l^* \leftarrow$ Identification of the correct cluster (Eq. 3.4).
12:    else
13:        $C_l^* \leftarrow \mathscr{C}_l$
14:    end if
15: end if
```

where $\mathscr{N}(l)$ is the set of locations neighboring location l, and C_m^* is the cluster of correct binds at the neighboring location m at the time of computation. Algorithm 2 outlines our approach.

3.2.2 Clustering Threshold Tuning

The quality of the clusters formed dictates the performance of erroneous bind detection. This section describes how the linkage function threshold d^* is determined.

During clustering, cluster pairs for which the value of the linkage function is larger than d^* are kept distinct. Thus, an optimal threshold is one that is effective at separating binds associated with different locations. If comprehensive labeled data is available, then the cut-off threshold can be set empirically, e.g. by cross-validation, to maximize the localizer performance. In practice, it is hard to obtain such large labeled datasets (indeed, this is the primary reason for pursuing organic localization). Instead, we assume that a handful of scans are available from each of a variety of locations. This data can be collected by the system designers or extracted from early user input. Using this data, we can cast the threshold tuning problem as a Bayesian decision problem, testing whether or not two binds originate from the same location.

Formally, suppose there are two binds b^s and b^t. We wish to evaluate the hypotheses:

$$\begin{aligned} \mathscr{H}_0 &: b^s \text{ and } b^t \text{ originate from the same location.} \\ \mathscr{H}_1 &: b^s \text{ and } b^t \text{ originate at different locations.} \end{aligned} \tag{3.5}$$

Let d denote the distance defined in Equation 3.2. In clustering, $d < d^*$ implies that $\mathscr{H}_0$ holds; otherwise it is estimated that $\mathscr{H}_0$ is false (i.e., that $\mathscr{H}_1$ is true). We assume the cost of false positives is the same as false negatives. Then, according to the Bayesian decision rule, $\mathscr{H}_0$ is accepted if

$$P(\mathscr{H}_0|d) > P(\mathscr{H}_1|d) \iff \frac{P(d|\mathscr{H}_0)P(\mathscr{H}_0)}{P(d|\mathscr{H}_1)P(\mathscr{H}_1)} > 1. \tag{3.6}$$

That is, we will select $\mathscr{H}_0$ and judge that b^s and b^t are from the same location if the ratio $P(\mathscr{H}_0|d)/P(\mathscr{H}_1|d) > 1$ (Fig. 3-5). A system designer can use the small amount of bind data to estimate the distributions $P(d|\mathscr{H}_0)$ and $P(d|\mathscr{H}_1)$. The optimal threshold, d^*, is the point at which the two *posterior* probability distributions cross. To estimate the posterior distribution, one must estimate the prior probabilities $P(\mathscr{H}_0)$ and $P(\mathscr{H}_1)$ (Eq. 3.6).

We tested two different ways of modeling the hypothesis prior. One possibility is to consider the hypotheses equally likely; this leads to a Neyman-Pearson-type likelihood-ratio test, $P(d|\mathscr{H}_0)/P(d|\mathscr{H}_1) > 1$. Alternatively, one can model the assumption that most binds associated with any location l are correct; in other words, hypothesis $\mathscr{H}_0$ is more likely in an operative system. For example, if we assume that 90% of binds are correct and that the erroneous binds are not mutually correlated, then $P(\mathscr{H}_0) \approx 0.9^2$ and $P(\mathscr{H}_1) \approx 1 - 0.9^2$. When we tested our data with each of these two assumptions, we obtained $d^* = 12$ dB and $d^* = 15$ dB, respectively. Each of these values was used as a stopping criterion in the

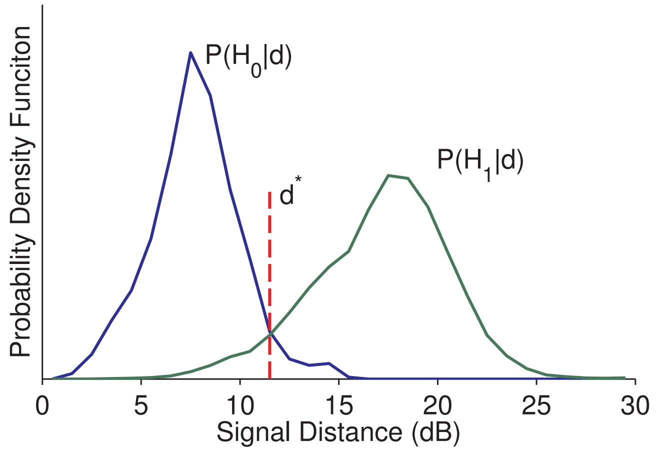


Figure 3-5: Determination of the clustering cut-off threshold. To achieve minimum probability of error, we choose $\mathscr{H}_0$ if $d < d^*$ and $\mathscr{H}_1$ otherwise.

hierarchical clustering algorithm (§ 3.2.1). The first value closely matches Bhasker et al.'s empirical closeness threshold [25].

3.3 Evaluation

In this section, we examine both algorithms in detailed simulation, where we could explore parameter changes easily, and in a live deployment, where we could gather real user input and feedback. Section 3.3.2 examines our claims that Voronoi diagram–based prompting improves coverage rates and helps explain localization precision. Section 3.3.3 evaluates our erroneous bind detector with organic user input.

3.3.1 Test Deployment

We launched a test deployment of OIL, inviting building residents to participate. Nineteen people participated, including two administrators, three people from a non-technical department, and four members of our group. We gave each participant a mobile tablet with the OIL client and building map installed and showed them how to make interval binds and operate the client in general. We asked users to respond to the tablet's prompts when they were able to do so, but not to go out of their way to provide coverage. Users were encouraged to take the tablets with them, including out of the building if they wished.

At the start of the deployment, we also installed fourteen stationary "spot check" tablets in different rooms in the building. We did not make any binds on these tablets, but left them to run and report their location estimates back to the server.

Table 3.1 summarizes the users' contributions as of the end of the study.

Map Spaces	1,373
Contributing Users	19
Bind Intervals (from users)	604
Scans (from devices)	1,142,812
Bound Scans	108,418 (9.4%)
Spaces with Bound Scans	116 (8.4%)

Table 3.1: Statistics for our 9-day test deployment.

3.3.2 Voronoi-based User Prompting

During initial planning for our deployment, we had several ideas for improving coverage rates. We evaluated them via simulation (§ 3.3.2), selected the most promising for deployment, then interviewed participants for feedback (§ 3.3.2).

Improving Coverage Rates

Our basic proposals for requesting user input included *periodic prompting*, where users are prompted at regular intervals, and *inverse coverage*, where prompting rates decline as more spaces on a floor are bound. These methods are simple, but do not direct users to where user input is actually needed. Instead, the dynamic Voronoi diagram approach captures the fact that a user is likely to be located in, or near, a space that requires more coverage. Our opinions were split on how we should expect contributors to act: were users active (willing to move to an adjacent space), or passive (willing to provide input only where they were when prompted)?

We compared four proposals via a simple simulation. We moved users randomly across an artificial 100 × 100 grid floor. To test the performance of prompting methods independently of a localization algorithm, we assume perfect accuracy. We also assume that users always bind when prompted. We let our "active" Voronoi users be willing to move to adjacent locations with probability 1/2 per request.

We examined coverage per number of user prompts, because the purpose of efficient user prompting is to improve coverage while minimizing user effort. Figure 3-6 shows the results of running each method 300 times. Voronoi-based prompting outperforms the other methods, especially if the active user model is assumed. Periodic prompting suffers from prompting too much – irritating users – because it does not consider the current coverage or location estimate. While the inverse coverage method adapts to increasing global coverage,

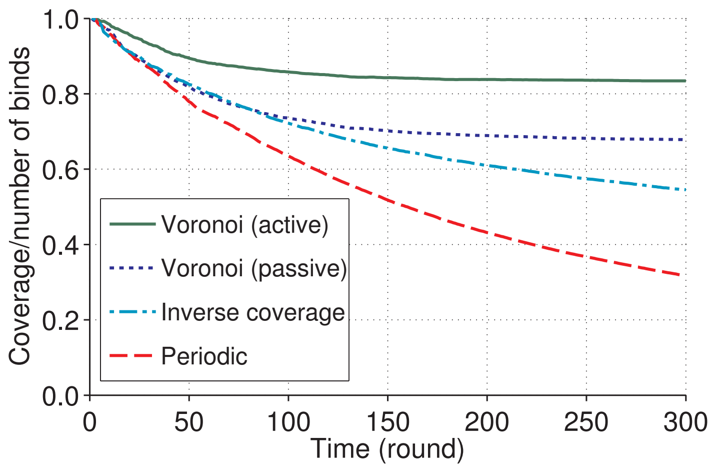


Figure 3-6: Voronoi-based user prompting significantly increases coverage at low user effort compared to other methods in our simulations. Active users, who are willing to move to an adjacent unbound space to contribute a bind, provide the most benefit.

it does not do as well as the Voronoi-based methods. Voronoi-based prompting considers both coverage and local spatial uncertainty, reducing unnecessary user prompts. When users provide binds in nearby locations, as assumed with the active user model, the effectiveness of Voronoi prompting further increases. Our OIL client asked users to bind in adjacent rooms, in effect suggesting to users that they adopt the active user model.

Conveying Spatial Uncertainty

After completing our test deployment, we interviewed participants about the Voronoi prompting mechanism. Overall, the responses were mixed. Of the top two contributors in the experiment (§ 3.3.1), one said the prompts were the main reason that she made so many binds. She also found that the Voronoi regions, as in Figure 3-2, were useful for quickly locating the room that she was in as well as assessing how well the tablet knew her current location. The other top contributor said that the prompting mechanism had no effect on his behavior. One less active user found the prompting irritating as he rarely left his office, had little interest in making binds, and continued to be prompted. Although he could have marked the unbound spaces surrounding his office as inaccessible, he did not do so – but he did turn off prompting.

These observations suggest that while Voronoi prompting can be helpful, it could be made more adaptive and personalized. For users who make few binds or do not bind when prompted, the system could prompt them less, whereas the system could continue to prompt users if it appears to be advantageous to do so.

3.3.3 Erroneous Bind Detection

We studied the performance of our erroneous bind detector. First, we examined the effect of time on the outlier detection. Next, we evaluated the end-to-end effect on accuracy as we varied the fraction of erroneous binds. Both evaluations were performed using simulation on organic data from an earlier OIL deployment, which had 16 users and lasted for 20 days [14].

Effect of Time on Detection

We used a discrete event simulator to see if erroneous binds could be detected after each space's fingerprint contained enough binds to measure a valid signal distance to its physical neighbors. We also varied the clustering threshold (§ 3.2.2) to observe its effect on overall detection. At each round of simulation, a correct bind is taken from the data set. Before being added to the fingerprint database, its location is changed to a random one with probability p_e, emulating an erroneous bind. Then, the bind is added to the fingerprint of the potentially-erroneous space.

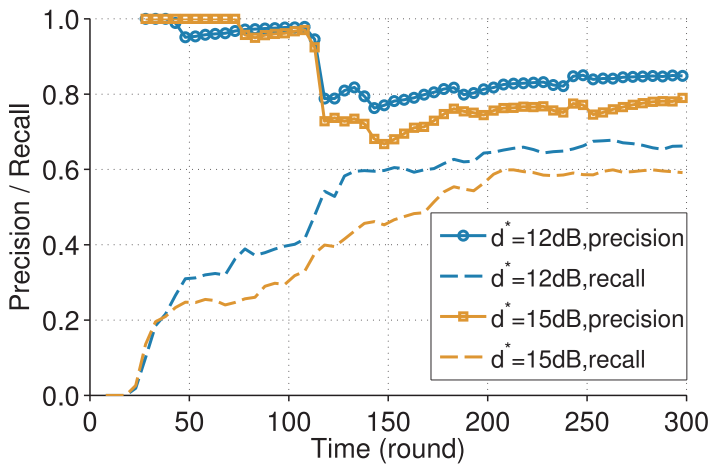


Figure 3-7: After each space's fingerprint acquires a sufficient number of binds, detection success significantly increases. In addition, conservative detection (12 dB) improved both precision and recall in our experiments.

Figure 3-7 shows the *precision* and *recall* for conservative (12 dB) and lenient (15 dB) thresholds, which correspond to physical separations of approximately 50 and 125 feet, respectively. Precision is the fraction of truly erroneous binds detected over all binds. Recall is the fraction of erroneous binds identified over all erroneous binds. We used $p_e = 0.1$ for this test.

The data show that the bind detection algorithm's performance increases as the fingerprint database becomes more populated. In detecting erroneous binds for a certain location l, initially the algorithm just accepts binds unconditionally until there is sufficient information to make an estimate. As fingerprints of neighboring spaces become more populated, the algorithm more readily identifies inconsistent binds. In this sense, the fingerprint database is self-repairing. After 300 rounds, the conservative threshold achieved a precision of 0.85 and recall of 0.68.

Localization Accuracy

We next examined the effect that different levels of error-proneness among contributors would have on overall accuracy. To do so, we varied the fraction of erroneous binds presented to the system, from zero to one, and measured localization accuracy. Figure 3-8 shows localization performance in three cases: an "oracle" (perfect) detector, our clustering detector, and no detection. The data demonstrate why filtering out erroneous binds is essential: erroneous user input greatly compromises accuracy. The proposed detector enhances localization accuracy by 5–9% over a wide range of p_e. Unsurprisingly, at high error rates, the algorithm results in low accuracy because the fingerprints of neighbor locations also contain many

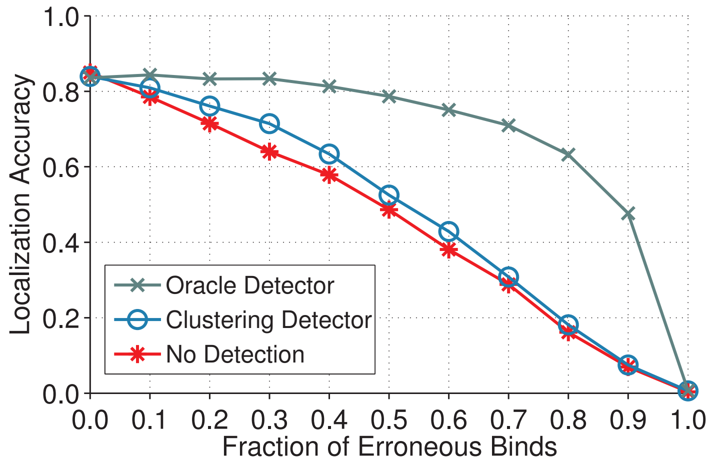


Figure 3-8: Filtering Erroneous Binds. We varied the fraction p_e of erroneous binds given to the system and computed overall system accuracy.

erroneous binds.

3.4 Related Work

We review related work on Voronoi diagrams and clustering in the context of localization.

3.4.1 Voronoi Diagrams

The Voronoi diagram is one of the fundamental geometric structures in computational geometry and has been used in many other different fields including computer graphics, robotics, physics, and sensor networks [75]. In the context of indoor positioning, Swangmuang and Krishnamurthy used closely related proximity graphs – Delaunay triangulation, Gabriel graphs, and relative neighborhood graphs – to obtain an analytical model for the localization error probability of a given fingerprint [78]. In contrast, we use the Voronoi diagram to approximate the spatial uncertainty that naturally arises from organic user contributions.

3.4.2 Robustness and Clustering

As localization using RF infrastructure has become widespread, researchers recently investigated its susceptibility to spoofing attacks. Chen et al. examined the robustness of several localization algorithms against signal strength attenuation attacks [79]. Tippenhauer et al. [76] studied various types of attacks including AP impersonation and spoofing as well as injection and corruption of the fingerprint database. Although our focus is on user input errors, malicious attacks are closely related because, in both cases, incorrect wireless signals can be entered into the fingerprint database.

Cluster analysis has been widely used for anomaly detection. For example, Portnoy et al. use clustering to detect anomalies in network traffic [80]. Clustering has been used for several purposes in localization systems: for example, Swangmuang and Krishnamurthy use it to improve performance prediction [81] and Lemelson et al. use clustering as a measure for error prediction [82]. To our knowledge, clustering has not previously been used to detect erroneous user input to localization systems.

3.5 Conclusion

While the concept of organically constructing a localization system is simple, building a working system in practice presents significant challenges. This chapter addresses two issues that arise in "growing" an organic indoor location system: modeling uncertainty, and handling erroneous user input.

We proposed a method, based on Voronoi diagrams, that suggests to active contributors what spaces around them need coverage, and conveys to all users the level of localization precision they can expect in their current vicinity. We also described a method that watches for erroneous user contributions and automatically discounts them. This method, based on outlier detection through clustering, allows an organic positioning system to maintain its accuracy over time. We examined the proposed methods in simulation and through a test deployment.

In the future, we plan to continue examining the role of time in organic indoor localization. In addition to discerning erroneous binds, our bind clustering method appears generalizable to other problems in organic localization. For example, we anticipate using it to detect addition, deletion, and movement of APs. Another interesting topic would be to investigate combining contributions from both trusted and untrusted surveyors. Building on the ActiveCampus approach [21], less trusted, organic refinements could complement an initial, trusted survey of mostly public spaces.

Chapter 4

Device Diversity in Organic Indoor Localization

Another key challenge to organic localization that we examine in this chapter is the device diversity problem. Variations in RF characteristics between different mobile devices pose a challenge in designing localization algorithms that can inter-operate across diverse types of devices. In this chapter, we study this problem for two representative RF characteristics: signal strength and AP (access point) detection rate. We design three different Bayesian localization models — signal strength–based, AP detection–based, and hybrid feature models — and analyze how device diversity affects localization performance under various localization algorithm designs.

4.1 Device Heterogeneity Problem

While conventional survey/use localization relies on a degree of uniformity among users' devices, in which surveys are conducted with one or a small set of well-understood devices, users in organic localization systems employ diverse types of 802.11 devices for the generation of fingerprints. This raises the "device heterogeneity" problem, which fingerprints from different users may be incompatible.

Two basic measurements available from WiFi scanning are *signal strength* and *access point detection*. Most WiFi-based localization methods utilize either or both features to construct a unique fingerprint for each location. The key assumption required for RF-fingerprints to be used in localization is that they are unique per location (i.e. different locations have different fingerprints), and that they do not vary due to other factors, in particular, the type of the device that measures RF signal.

In practice, however, various factors do affect RF measurements by a mobile device, including environmental factors (fading, transient occlusion, etc.), hardware (antenna sensitivity, decoding algorithm, etc.), and software factors (device driver implementation). Moreover, 802.11 specifications do not specify in detail how WiFi scanning must be performed, leaving some freedom on the actual RF measurements (e.g. signal strength value) up to device manufacturers.

Previous work has acknowledged the problem of sharing fingerprints between heterogeneous devices and suggested that it could be solved by a simple linear transformation between received signal strength indicator (RSSI) values [11]. We demonstrate that linear transformation of signal strengths is insufficient for cross-device localization. Instead, we find that a wide kernel applied to the signal strength distribution provides significantly better end-to-end accuracy than linear transformation, because the former captures the primary difference across devices: signal strength dispersion. A second complication afflicts fingerprint sharing: the set of access points detected by each device can be different. We show experimentally that the number and identity of APs detected by each device can vary widely even in nearby locations. As a consequence, a common alternative to RSSI-based localization — relying on access point presence or absence — fails when this type of fingerprint is shared across heterogeneous device types. We show through an information-theory argument that augmenting RSSI-based localization methods with presence/absence will actually degrade performance.

4.2 Localization Models

In the Bayesian localization method in Equation (2.3), the MAP estimate depends on the class-conditional probability $p_{O|L}(o|l)$, which is defined in terms of features arising from WiFi measurement characteristics. The most common feature used for WiFi localization is signal strength, which gives the localization algorithm in Equation (2.4). However, other features such as AP presence/absence can be used for localization as well. This section describes alternative models for the class-conditional probability for a choice of different features. For all models, we assume that each feature is conditionally independent of every other feature given a location, yielding the naïve Bayes classifier.

4.2.1 Signal Strength

Section 2.3.2 presented signal strength–based Bayesian localization. The class-conditional probability $p_{S_i|L}(\cdot|\cdot)$ can be estimated from training data in different ways, by modeling it as a categorical distribution (histogram), a Gaussian distribution (with maximum-likelihood parameter estimates), or a kernel density estimator (Parzen window estimator). The latter method accounts for the variance of each sample.

The kernel density estimator $\hat{p}_X^k(\cdot)$ [83] estimates the probability density function $p_X(\cdot)$ as:

$$\hat{p}_X^k(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right) \tag{4.1}$$

where x_i is an observed sample of random variable X, h is a kernel width, and $K(\cdot)$ is a kernel function. A Gaussian kernel is often used for the kernel function. The kernel width determines the degree of sample "smoothing" effected by the kernel.

The estimated probability density function is discretized and regularized using an m-estimate as described in Section 2.3.2.

4.2.2 Access Point Detection

Another feature vector can be constructed to reflect the presence or absence of access points. If we model the presence/absence of the signal from a certain AP as a Bernoulli process, the observation follows a multivariate Bernoulli model in which $O = (J_1, J_2, \ldots, J_k)$ for k APs, where J_i is a binary variable with value 1 indicating presence of signal from AP i, and value 0 indicating its absence. In this framework, the decision rule (Eq. 2.3) becomes:

$$\hat{l} = \underset{l \in L}{\operatorname{argmax}} \left[\prod_{1 \le i \le k} \left\{ p_{J_i|L}(1|l) \right\}^{J_i} \left\{ 1 - p_{J_i|L}(1|l) \right\}^{1-J_i} \right] \tag{4.2}$$

where $p_{J_i|L}(1|l)$ is the probability that AP i is detected in a WiFi measurement at location l.

This localization algorithm requires only presence/absence information of access points, which can be easily obtained from any WiFi device, and from which constructed fingerprints are compact compared to those of signal strength–based localization. Because of these merits, presence/absence information is particularly well-suited to large-scale, coarse-grained localization.

However, this formulation explicitly considers negative evidence — absence — of a signal from a certain access point; we show later that this becomes problematic when different devices detect partially disjoint sets of APs.

4.2.3 Hybrid: Signal Strength and AP Detection

The detection probability $p_{J_i|L}(1|l)$ and the signal strength can be used together as feature variables. In this formulation, the observation variable becomes $O = ((J_1, S_1), (J_2, S_2), \ldots, (J_k, S_k))$. The signal strength variable S_i is conditioned on the detection variable J_i, and must be marginalized if it is not observed (i.e., if $J_i = 0$). We derive the following classification rule from Equation (2.3) as follows:

$$\hat{l} = \underset{l \in L}{\operatorname{argmax}} \left[\prod_{1 \le i \le k} \left\{ p_{J_i|L}(1|l)\; p_{S_i|J_i,L}(s_i|1,l) \right\}^{J_i} \left\{ 1 - p_{J_i|L}(1|l) \right\}^{\alpha(1-J_i)} \right] \tag{4.3}$$

where α, $0 \le \alpha \le 1$, is a discounting factor which determines how much to discount negative evidence. This prevents localization from being dominated by negative evidence if many access points are present but each WiFi scan captures only a small fraction of them. When $\alpha = 1$, the formula is identical to the one used in [84]; when $\alpha = 0$, it is equivalent to [85]. We set $\alpha = 1$ for our experiments.

4.3 Experimental Setup

In order to examine the effect of device diversity on indoor positioning, we collected WiFi scans from six different devices at 18 locations in one building. We used two different commodity laptops, a netbook, a mobile phone, and two tablet computers. The tablets were the same model, illustrating the homogeneous organic localization case. Table 4.1 summarizes the devices we compared as part of the experiment.

The six devices were placed on a rolling cart, enabling simultaneous data collection, with each device logging to local storage to avoid using their radios for data transmission while data collection was in progress. The device radios performed no activity other than scanning.

In each location, each device recorded WiFi scans for seven minutes. Scans were taken near each device's peak rate, with a one-second gap between scans. Because of the variation

Device	WiFi Chipset	OS	Kernel
Clevo D901C laptop	Intel 5300AGN (802.11a/b/g/n)	Linux Ubuntu 10.04	Linux 2.6.32
Asus EEE900A netbook	Atheros AR5001 (802.11b/g)	Linux Ubuntu 10.04	Linux 2.6.32
Lenovo Thinkpad X61 laptop	Intel 4965AGN (802.11a/b/g/n)	Linux Ubuntu 10.04	Linux 2.6.32
Nokia N810 tablet (x2)	Conexant CX3110X (802.11b/g)	Maemo OS2008	Linux 2.6.21
Nokia N95 cellphone	TI OMAP2420 (802.11b/g)	Symbian S60 FP1	EKA2

Table 4.1: Devices used for data collection.

in the time each device takes to complete a scan, this resulted in a maximum of 552 and a minimum of 61 scans collected at any location; the EEE900A laptop, for example, often took seven seconds to complete a scan request. While this difference would affect time-to-update performance for moving users, our analysis ignores this factor because it is not relevant for instantaneous localization. To remove the effect of this factor from our results, in each experiment we selected 60 scans at random per device from each seven-minute period. (See `http://rvsn.csail.mit.edu/location` for the raw data, and the samples used for each experiment.)

In the following sections, we study the effect of differing WiFi signal strength measurements (§ 4.4) and access point detection patterns on WiFi localization (§ 4.5) where the survey and use devices are different. In particular, we focus on the cross-device localization problem, in which survey data from one device is used to localize another device. In machine learning terminology, the "survey" device becomes the source of *training* data and the "use" device is the *test* device. We use each set of terms interchangeably.

4.4 Heterogeneous WiFi Signal Strengths

This section shows that signal strength scans from different devices exhibit not only a linear shift in signal strength but also a difference in dispersion. This suggests that sharing fingerprints between different devices would be more effective with "smoothed" signal strength values, e.g. a wide kernel function. We then show that using a relatively wide kernel to share signatures does indeed lead to a significant improvement in accuracy of location estimation.

4.4.1 Analysis of Pairwise Device Calibration

Previous work suggested that inter-device calibration can be achieved by applying a linear transformation of signal strength values from one device to the other [11, 86], estimated from WiFi scan data taken from both devices at the same time and place.

We compare signal strength measurements of the six devices in our dataset, showing pairwise scatter plots in Figure 4-1. We observe a strong correlation in the *mean value* of signal strength measurements between every pair of devices. Therefore, as suggested by previous work, we first attempted the following simple procedure for cross-device localization.

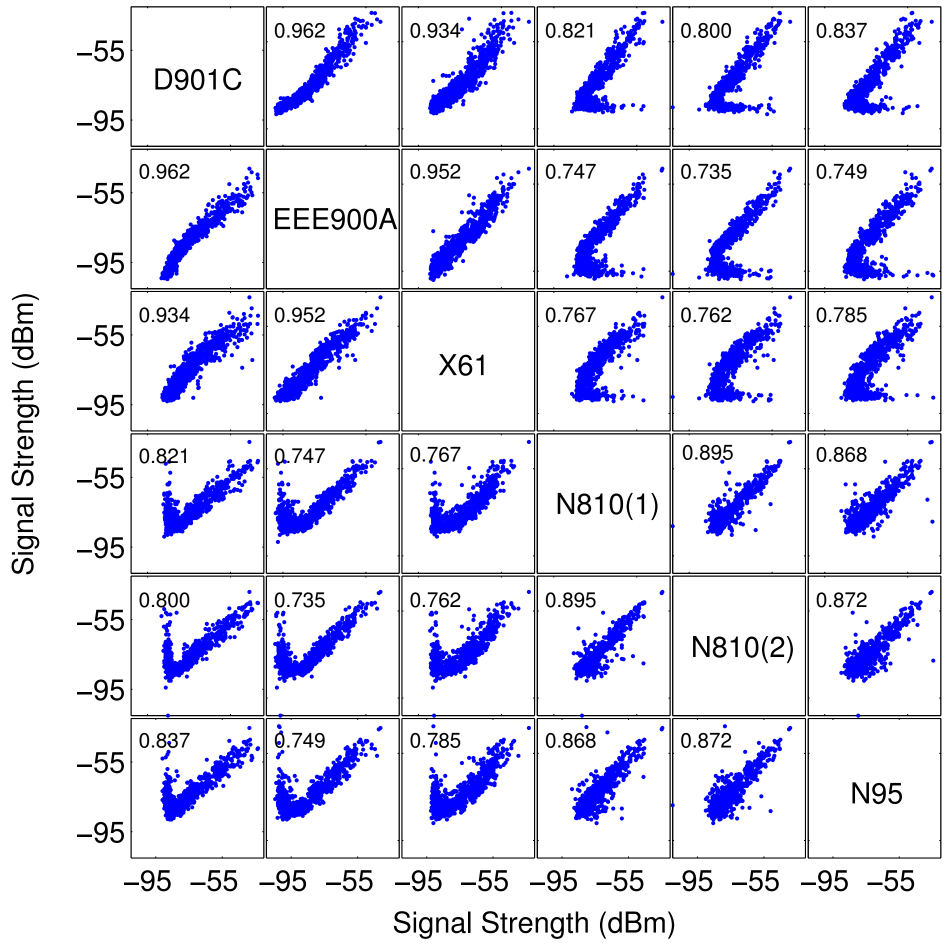


Figure 4-1: Each point in each scatter-plot represents a pair of mean RSSI values from the same access point, observed by a pair of devices placed in the same location for the same time interval. For example, the scatter-plot in row 3, column 4, compares scans of the `N810(1)` with scans of the Thinkpad `X61`. Most pairs show a strong linear correlation, but some devices, e.g. N810s, show noisy values at low signal strength ranges. The Pearson correlation coefficient for each dataset is given in the corresponding scatter-plot.

1. **Pairwise device calibration.** For every pair of devices A and B, the coefficients for linear transformation from device A to device B are computed as:

$$\bar{S}^{B}_{i,l} = \beta^{B}_{A}\bar{S}^{A}_{i,l} + \alpha^{B}_{A}, \tag{4.4}$$

   where $\bar{S}^{A}_{i,l}$ ($\bar{S}^{B}_{i,l}$) denotes the mean signal strength value of device A (resp., device B) for WAP i at location l, and α^{B}_{A} and β^{B}_{A} denote linear coefficients for the transformation from device A to B.

2. **Positioning.** If device A is used for training and B is used for positioning:

   (a) Linear transformation from B to A is applied to test scans of device B.

   (b) Device B is then localized using device A's training data.

The linear transformation is computed by linear least squares, with sample pairs differing by more than 20 dB excluded from fitting.

Figure 4-2 shows the resulting localization error in meters when WiFi measurements from each device are tested against training data from device `N810(2)`. Other combinations of devices showed similar characteristics. For baseline evaluation, we used the Gaussian distribution for class-conditional probabilities. The linear transformation with Gaussian class-conditional probability improved localization performance significantly only for `EEE900A`, but not provide significant improvement for other devices. Among every combination of training and test device, linear transformation improved performance significantly only when the `EEE900A` was used for either the training or test device. For the 10 device pairs including `EEE900A`, the improvements in spot-on accuracy and error distance were 29.8%

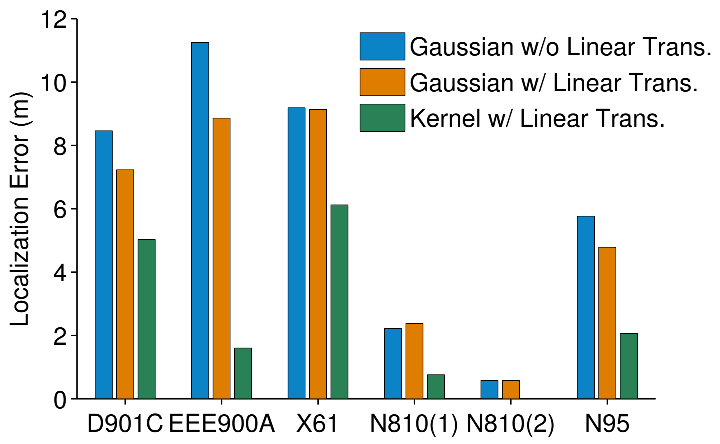


Figure 4-2: Localization accuracy, as mean physical distance away from the correct room, when the `N810(2)` acted as the organic surveyor and the other devices act as users.

and 5.47 meters respectively, while for the other 20 device pairs, improvements were 3.29% and 0.418 meters, which are not significant.

This observation led us to investigate the net effect of linear transformation on actual WiFi measurements of each device. Table 4.2 shows the dynamic range of each device. Only the EEE900A was significantly different from other devices with respect to the signal strength dynamic range. As a result, the linear transformations shift signal strengths only slightly for most devices, except for the EEE900A. However, the mean deviation (regression residual) of signal strength values, excluding outliers, from the linear transformation lines was 3.5 dB. This means that the amount of deviation of each signal strength value is comparable to the amount of global shift by linear transformation, except for EEE900A. Thus linear transformations are ineffective for other devices. Figures 4-3a to 4-3c illustrate the details.

4.4.2 Kernel Density Estimation

This observation implies that the major characteristics of signal strength diversity lie not only in the linear difference between devices, but also in the different local deviation and shape of individual signal strength distributions. While global linear transformation may be able to adjust for large differences in dynamic range, it fails to adjust for local differences that are specific to a certain location and AP. In order to reduce such differences in signal strength distributions across devices, we consider kernel density estimation (Eq. 4.1) in computing individual class-conditional probabilities.

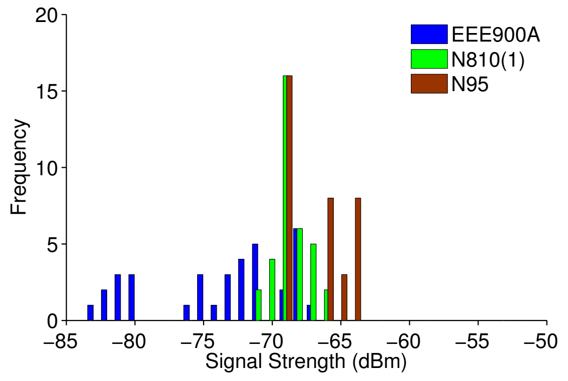
Kernel density estimation takes the noisiness of individual samples into account. Here, we use kernel density estimation to compensate for the difference between signal strength distributions across different devices. We evaluated a Gaussian kernel with widths varying from 1 dBm to 10 dBm. An example of kernel estimate with width 4 dBm is shown in Figure 4-3d. Figure 4-2 also shows enhanced localization accuracy when kernel estimation with the same width is used. (N810(2) acted as the training device.)

Figure 4-4 shows the effect of kernel width on cross-device localization, and compares

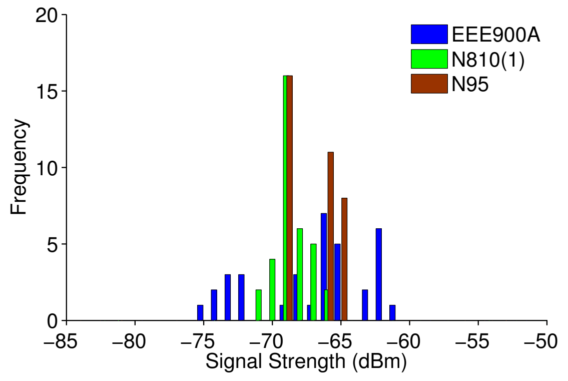
(dBm)

%	D901C	EEE900A	X61	N810(1)	N810(2)	N95
0	-92	-106	-93	-92	-110	-90
25	-86	-98	-87	-81	-83	-81
50	-81	-90	-81	-76	-77	-75
75	-72	-79	-69	-69	-70	-68
100	-25	-41	-29	-35	-39	-35

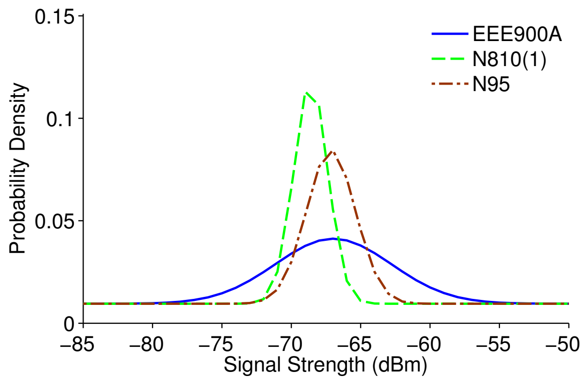
Table 4.2: Dynamic range of each test device (in percentiles).



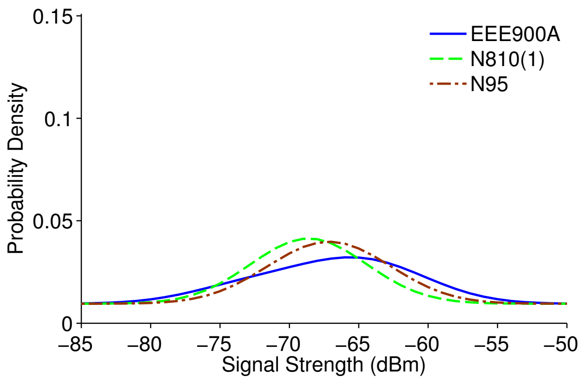
(a) Signal strength histograms before linear transformation



(b) Signal strength histograms after linear transformation



(c) Gaussian estimation



(d) Kernel estimation

Figure 4-3: Kernel density estimation vs. Gaussian estimation. The raw data in (a) show the histogram of RSSI values from a single AP that three devices observed in the same room during the same seven-minute window. The signal strengths for the `EEE900A` are considerably different, while the difference between `N810(1)` and `N95` is smaller. In (b), the linear transformation is effective for `EEE900A`, while its effect is minimal for `N95`. Even after transformation, the dispersion and shape of signal strength values for each device differ significantly. The Gaussian probability estimates (and histogram) for these devices differ significantly (c). This difference adversely affects localization. For example, a signal strength of -64 dBm is observed often for `N95`, but has near-zero probability if Gaussian-fitted training data for `N810(1)` is used. Transformation with a wide kernel significantly reduces the difference between different devices (d).

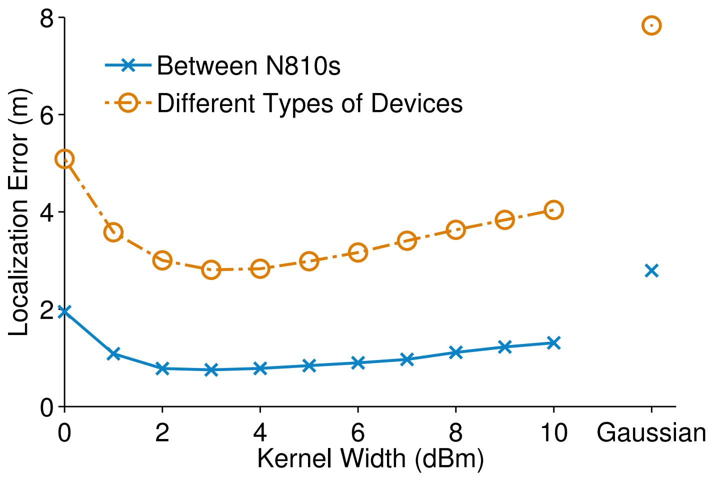


Figure 4-4: Use of kernels significantly improves cross-device localization. Localization performance with varying kernel widths for the same type of devices (N810) and for the heterogeneous pairs of devices (all excluding N810 pairs) are shown. The results with Gaussian density estimation are also provided for comparison. As the kernel compensates for the difference between signal strength distributions across devices, localization accuracy improves significantly. Improvement is greater for different device types than for homogeneous devices. However, if too wide of a kernel is used, localization performance starts to degrade as RSS differences arising from true changes in location are masked.

kernel estimation to the histogram method and Gaussian density estimation. We show localization error between the same type of device (i.e. between `N810(1)` and `N810(2)`) and, separately, the error between different types of devices. A kernel width of 3 dBm provided the best localization performance with our dataset. Not surprisingly, the effect of kernel estimation is more significant for different device types, as their deviation was greater. Neither raw histogram estimation (kernel width $\rightarrow 0$ dBm) nor Gaussian density estimation perform well, particularly for localization across different device types.

The standard deviation computed from the signal strength samples taken from one device for two minutes was approximately 2 dBm. Compared to this value, the best kernel width of 3 – 4 dBm for cross-device localization is somewhat higher than the smaller-scale variation of a specific device type. The reason is that, as each device shows a different dispersion and shape of its signal strength distribution, a strategy of doing more "smoothing" than that required for single device localization is more effective.

4.5 Feature Design of Localization Algorithms for Organic Indoor Localization

This section first analyzes another characteristic of the wireless scan signal: visibility, or *detection* of access points by diverse devices. The detection feature has been used exclusively, or augmented with the signal strength feature in various contexts of RF-based localization [84, 85, 87, 88]. We compare its use to the signal strength feature, and discuss feature design of localization algorithms for heterogeneous devices.

4.5.1 Analysis of Detection Rate for Diverse Devices

In any practical, large-scale localization system based on wireless networks, a mobile device captures only a subset of all access points "visible" at a given location because some access points are only intermittently detected by the mobile device. Factors may be both environmental, such as multipath fading, and transient, such as occlusion by humans or other objects. In addition, the OS or driver may allow only a limited time for collecting scan information, so APs may be missed if the beacon and driver are off-cycle.

A localization algorithm can use the probability of observing an individual access point, or detection rate, to give a different weight for the features it uses, i.e. signal strength likelihood. The rationale behind using AP presence/absence is that more frequently observed access points may be more informative for distinguishing locations. Alternatively, presence/absence can be used as the exclusive factor, providing coarser precision at a lower information cost, as no RSSI values are used; this may be beneficial when the (binary) signal maps for physically large areas are stored on low-memory mobile devices.

However, different devices not only detect WiFi signal strengths differently, as we saw in Section 4.4, but also differ in the sets of APs that they observe. One device may detect a nearby AP consistently, while another may not detect it at all (Figs. 4-5 & 4-6). This occurs because of differences in frequency band (2.4 GHz and/or 5 GHz), radio/antenna sensitivity, firmware/driver implementations, and other factors.

Consequently, localization performance can degrade if the detection probability for each access point is used as a feature for localization. To illustrate this problem, we consider three different types of Bayesian localization algorithms according to the degree of detection information used in the localization process (§ 2.3.2): the localization rule in Equation (2.4) is least dependent on the presence/absence information, while Equation (4.2) uses the detection rate exclusively.

We compare the performance of each Bayesian localization feature choice in Figure 4-7. The algorithm that is exclusively detection-based exhibits the worst performance; the gap between the same type of device (N810) and the different types of devices is also the

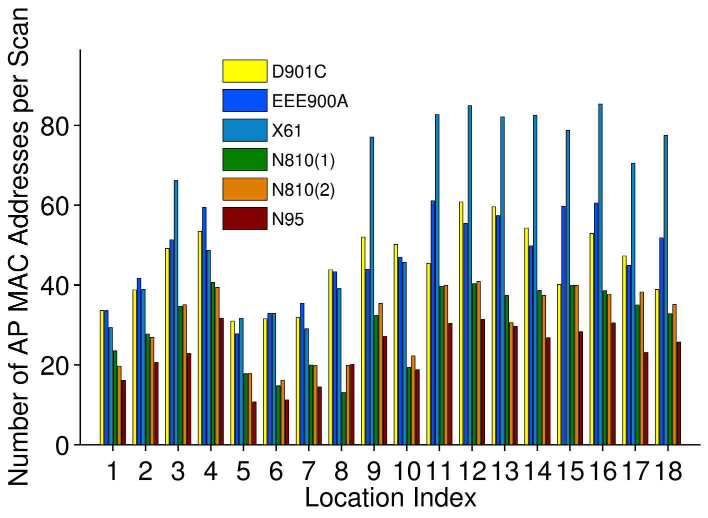


Figure 4-5: Number of distinct AP MAC addresses observed per WiFi scan for each device. The number of APs differs by more than a factor of two. In general, devices with larger form factors (laptops) observe more APs than smaller devices (tablets and cellphones).

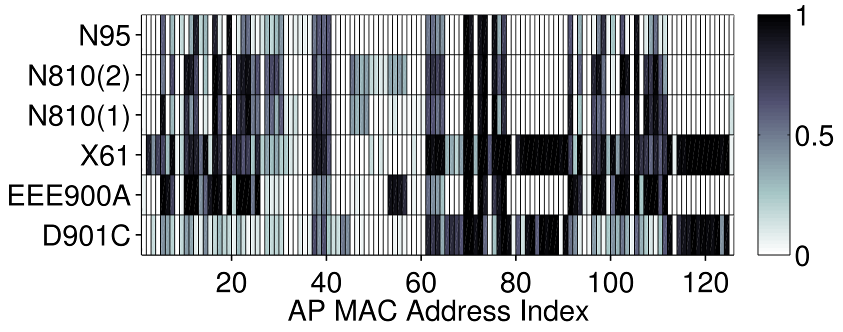


Figure 4-6: Presence/absence pattern across different devices. While the primary difference is due to frequency (i.e. , `D901C` and `X61` operate at both 2.4 GHz and 5 GHz bands while others use only the 2.4 GHz band), there is still considerable difference across same-band devices. This difference reduces the efficacy of localizers that rely on AP presence/absence as a feature.

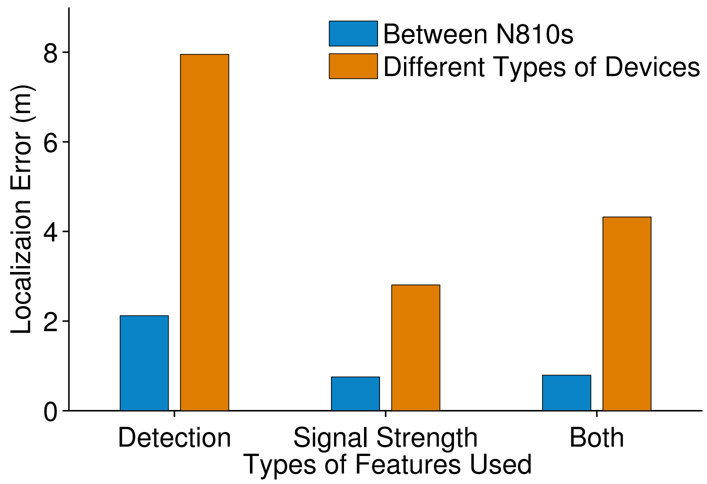


Figure 4-7: Comparison of localization error using three feature types: AP Detection, Signal Strength, and Both. Because different devices observe partially disjoint sets of APs, it is better not to include AP presence/absence in cross-device localization. Degradation in localization error due to use of the detection rate feature is most prominent between different device types.

largest. The signal strength–based algorithm performs best among the three, as it does not rely explicitly on presence/absence information. The localization algorithm that uses both signal strength and presence/absence information shows comparable localization accuracy for same-type-of-device localization, but significantly degraded accuracy for cross-device localization.

4.5.2 Effect of Dissimilarity in AP Detection on Localization

In order to better understand the effect of AP detection on the performance of each localization algorithm, we consider Kullback-Leibler divergence (KLD) [89], an information-theoretic measure that captures the asymmetric dissimilarity between two probability distributions. As in Section 4.2.2, we model the presence/absence of each access point as a Bernoulli process. For access point i, $1 \leq i \leq k$, let J_i^S and J_i^T denote the binary random variables associated with the Bernoulli processes for test device S and training device T respectively. Then, given location l, the Kullback-Leibler divergence of J_i^S over J_i^T for access point i is given by:

$$D_{KL}(J_i^S \| J_i^T; l) = \sum_{x \in \{0,1\}} p_{J_i|L}^S(x|l) \log \frac{p_{J_i|L}^S(x|l)}{p_{J_i|L}^T(x|l)}. \tag{4.5}$$

where $p^S_{J_i|L}(x|l)$ (or $p^T_{J_i|L}(x|l)$) is the probability that access point i is detected by device S (resp., T) in location l for each WiFi measurement.

To compute the overall divergence of device S w.r.t. device T, Equation (4.5) is summed over all k access points and all locations in the data:

$$D_{KL}(S||T) = \sum_{l\in L} \sum_{1\leq i\leq k} \sum_{x\in\{0,1\}} p^S_{J_i|L}(x|l) \log \frac{p^S_{J_i|L}(x|l)}{p^T_{J_i|L}(x|l)}. \tag{4.6}$$

The Kullback-Leibler divergence of S over T can be considered a measure of how much extra information is required to encode the detection process for device S when using the detection process for device T. Thus, it naturally captures the "divergence" of testing device S when training data from device T is used.

We compared KLD from each training-test pair of devices with the location error between them. Figure 4-8 shows the correlation between KLD and the localization error for each localization algorithm. The effect of device difference captured by KLD is more correlated with detection-based and hybrid localization errors, than with errors observed using a signal strength–based localization algorithm.

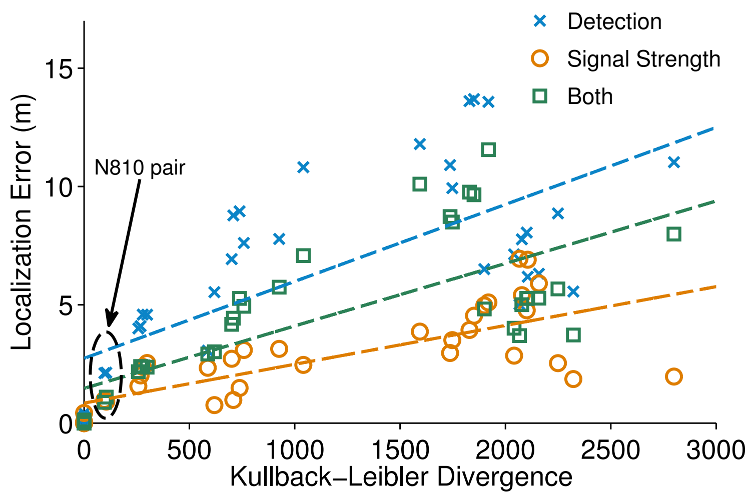


Figure 4-8: Relation between KLD and localization error for three localization algorithms. Each point represents detection dissimilarity captured by KLD and localization error between a pair of devices. When two devices differ more in WAP detection, localization based on detection degrades more significantly. Cross-device localization using devices of the same type (`N810 pair`) does not show such degradation.

4.5.3 Feature Design of Localization Algorithms for Heterogeneous Devices

The previous two sections (§ 4.5.1, § 4.5.2)) showed that AP visibility varies considerably across heterogeneous devices, and that dissimilarity in AP detection adversely affects cross-localization. In general, we found that using only the signal strength feature, without incorporating negative evidence, showed the best localization performance for heterogeneous devices.

As shown in Figure 4-7, the detection rate feature augmented in Equation (4.3) does not give much extra information for distinguishing locations over signal strength, even between same-type devices (the N810s). Similarly, the localization algorithm based solely on the signal strength feature is also affected by dissimilarity of AP detection (§ 4.5.2). This is because the use of m-estimate smoothing applied to the class-conditional probability (§ 2.3.2) for each AP implicitly encodes the detection information of that AP. For example, if there is no observation in the training data for a certain AP in a certain location, but the AP is detected during localization, the corresponding class-conditional probability is initialized as a uniform distribution, which encodes the least amount of information possible from that new observation. Consequently, the signal strength–based localization algorithm will assign a minimal score to that location according to Equation (2.5), in which $N = 0$. However, as more readings are observed, the probability distribution will converge to the empirical maximum-likelihood estimate.

Therefore, the localization algorithm using only signal strengths as features is also weakly affected by dissimilarity in AP detection. Given location l, if the test device observes a new access point i which was not observed by the training device in the same location, Equation 2.4 assigns a minimal score determined from the m-estimator for class-conditional probability to location l for feature i. If the same access point i is present in another location l' instead, this may bias the localization decision to l' over l.

However, the effect of dissimilarity in AP detection is less significant than with algorithms that explicitly use detection rate, because negative evidence — failure to observe access point i — will not be directly incorporated into the localization score in Equation (2.4). In this sense, incorporation of evidence is asymmetric, and the effect of a mismatch in AP detection is less severe than in algorithms that incorporate AP absence information directly (Eqs. 4.2 & 4.3).

Even for algorithms that do not explicitly use detection rate, we expect that the presence or absence of a certain access point will implicitly affect localization results. For example, many instance-based classification algorithms, such as k-nearest-neighbor or support vector machines, require choosing a value for each missing entry in each instance. A typical value used for WiFi localization is -100 dBm, encoding prior information that non-detected APs are expected to be far away, and that if they were detected it would be with low signal strength. However, this effect is symmetric in contrast to the signal strength–based Bayesian algorithm presented in this chapter.

4.6 Related Work

Relatively few researchers have addressed the problem of using heterogeneous devices for localization. For GSM localization, Chen et al. tested cross-device localization using three different devices, showing that the heterogeneity of training and test devices considerably degrades the accuracy of their fingerprinting method [90]. Kaemarungsi compared RSSI values from different devices, but did not evaluate their effect on localization [91].

Researchers have proposed several methods for compensating for differences in signal strengths or RSSI values. Linear transformation from one device to another has been computed either manually or on-line using an expectation-maximization algorithm [11,86,92]. Dong et al. suggested using the difference between signal strengths across access points, rather than the absolute signal strength vector, as a localization feature [93]. While the difference between signal strength values is a major factor in localization using heterogeneous devices, we showed that the algorithm must be designed to compensate for the different shape and dispersion of signal strength values among devices.

Detection rate, or response rate, of access points has also been used for RF localization. Bargh and Groote used the inquiry response rate of Bluetooth devices for indoor localization, as signal strength for Bluetooth devices is not readily available without connection establishment [87]. In contrast, 802.11 devices can scan access points without establishing connections. For WiFi localization, Cheng et al. considered response rate as an alternative set of features for localization and showed that its performance is comparable to that of signal strength–based localization [88]. Our results show that while it is possible to use response rate as a feature, doing so will not increase the accuracy of cross-device localization.

4.7 Conclusion

This chapter analyzed device diversity and its effect on localization. We reported simultaneous collection of data from six 802.11 devices in 18 indoor locations. While there is a clear linear correlation of signal strengths across devices, linear transformation alone is not enough for cross-localization: we find that local variations occur on the same order of magnitude as the compensation provided by linear transformation. Instead, wide smoothing can accommodate the different shapes of signal strength distributions across devices, and proves effective for cross-localization. We also found that access point detection rates vary widely across client devices. As a result, incorporating access point presence and absence, in particular, relying solely on this factor to reduce storage costs and simplify positioning, provides poor localization performance when fingerprints are shared across different devices. To better understand this issue, we used Kullback-Leibler divergence to capture device differences with respect to AP detection, and showed that a correlation exists between detection similarity and localization accuracy.

Part II

Motion Compatibility–Based Indoor Localization

Part II presents an indoor localization method that estimates indoor location by matching a user's motion sequence with respect to a prior map. Our localization method is based on the notion of path compatibility: metric, topological, and semantic agreement of a user motion sequence with the prior map in indoor environments.

We outline the overall architecture of the motion compatibility–based localization system in Chapter 5. Our localization system consists of three parts: a *motion classifier*, which takes low-level sensor measurements as input to produce fine-grained motion estimates; a *route network generator*, which parses legacy floor plans to create "route networks"; and a *trajectory matching algorithm*, which finds the most likely path of a sequence of motion estimates on the route network.

In Chapter 6, we present a CRF-based classification algorithm. The classification algorithm produces accurate motion estimates at fine granularity from uncalibrated data capture by mobile sensors.

Chapter 7 explains how to create route networks, graph representations of indoor paths, from legacy floor plans. From space contours, horizontal and vertical adjacency, and space semantics coded in the floor plans, we create annotated route networks that is used for matching user motions during map matching.

This part concludes by describing a matching algorithm that takes the motion estimates and route networks as input to recover user paths. In Chapter 8, we encode path compatibility in an HMM-based matching model, from which the method recovers the user's location trajectory from the low-level motion estimates. We also show empirical results, demonstrating that our method can recover an accurate trajectory for a user using only proprioceptive sensor data of the quality typically available on modern smartphones.

Chapter 5

Motion Compatibility–Based Indoor Localization: Overview

We begin with motivations behind the motion compatibility–based localization method (§ 5.1). Then we present a brief overview of the overall architecture of the system (§ 5.2). The testing platform and sensors with which the method was deployed are explained next (§ 5.3). We conclude this chapter by reviewing related work that influenced the design of our method (§ 5.4).

5.1 Introduction

Most 802.11-based indoor localization methods require surveying the deployment area to construct RF-fingerprints [10–12]. To mitigate the mapping burden, researchers have proposed localization systems that either use contributions from end-users by crowdsourcing [1], or algorithms that infer the RF fingerprint map from fewer or no location labels [17, 94]. Although these improvements can reduce the initial deployment burden, RF-based methods have other restrictions: they typically construct a fingerprint map only over a short time interval, which will have limited utility at other times due to the time-varying nature of RF signals; and the high-frequency RF scanning required for location updates (especially continuous updates) can be energetically costly.

Recently, localization algorithms which use sensors found in off-the-shelf mobile devices have been proposed [16, 18]. Such methods extract distance and heading measurement from MEMS IMUs, and estimate user position and attitude through dead reckoning. Since MEMS IMUs tend to drift quickly over time, these methods require sensors to be placed in specific positions (e.g. on the feet), or rely on frequent external position fixes from another source (e.g. WiFi-based localization systems). Others use Kalman filters or particle filters to account for measurement uncertainty [15–17]. However, these still depend directly on low-level sensor measurements. Also, these forward-filtering–based methods are often formulated to update only the latest location given new measurements; they do not recover the user's recent path history.

In contrast, our work attempts to recover the *entire* user trajectory from a sequence of *discrete* motion descriptions. This approach is inspired from the way that humans describe indoor routes in abstract terms, including motion descriptions (e.g. "walk" or "turn left") and motion-related actions (e.g. "open the door") rather than precise distance and angle specifications. (Human motion descriptions can include absolute directions (e.g. "north") or durations (e.g. "for 5 seconds"), but these are typically interpreted qualitatively as detailed annotations of more abstract navigation guidance.)

Humans can also perform the inverse process — inferring a route given a motion sequence and an indoor map. Given a sufficiently long motion sequence, we can narrow our idea of the route taken using various kinds of motion-related information: implied walking lengths and turn angles (geometric constraints); spatial layout and path continuity (topological constraints); and agreement of the space type and motion type (semantic constraints). In other words, one can view location estimation as a decoding process in which the originating route is inferred from an observed sequence of motions, combined with spatial constraints imposed by the prior map.

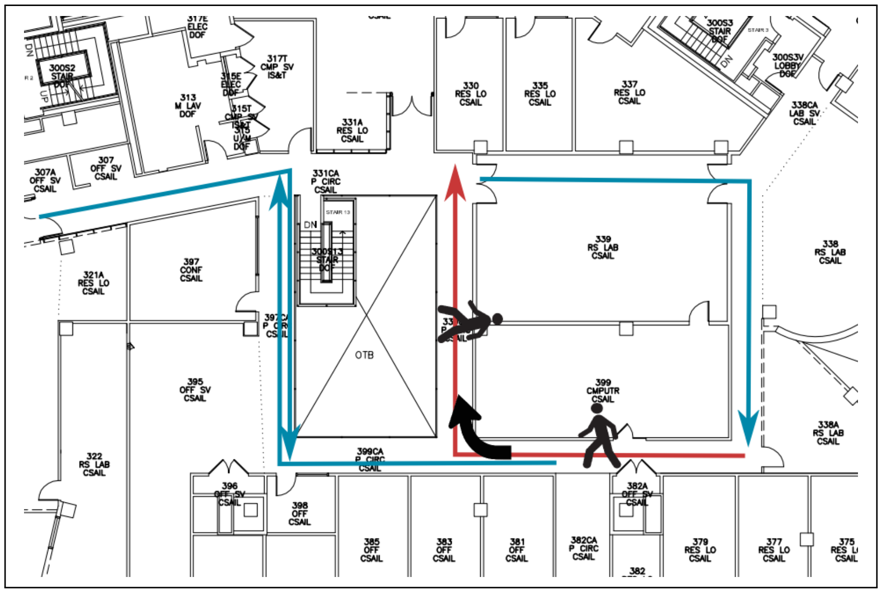
Part II of the thesis describes an indoor localization method that codifies this intuition. In the next section, we outline the overall design of the localization method.

5.2 Overview

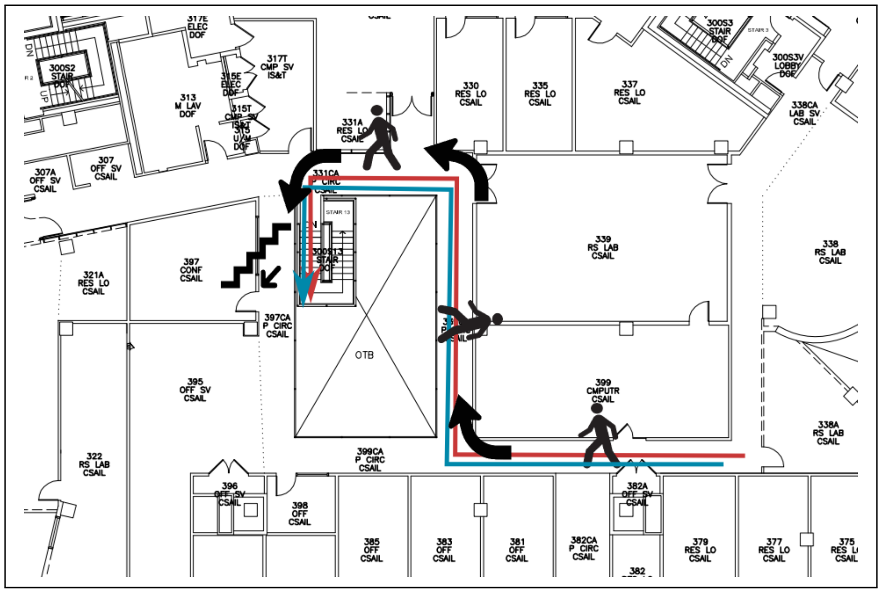
In the map-matching of a sequence of navigational descriptions on a prior map, we use the notion of "path compatibility", an agreement between user motions and trajectory hypotheses.

Figure 5-1 illustrates this idea for a walking user carrying a mobile device. The originating path consisting of two straight-line segments in Figure 5-1a gives rise to a series of three-motion segments: two straight-line walks and a right turn between them. Using the sensors equipped in the mobile device, the user's recent motions can be recognized, along with some physical properties associated with the motions (e.g. duration, walking speed, etc.). Given a prior map and the motion sequence estimates, we can attempt to recover the user path by matching the motions on the plausible paths in the map. During this process, we use various constraints that naturally arise from human movements in indoor environments: e.g., a path must be contiguous, walking motions match straight-line segments, or turn motions match intersections.

However, a path with only a few motion segments, as in Figure 5-1a, especially when we do not have access to accurately measured physical properties (e.g. walking speed or turn



(a)



(b)

Figure 5-1: Path compatibility. Red paths show true user paths; blue paths correspond to (some) compatible paths. (a) After only three motions (Walking – Right Turn – Walking), there exist many compatible paths; (b) with additional motions, the seven-motion path (Walking – Right Turn – Walking – Left Turn – Walking – Left Turn – Going Down Stairs) can be embedded in this map in only one way.

degrees), could be matched to many candidate paths in the floor plan. However, as the user continues to move (Fig. 5-1b), it can be embedded on the map only in an increasingly limited number of ways. In particular, certain motions, such as walking on stairs, provide strong constraints on the user path, as the number of locations in which those motions can occur is typically small in a usual building.

The example above illustrates the elements required for the motion compatibility–based localization to be implemented as an automatic process. First, we need a motion labeling method that classifies user motion accurately from noisy sensor data formed by the low-cost sensors typically available in consumer-grade mobile devices. Second, we require a proper representation of all possible user paths, which encodes metric/topological/semantic information on possible user motions in the environment. Last, a matching algorithm that can handle uncertainty in the motion estimates as well as in the map representation is required. Figure 5-2 depicts the whole process of map matching from the raw sensor inputs to yield matched trajectories. We give a brief overview for each component in the following sections.

Summarizing, our motion compatibility–based localization method in this thesis consists of several components. The core map matching algorithm uses a hidden Markov model (HMM) to find the most likely path given a motion sequence (Ch. 8). The method takes as input a sequence of motion labels, automatically generated by a CRF-based low-level motion labeling algorithm that takes raw sensor measurements from a hand-held mobile device (Ch. 6). It uses a "route map" extracted automatically from legacy floor plan data (Ch. 7). The matching model (§ 8.2) defines a compatibility measure between the input motion and candidate paths, among which the best path is found by HMM algorithms (§ 8.3).

5.2.1 Automatic Motion Sequence Labeling (Chapter 6)

Our labeling algorithm takes time-stamped sensor data as input, and outputs user motion states. The sensor measurements come from consumer-grade sensors, including accelerometer, gyroscope, barometer and magnetometer (see § 5.3). The labeling algorithm uses a

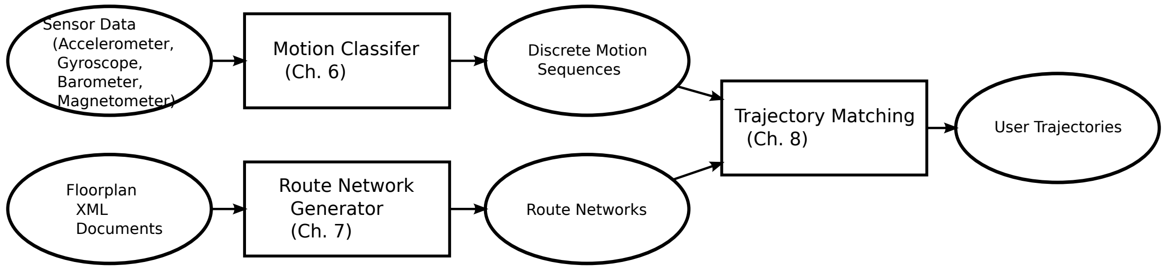

Sensor Data (Accelerometer, Gyroscope, Barometer, Magnetometer)
Motion Classifer (Ch. 6)
Discrete Motion Sequences
Trajectory Matching (Ch. 8)
User Trajectories
Floorplan XML Documents
Route Network Generator (Ch. 7)
Route Networks

Figure 5-2: Overview of motion compatibility–based indoor localization

conditional random field (CRF), a class of probabilistic models used for structured predictions [95], allowing use of long-range and/or overlapping features.

From raw IMU sensor data, the method extracts features including median, variance, and peak frequency of acceleration magnitude, average yaw rate (from the gyroscope), azimuth change (from the compass), the frequency of deviations, and the frequency of pre-defined characteristic patterns. The CRF model learns optimal association weights between each of these features and the annotated motion labels. We annotated user motion data with motion labels using simultaneously recorded video, and used them to train the algorithm.

5.2.2 Route Networks (Chapter 7)

Our motion-based matcher requires a *route network*: a graph representation of all possible user paths within the environment. We implemented an automatic graph construction algorithm that analyzes and interprets floor plan data to generate a route network created from generalized Voronoi graphs. A Voronoi-based route network representation has been used in prior work for robot or human navigation and map matching in indoor environments [73, 96, 97].

Our route network generation process builds upon prior work by Whiting et al. [73], which uses a constrained Delaunay triangulation to approximate the medial axis of each space. A route graph is generated for each space using the Delaunay triangulation of each space, then it is combined with other graphs for spaces abutting via horizontal and vertical "portals" to create a complete route network for the entire corpus. Figure 5-3 shows a route network for one floor of the deployment building.

We use an automated process to generate route networks from floor plans. However,

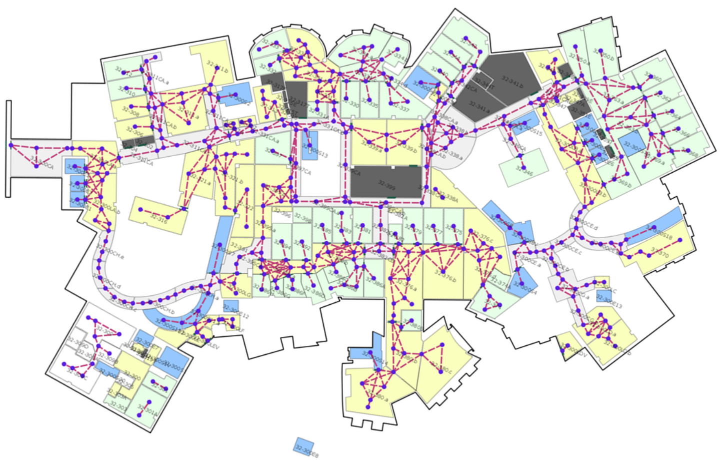


Figure 5-3: Route network (red, blue) of the third floor of the MIT Stata Center, extracted from a legacy floor plan.

manual authoring would also be feasible in many settings. Since for most buildings only a portion is occupied by end-users (e.g. in malls, hospitals and airports), and interior structure tends to change slowly over time, the maintenance burden to author and keep current the building's route network should be tolerable.

5.2.3 Trajectory Matching (Chapter 8)

The trajectory matching algorithm is based on a hidden Markov model (HMM). Given the inputs from the motion classifier and a route network, we define "path compatibility", the agreement between the conceived path from motion estimates and a path hypothesis in the route network. To find the path that maximizes the compatibility, we formulate the problem as a sequence labeling problem, in which each motion is labeled one of elements in the route graph.

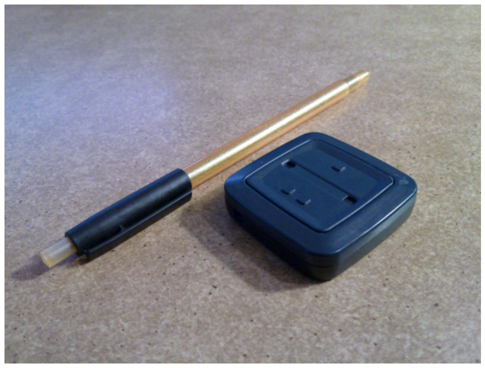
By defining proper transition and emission models, we can use HMM algorithms to infer the most likely trajectory of a user given an observed sequence of his/her motions. The HMM framework also allows us to find a good set of model parameters without the need of annotated data.

5.3 Platforms and Sensors

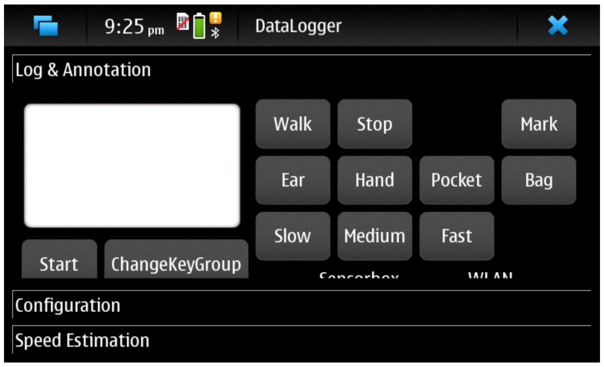
We design our method to operate with consumer mobile devices that people use in everyday life, such as smartphones and tablets. Recent mobile devices have a variety of sensors in them, including accelerometers, gyroscopes, magnetometers, proximity sensors, light and acoustic sensors, and barometers. Even though indoor localization methods, in principle, can benefit from any of these sensors by modeling an association from the signals to the physical locations, we chose to use the sensors available in the Nokia Sensorbox (Fig. 5-4a) in this work.

We use a Nokia N900 smartphone with a Nokia Sensorbox to demonstrate that our method can be applied to commodity mobile devices. The Nokia Sensorbox is a sensing device developed for research and containing five sensing modalities in compact $4 \times 4 \times 1$ cm package: consumer-grade tri-axial accelerometer, tri-axial gyroscope, tri-axial magnetometer, thermometer, and barometer. Currently, many smartphones include a similar or the identical set of sensors. Accordingly, we expect that the method presented in this thesis can be applied to commodity devices without modification. Table 5.1 shows the specifications of the sensors in the Nokia Sensorbox.

The Nokia Sensorbox was connected to a host computer, a Nokia N900 mobile phone, via Bluetooth. We wrote a data logging program (Fig. 5-4b), written in C++ and Qt toolkit,



(a)



(b)

Figure 5-4: Nokia Sensorbox and data logging program

that records time-stamped sensor data transferred via Bluetooth. Ground-truth annotations were made by capturing simultaneous video and later annotating it.

5.3.1 Sensor Characteristics

We conducted a series of analyses on sensor data from the Nokia Sensorbox. Our summary of findings are that:

1. Low-cost sensors used in the Sensorbox, and mobile devices in general, at their factory default states are not precisely calibrated. They show significant bias and drift, which would make dead-reckoning–based approaches for indoor navigation inappropriate.

2. In indoor environments, earth magnetic field measurements by magnetometers vary significantly depending on the location of the device, irrespective of the heading, due to varying magnetic disturbance. As a result, magnetometer headings do not provide reliable orientation estimates.

3. There are intrinsic limits to the degree to which the sensors can be calibrated. For in-

Sensor	Measurement range	Sampling freq. (Hz)	Std. dev. at rest
3-axis accelerometer	2/4/8 G	100	0.06 m/s^2
3-axis gyroscope	600 deg/s	200	0.4 deg/s
Barometer	30 - 120 kPa	2	1.7 Pa
Thermometer	-	2	0.043 °C
Magnetometer	710 μT	20	1.2 μT

Table 5.1: Nokia Sensorbox Specifications

stance, the drift characteristics of gyroscopes are difficult to calibrate precisely without a turntable. Moreover, even after calibration, physical quantities that are computable by integration from raw sensor measurements, such as distance or orientation, are highly inaccurate, when integrated over long periods.

These findings imply that naïve methods that simply integrate raw measurements over time would not work reliably in practice. They also indicate that the motion classification and map matching methods must be designed in a way that they are robust against such undependable characteristics of low-cost sensors. These considerations provided some bases for the design of methods we present in the following chapters.

5.4 Related Work

We review prior work on indoor localization and relevant areas, including robot localization, activity recognition and language grounding for robotic platforms.

5.4.1 RF-Fingerprint Localization

Some localization methods associate ambient RF fingerprints with physical locations, enabling devices to estimate location by finding the stored fingerprint most similar to the current RF observation [10–12]. These methods exploit ubiquitous WiFi infrastructure, unlike approaches which required dedicated localization infrastructure [6, 8]. Since constructing the RF map is an expensive process that requires intensive human labor, there have been a number of attempts to reduce or even remove the burden of manual location labeling, by spatial regression [63], joint mapping of the RF map and access point locations [66], crowdsourcing [1], or by incorporating other types of sensor data [17, 94].

5.4.2 Indoor Pedestrian Navigation and Robot SLAM Algorithms

Recent advances in small, lightweight MEMS sensors have made indoor pedestrian navigation with hand-held devices practical. In particular, low-cost strap-down IMU sensors equipped in recent smartphones draw increasing interest for indoor inertial navigation.

However, the low-cost inertial sensors in mobile devices are usually inaccurate, having high bias and drift characteristics and preventing naïve dead-reckoning from working in practice. The error grows cubically in time, so a double-integration system can diverge more than a few tens of meters in only a minute [16].

To circumvent this problem, prior work has relied on foot-mounting of sensors to re-calibrate IMU at every gait cycle. This "zero-velocity-update" scheme has been commonly used for inertial pedestrian navigation systems [16, 18, 98, 99]. The compensated measurements are usually combined with state-space filters such as Kalman filters or particle filters [15, 16, 100]. Woodman [101] studied drift and bias characteristics of MEMS inertial sensors, suggesting that error in orientation estimate due to gyroscopic error is a main source that limits overall accuracy. Other methods used position fixes from external measurements [15].

The use of filtering algorithms for state-space models has been widely explored in the robot localization and mapping community. The robotics community has been developing methods for robots to explore an unknown environment and build a map representation of landmarks automatically from sensor data. Recent work on simultaneous localization and mapping (SLAM) [33] uses diverse sources of metric/semantic information from a rich set of sensors, including IMUs, LIDARs, and depth-enabled cameras, equipped on a robot or as a wearable device [102, 103]. Also robot platforms have explicit measurements of control signals, which cannot be well measured for humans. Since such active, exteroceptive sensors are unlikely to be incorporated into smartphones in the near future, our work addresses the challenges of accurate location discovery from the cheaper sensors available today. Prototypical SLAM implementations include EKF-SLAM, GraphSLAM, and FastSLAM, differing in what kind of estimation algorithms they are based on. Among these, GraphSLAM [68] is a nonlinear optimization formulation over graphs, and similar to the graph reconstruction approaches presented in Sections 2.2.4 and 2.2.5.

5.4.3 Activity Recognition

Another key component of the system involves recognizing the physical activities of the user. Researchers have proposed different types of wearable computers that recognize user activity from multi-modal sensor inputs. Exploiting recent advances in low-cost, small sensors, they attach sensors that read motion from various parts of the body, and infer user activities from sensor measurements.

In particular, acceleration data are often used for recognizing physical activities [104]. Acceleration data are particularly useful for recognizing cyclic activities, such as ambulation, by extracting frequency domain features using FFT. Ravi et al. studied a variety of classifiers, including meta-classifiers such as bagging and boosting, using acceleration data from a single triaxial accelerometer [105]. Several prior work attempted to incorporate temporal dynamics; Lester et al. [106] combined boosting with an HMM to capture temporal smoothness of human activity. Krause et al. [107] proposed unsupervised identification of user activities using k-means clustering and self-organizing maps.

Researchers have also proposed methods to recognize higher-level activities other than primitive motions. Inferring higher-level activities often involves hierarchical models, such as topic models [108], as each high-level activity is typically composed of several low-level activities.

5.4.4 Route Networks

Route networks, metric-topological representations of routes, have been widely studied in the context of localization and path planning for vehicles, humans, and robots.

Outdoors, road networks unambiguously define route networks for vehicles. Map matching on a road network is usually formulated as a trajectory matching problem, to exploit known vehicle dynamics and handle inaccurate GPS observations [109, 110]. Sensor measurements from users' mobile devices are often combined to enhance matching accuracy [111, 112]. Just as the state model for outdoor map matching is naturally given by a road network, we derive an analogous "route network" from a legacy floor plan. We do not assume any external position infrastructure (e.g. Loran, GPS).

However, creating a route network for an indoor environment poses its own challenges due to intrinsic ambiguity of the definition of "routes" in indoor spaces. A widely adopted scheme is to generate a graph so that edges (which represent routes) follow the shapes of the spaces. To recognize a shape of a space computationally and derive a graph from it, the medial axis representation [113] and its approximations, such as generalized Voronoi graphs [114], have been used. In particular, generalized Voronoi graphs, or equivalently, constrained Delaunay triangulations, have been used in the context of robot localization in indoor environments [96, 97].

5.4.5 Human Navigation in Indoor Environments

Our use of low-level motions to recover high-level trajectories is motivated by human perception of indoor route navigation. Studies suggest that humans rely on geometric cues (i.e. space layout) as well as non-geometric cues (e.g. landmarks) when learning and navigating spaces [115]. Based on this idea, Brush et al. [116] performed a user experience study for activity-based navigation systems that present a trail of activities to the destination. Also, there have been recent attempts to make an automated agent "ground" spoken natural language to support navigation and mobile manipulation [117–119].

Chapter 6

Accurate Classification of Navigational Motions using Mobile Devices

In this chapter, we present a classification method that estimates fine-grained user motions from low-cost MEMS sensors in mobile devices. We first define a factorized representation of motions based on typical navigation activities in indoor environments (§ 6.1). The classification algorithm is based on a conditional random field (CRF) model parameterized with "feature templates," which associate sensor data with motions with varying window widths (§ 6.3–6.4). The performance of the method is evaluated with a test dataset (§ 6.5).

6.1 User Motion Models for Indoor Navigation

The motion classification method in this chapter aims to produce fine-grained user motion estimates that are representative of the navigation activities of users carrying a mobile device in indoor environments. Consequently, our modeling of user motions is directly inspired by how humans describe indoor routes.

We model user motion traces as a series of *discrete* actions parameterized by properties associated with each motion. A property can be either a discrete or a continuous value, representing the characteristics of the associated motion (e.g. duration, velocity or direction). In indoor navigation scenarios, most walking paths can be well-modeled by the following set of actions:

Rest Periods of little or no net motion, for example, while seated in an office, classroom, cafe, library, etc. Detection of user resting can be used to favor certain place types over others when estimating location.

Sitting Down and Rising These motions separate Rest from non-Rest activities in time.

Standing Standing with little horizontal or vertical motion, after arising or between other motions.

Straight Walk The user walks approximately along a straight line. Total traveled distance can be estimated either directly from the acceleration trace, by counting user strides, or from duration (assuming constant walking speed).

Turn Change of walking or standing direction over a brief period. We quantize turns on an eight-point (body-centric) compass rose.

Walking Ascent and Descent Walking on ramps and stairs (on spiral or multi-stage stairwells, often accompanied by successive turns.

Elevator Periods of vertical ascent or descent in an elevator. Elevator motions typically do not involve Walk or Turn motions.

Access Auxiliary actions required to move within typical indoor environments, including opening doors and pressing elevator buttons.

These motion descriptors convey both local and global information about the user's motion and location. Locally, they describe the current status of user's "egocentric" motion, i.e., how the user is moving, or changing the motion state. Globally, they convey navigational cues that constrain the user's current location given a metric-topological-semantic indoor map. For example, detection of an elevator ride motion unambiguously narrows down the user's current location to one of the elevators in the map. The matching algorithm in Chapter 8 is formulated to use both kinds of information.

The complementary set of sensors available in the Sensorbox (§ 5.3) — accelerometer, gyroscope, magnetometer and barometer — provides sufficient sensing data required for estimation of the motions described above with high accuracy, as they produce signals arising from user motions along different axes, capturing linear, rotational, horizontal, and vertical motions, respectively.

6.1.1 Classification Labels: Factorized Representation of Motions

The design of classification labels reflects the motion models described in Section 6.1. However, for the purpose of classification, we do not use the motion descriptor terms as they are, but use a *factorized* representation of motions instead.

The major parts of the motion descriptors in Section 6.1 are egocentric descriptions of navigational activities. A navigational motion in a typical 2.5-dimensional indoor space consists of three components: *Principal* mode of motion (e.g. sitting or walking), *Lateral* component (e.g. left/right turn), and *Vertical* component (e.g. stair or elevator ascent/descent). Hence, a basic motion "clause" describing a navigation action can be represented as a Cartesian product of three orthogonal motion components:

$$\mathit{Principal} \times \mathit{Lateral} \times \mathit{Vertical}$$

where the non-null symbols in each components are:

$$\begin{aligned}
\mathit{Principal} &= \{\ \mathsf{Sitting},\ \mathsf{SittingDown},\ \mathsf{StandingUp},\ \mathsf{Standing},\ \mathsf{Walking},\ \mathsf{Running}\ \}\\
\mathit{Lateral} &= \{\ \mathsf{Straight},\ \mathsf{LeftTurn},\ \mathsf{RightTurn},\ \mathsf{LeftUTurn},\ \mathsf{RightUTurn}\ \}\\
\mathit{Vertical} &= \{\ \mathsf{Flat},\ \mathsf{StairUp},\ \mathsf{StairDown},\ \mathsf{ElvatorUp},\ \mathsf{ElevatorDown}\ \}.
\end{aligned}$$

Most basic navigation actions can be described as a product of these three motion components. For example,

$$\begin{aligned}
\text{Walking straight on a flat surface} &\Rightarrow (\text{Walking, Straight, Flat})\\
\text{Turning right while climbing stairs up} &\Rightarrow (\text{Walking, RightTurn, StairUp})\\
\text{Riding an elevator down} &\Rightarrow (\text{Standing, Straight, ElevatorDown}).
\end{aligned}$$

This decomposition eases analysis and design of the method in many ways. For example, some sensors measure physical values associated with only a specific motion component, e.g., barometric measurements are related only to the vertical component, i.e., Flat, Up or Down. This reduces unnecessary modeling effort for many combinations of motions and sensors.

Still, there exist other instances in which the user motion can be better explained by introducing some non-primitive, special descriptors into our vocabulary of motions. In particular, certain types of motions are bound to occur only (or at least with high probability) at certain types of places. To take advantage of such prior knowledge, we introduce the fourth component, the *Auxiliary* axis, to explain auxiliary actions required to move within indoor environments. In this work, we define two additional actions that occur frequently when moving within a typical building:

$$\mathit{Auxiliary} = \{\ \mathsf{DoorOpen},\ \mathsf{ButtonPress}\ \}.$$

This component is exclusive of the basic motion component. That is, when these actions occur, we do not set values for the first three components.

Summarizing, a classification label is a product of the three basic motion components, augmented with an auxiliary set of actions:

$$\mathit{Principal} \times \mathit{Lateral} \times \mathit{Vertical} + \mathit{Auxiliary}.$$

When representing a classified label for a motion, we use a shorthand representation in which we use either only the first three components or the last component e.g., (Walking, Straight, Flat) or ButtonPress.

6.2 Segmentation

The input data, consisting of multiple data streams from four sensors running at different rates (Table 5.1), must be segmented before being labeled. Each segment, or *frame*, is tagged with one of the motion labels that we define in Section 6.1.1. As we aim to provide fine-grained motion estimates useful for trajectory matching, determining a good segmentation scheme is crucial for achieving high performance of the localization system.

In general, pre-classification segmentation methods can be categorized into two schemes: *variable-length* and *fixed-length* segmentation. Variable-length segmentation methods attempt to find "good" segmentation boundaries by analyzing signal characteristics based on some prior knowledge, for example, by placing a boundary where the change in the magnitude of the signal exceeds a threshold. In this scheme, each frame can have a distinct length, and two adjacent frames are more likely to have different labels than in the fixed-length segmentation. On the other hand, in fixed-length segmentation, input signals are sliced into fixed-length frames. If a segmentation interval used is small, the same label may repeat over multiple frames to represent a long motion; those successive frames may be concatenated later to yield a single frame with a longer duration. (In this case, segmentation boundaries are implicitly determined by the labeling method.)

In this work, we use a fixed-length segmentation scheme in which the labeling algorithm assigns a motion label at 333 ms intervals (3 Hz). We use the fixed-length segmentation, because the variable-length segmentation scheme may have the following disadvantages: 1) an ill-designed variable-length segmentation algorithm may fail to partition critical motion segments, such as short-term turn motions; and 2) the features, computed for each segment, must be carefully normalized so that they are independent of the duration of individual segments. On the other hand, the fixed-length segmentation scheme does not suffer from such issues.

We chose the frame interval as 333 ms (3 Hz), which is shorter than the duration of most motions. The major challenge in finding a right segmentation interval is that different motions can have very different durations. While short-term motions, such as turns or door-open actions, last no more than one or two seconds, longer actions such as walking or riding an elevator can last for as long as a few tens of seconds or more. If a large value (e.g. 3 seconds) is used, it may cause the signals from the short-lived motions to be "buried" in feature computation. On the other hand, a smaller interval makes it possible to represent a motion as a series of small frames. When concatenated, the small frames can determine the boundaries (the start and end) of each unit motion, giving a precise duration estimate.

However, such a small frame size may not be optimal for longer motions or slower sensors. For example, the barometer in the Sensorbox measures atmospheric pressure every 0.5 seconds; with 3 Hz frames, each frame will have at most one pressure value, making it unable to compute the gradient of the atmospheric pressure, which is an essential feature in detecting vertical motions. Also, even for high-rate sensors, features may not contain faithful characteristics of a longer motion; the 333 ms window does not contain a full single walking cycle occurring at around 1 Hz. Therefore, the computed features may be very noisy (i.e., computed statistics have a high sample variance) if computed for a small interval. On the other hand, if we use longer, overlapping windows, the features become dependent and may violate the independence assumptions required for certain sequence models (e.g. HMMs), making the model double-count the same features. To overcome this problem, we use conditional random fields (CRFs), which allow the use of long-range and/or overlapping features, while segmenting input data with the short interval of 3 Hz. The feature functions we use are designed to adapt for different lengths of motions by using overlapping windows (§ 6.4.1). The CRF-based motion labeling method is described in the next section.

6.3 Conditional Random Fields

With the motion descriptors described in Section 6.1, our task is to infer the most probable sequence of motion labels given time-series sensor data from the user device. To this end, we use conditional random fields (CRFs) [95], a class of probabilistic models for structured predictions. In particular, we use a linear-chain CRF, in which the output (motion) variables are arranged linearly over time.

In the discussion below, let $\mathbf{x}$ be the latent variable we infer and $\mathbf{y}$ be the observation variable. In the motion labeling problem, $\mathbf{x}$ represents a sequence of motion states and $\mathbf{y}$ denotes time-series sensor data, segmented as we described in Section 6.2. x_j and y_j denote j-th frame of the label and the observation sequence, respectively. A CRF is a discriminative probabilistic model that learns a conditional probability, $p(\mathbf{x}|\mathbf{y})$, directly from the training data.

A linear-chain CRF, a discriminative model for sequences, can be written as [120]:

$$p_\lambda(\mathbf{x}|\mathbf{y}) = \frac{1}{Z_\lambda(\mathbf{y})} \exp\left(\sum_j \sum_i \lambda_i f_i(x_{j-1}, x_j, \mathbf{y}, j) \right) \tag{6.1}$$

where $f_i(\cdot)$ is the i-th feature function representing compatibility, or the desired configuration, between two successive states x_{j-1}, x_j and $\mathbf{y}$, and λ_i is the feature weight for f_i. In linear-chain CRFs, feature functions are restricted to represent interaction between the observation $\mathbf{y}$

and, at most the previous state and the current state only, instead of allowing arbitrary interactions between $\mathbf{y}$ and $\mathbf{x}$. This makes the model to have a chain structure, allowing the use of efficient dynamic programming–based algorithms for inference [120].

CRFs do not suffer from the restrictive independence assumption between observations. On the other hand, in HMMs, the observation at frame i, y_i, is assumed to be dependent only on x_i. This assumption is restrictive for the motion labeling problem in which long-range dependencies between observations are apparent. For example, an elevator motion is almost always accompanied with a pair of upward and downward accelerations at its start and end; while a user is walking, the measured acceleration signals exhibit the same oscillation pattern for a long period of time. HMMs cannot be used to model these long-range, overlapping features; otherwise it will accrue the same evidence multiple times in the inference.

6.4 Features

We compute features using low-level sensor measurements from four sources of proprioceptive sensors available in the Sensorbox: a tri-axial accelerometer, gyroscope, magnetometer and a barometer. Before extracting features, sensor measurements are smoothed by low-pass filtering. Gyroscope and magnetometer measurements are aligned to match the vertical axis (gravity direction) and the horizontal plane (the orthogonal plane to the gravity direction) using a tilt angle estimate from accelerometers. In this way, lateral features can be extracted regardless of the device orientation.

In Section 6.4.1, we describe how we utilize the strengths of the CRF-based models — their ability to incorporate overlapping features — in the motion labeling problem by the use of sets of the same feature functions computed on varying window widths, or *feature templates*. Then, we explain how sensor data are preprocessed and feature functions are extracted from them.

6.4.1 Feature Templates

A *feature function* is any real-valued function that defines compatibility between labels, $\mathbf{x}$, and observations, $\mathbf{y}$. A typical example is a statistic derived from physical measurements, e.g., the median of the acceleration magnitude; a certain range of median values may indicate that certain types of motions are more likely. However, a feature function does not have to be limited to such real-valued statistics. For example, it could be a boolean function indicating whether or not a certain predicate is satisfied or not (1 or 0), i.e., whether a specific pattern is found in the current observation window.

From sensor data, we compute various types of feature functions, each of which uses a different combination of sensors. Some examples include: the variance of the acceleration, the location of the peak frequency in the acceleration spectrum, the gradient of the atmospheric pressure, or the number of up-down patterns in the signal. In labeling x_j for a certain time frame j, CRF models allow using evidences drawn from *any* frames, not only the observation within the j-th frame. That is, in computing feature functions, we can use observed signals from either a single frame ($\{y_j\}$), adjacent frames (e.g. $\{y_{j-3}, ..., y_{j+3}\}$), or even distant frames that do not include j (e.g. $\{y_{j-5}, ..., y_{j-3}\}$).

However, complications arise when deciding how large a feature window should be used for each feature. That is, we often do not have a precise prior knowledge on the range of interactions between sensor signals and motions. We might guess that, for example, turn motions must be associated with a short window size as they are short-lived, whereas the detection of rest motions can benefit from having a large-sized window, because transient disturbance can be ignored in the large windows. However, window widths that are incorrectly set without a supporting evidence may degrade the performance of the motion labeling method.

Rather, our approach is to learn the range of associations for each feature from data. To this end, we define multiple feature functions for every feature statistic with exponentially growing window sizes, allowing the model to learn an individual weight for each combination of different window sizes and features. For example, we compute the variances of the acceleration magnitude with five different window sizes — 1, 2, 4, 8 or 16 frames. For most features, we use five window sizes (1, 2, 4, 8, or 16 frames, 1 frame = w = 333 ms) centered at the segment to be labeled. (i.e. feature windows are symmetric.) Therefore, to inter a label at frame j of which the timestamp at the center is t_j, we draw evidence from five observation windows, $[t_j - 0.5w, t_j + 0.5w]$ to $[t_j - 8w, t_j + 8w]$ (Fig. 6-1). Exceptions are: 1) slow sensors, e.g. barometer, whose operating frequency is low; and 2) certain types of features whose values are not well-defined for small windows. For these cases, only wider windows (from $[t_j - 2w, t_j + 2w]$ to $[t_j - 8w, t_j + 8w]$) are used.

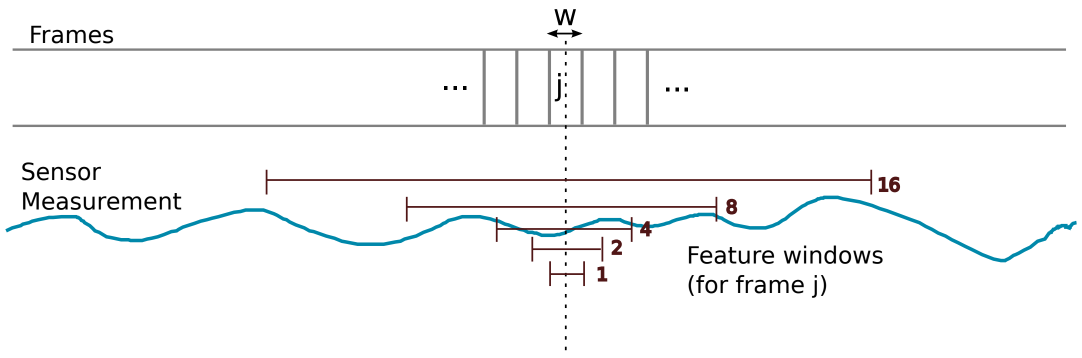


Figure 6-1: Feature windows. For every feature for frame j, multiple feature windows are defined. The feature windows are arranged symmetrically, being centered at the frame j.

The log-linear parameterization of Equation (6.1) brings another design consideration for feature functions: the feature functions must be monotonic in terms of the compatibility strength. That is, the higher (or lower) the value is, the more likely the configuration between input **x** and **y** should be. However, this does not necessarily hold for every statistic. For example, while a moderately high variance of acceleration indicates a high probability of Walking, a very high or low variance may imply Running or Standing, and decrease the likelihood of Walking. To handle this, all real-valued features are quantized into k discrete bins, from which k Boolean feature functions are derived. A Boolean function is "activated" (having the value of 1) only when the corresponding statistic falls within the range of the associated bin.

With these considerations, we define feature templates such that each function derived from of a certain feature type c is evaluated to one if and only if it is assigned a specific pair of labels (for previous and current labels) *and* it has a specific quantized feature value computed from the observation window τ (which is one of the three or five windows):

$$\begin{aligned} f(x_{j-1},x_j,\mathbf{y},j) &= f_c^\tau(x_{j-1}=x',x_j=x'',y_c^\tau=y'^\tau_c,j) \\ &= \delta_c^\tau(x_{j-1},x',j)\,\delta_c^\tau(x_j,x'',j)\,\delta_c^\tau(y_c^\tau,y'^\tau_c,j) \end{aligned} \tag{6.2}$$

where x' and x'' are label values at $j-1$ and j respectively, y'^τ_c is a quantized feature value for feature c with window τ, and $\delta_c^\tau(z,z',j)$ is a delta function evaluated to one if $z=z'$ at time j.

In this work, except for "state transition" features, $\delta(x_{j-1},x',j)\,\delta(x_j,x'',j)$ (analogue of transition probabilities that are independent of observations), we do not utilize the previous labels in feature functions. That is, $\delta_c^\tau(x_{j-1},x',j)=1$, except state transition features. Because we have 152 possible labels (6 principal $\times$ 5 lateral $\times$ 5 vertical + 2 auxiliary) and a maximum of five different window sizes, each feature statistic yields, $152 \times 6 = 912$ Boolean features via the use of templates.

6.4.2 Preprocessing

Before computing feature values, sensor measurements are pre-processed by low-pass filters to remove noisy components and/or to capture nearly stationary components of the motions. We use 3rd-order Butterworth filters to remove noisy high-frequency components while retaining salient low-frequency signals that are directly related to human motions. The cut-off frequency of the filter, which determines smoothness of the filtered signal, is determined empirically, considering the individual characteristics of each sensor and the feature functions using it. In general, the signals from the sensors operating at a high frequency, such as the accelerometer operating at 100 Hz and the gyroscope at 200 Hz, are passed through two different low-pass filters, to obtain different degrees of attenuation: one with a relatively high cut-off frequency (3 Hz at -3 dB cutoff) to remove noise only; the other with a very low cut-off frequency (0.2 Hz at -3 dB cutoff) to capture the slowly-varying trend of the signals. The latter is particularly useful for computing properties that are stationary for a long duration, for example, the tilt angle of the device.

To compute rotational properties around the "yaw" axis, gyroscope and magnetometer measurements must be transformed to be aligned with the gravity vector. To do so, we estimate the device tilt angle, or equivalently, the directional cosines with respect to the gravity axis, from the heavily attenuated acceleration measurements. The directional cosines are used for computing a yaw rate from gyroscope measurements, and an azimuth angle (from the magnetic north pole) from magnetometer measurements.

Magnetometer measurements must be adjusted to compensate for the iron offset caused by external magnetic disturbances. There exist many sources of magnetic disturbances in indoor environments, such as electronic devices and metal furniture. The distortion in magnetic signals caused by these disturbances often has the same order of magnitude as the earth's magnetic field, disrupting accurate estimation of azimuth angle. We compensate the iron interference using the least squares method, assuming that the geomagnetic field magnitude is constant during the operation period and any remaining variation is due to the disturbances [121].

6.4.3 Feature Extraction

From pre-processed sensor measurements, we extract a total of 19 features from the preprocessed sensor measurements (Table 6.1).

State and Transition Bias

The first two features, the state bias and transition bias capture the prior and transition probabilities of the motions, independently of observations. More precisely, the state bias features represent potential energy of state configurations, independently from each other; the transition bias determine potential energy of pairwise configurations of two adjacent states (adjacent in time). The weights for these features are largely determined by how frequently specific configurations of states occur in the training data.

Discretization Methods

As described in Section 6.4.1, all the continuous features are discretized into seven discrete bins. If a feature function is, by definition, symmetric around a certain value corresponding to a reference state, such as zero for the yaw rate in the absence of any turn, we use a symmetric

Feature	Sensors[a]	Processing[b]	Windows[c] (frame)	Binning[d]
State bias[e]				
Transition bias[f]				
Range of accel. magnitude	Acc.	LPF	1-16	Equal
Variance of accel. magnitude	Acc.	LPF	1-16	Equal
Frequency of over-threshold deviations of accel. magnitude[g]	Acc.	LPF	1-16	Equal
Median accel. magnitude	Acc.	LPF-H	1-16	Equal
Number of up-down patterns in accel. magnitude	Acc.	LPF-H	4-16	–
Number of down-up patterns in accel. magnitude	Acc.	LPF-H	4-16	–
High-low-high magnitude pattern[h]	Acc.	LPF	1-16	–
Peak frequency of accel. spectrum	Acc.	128-FFT	1-16	–
Peak magnitude of accel. spectrum	Acc.	128-FFT	1-16	Equal
Average yaw rate	Gyr., Acc.	LPF-H	1-16	Symmetric
Maximum positive yaw rate[i]	Gyr., Acc.	LPF-H	1-16	Equal
Minimum negative yaw rate[i]	Gyr., Acc.	LPF-H	1-16	Equal
Net change in yaw rate[j]	Gyr., Acc.	LPF-H	1-16	Symmetric
Frequency of over-threshold deviations of angular velocity magnitude[k]	Gyr.	LPF	1-16	Equal
Difference of half-window averages of compass azimuth[k]	Mag., Acc.	LPF-H	1-16	Symmetric
Net change in atmospheric pressure[k]	Bar.	LPF	4-16	Symmetric
Difference of half-window averages of atmospheric pressure[l]	Bar.	LPF	4-16	Symmetric

[a]Acc.: accelerometer, Gyr.: gyroscope, Bar.: barometer, Mag.: magnetometer.

[b]LPF: low-pass filtering; LPF-H: low-pass filtering with high attenuation; 128-FFT: 128-point FFT.

[c]1–16: 1, 2, 4, 8 and 16 frames; 4–16: 4, 8, and 16 frames; –: feature values are discrete.

[d]Seven bins. Equal: equal-sized; Symmetric: symmetrically arranged, centered at zero (equal-sized).

[e]Equivalent to prior probability of the state.

[f]Equivalent to transition probability between the states.

[g]Over 0.1 m/s^2.

[h]1 if the predefined pattern of high-low-high acceleration range is detected, 0 otherwise.

[i]Excluding negative or positive yaw rate values.

[j]Between the start to the end of the window.

[k]Over 10deg/s. The magnitude is defined as L-1 norm of the angular velocity vector measure by the gyroscope.

[l]Between the first and the second half.

Table 6.1: List of features extracted for motion classification.

discretization scheme: we place seven equal-sized bins that are arranged symmetrically around the reference point. Otherwise, we partition the data into seven equal-sized bins. To handle measurement outliers, we used values between the 5th and 95th percentile range, excluding outliers.

Acceleration Magnitude–Based Features

Nine features are extracted from the magnitude of the acceleration. The magnitude of the acceleration captures the overall energy of user motion. It is hence directly related to the classification of the principal component of user motions (Standing, Sitting, Walking, StairUp/Down, Running). Among the acceleration magnitude–based features, the range and the variance of the acceleration magnitude, in particular, reflects the overall strength of motion. On the other hand, the frequency of acceleration measurements over a threshold measures the consistency of the activity with high acceleration is during the period of each window.

Some features overlap with each other, showing non-negligible correlation. For example, the range and the variance of acceleration magnitude are strongly correlated and carry similar information in many cases. CRF models, however, can still benefit from having both features, as the models do not pose restrictions on overlapping features. We design the features so that they constitute a complementary set of information sources.

Also, we include some transient patterns in the acceleration magnitude as they may indicate abrupt vertical movements, such as StandingUp, SittingDown, ElevatorUp/Down. For example, a higher (or lower) median value of the acceleration magnitude during a short period of time can indicate that the user is undergoing a transient vertical acceleration in a downward (or upward) direction. Also, when a user stands up or sits down, a pair of abrupt changes, a quick increase in acceleration magnitude followed by a rapid decrease (or vice versa) over a brief period is observed.

When examining these features, we filter the acceleration signals with a heavy-attenuating low-pass filter, using the filtering frequency below the typical gait frequency. It is because that it makes the transient, non-stationary patterns easily distinguishable from the oscillation corresponding to normal gaits.

Acceleration Spectrum–Based Features

We capture the frequency-domain characteristics of user motions from the 128-point fast Fourier transform of the acceleration magnitude. In particular, when a person is walking, the peak frequency of the acceleration spectrum corresponds to his/her half-gait cycle, enabling the classifier to distinguish walking motions easily from other non-repeating motions. At the sampling rate of 100 Hz, at which the accelerometer in the Sensorbox operates, the frequency resolution is 100 Hz/128 points ≈ 0.78 Hz, which provides sufficient frequency resolution for the typical walking frequency at approximately 2 Hz (half-gait). We also extract the corresponding spectral magnitude of acceleration at the peak frequency. This characterizes walking motions in different strengths (i.e. walking on flat surfaces vs. walking on staircases) differently.

Turn Features

The features relevant to turn motions are extracted from the gyroscope and magnetometer. Because the sensors in a mobile device can be positioned arbitrarily, angular velocity and magnetic field measurements must be transformed to be aligned with the local vertical axis. To this end, we use three-dimensional acceleration vectors, which are low-pass filtered with high attenuation and averaged over a sufficiently long period of time, to compute the tilt angle of the device reliably (see § 6.4.2).

We characterize the angular velocity captured by instantaneous yaw rates in multiple ways. Other than the average yaw rate, we use the net change from the start to the end of the feature window. When the length of a feature window is longer than the turning motion, the net change of the yaw rate can become near zero even when a turn is present, whereas the average yaw rate has a non-zero value. Therefore, it helps to distinguish longer turns, such as U-turns, from shorter and/or sharp turns.

Some asymmetric quantities are measured as well. In the right-handed coordinate system with the vertical axis upward, a positive yaw value indicates a left-turn and a negative value represents a right turn. Therefore, computing a one-sided maximum (from only either positive values or negative values) adds more clarity in distinguishing subtle left and right turns from each other.

On the other hand, magnetic field measurements themselves are inaccurate particularly in typical indoor environments, where there exist many sources of magnetic disturbances. Hence, we use magnetic measurements only as a supplementary source of information in addition to gyroscope measurements. To obtain robust turn estimates from inaccurate magnetometer measurements, the difference between the average of the first half and that of the second half of the feature window is taken.

Barometer-Based Features

A ceiling height of a typical building ranges from 7 to 15 feet. This brings a difference in atmospheric pressure of about 50 Pa per floor. A barometer, which begins to be equipped in the current generation smartphones, can measure this difference precisely. We compute the net change and the averaged difference of atmospheric pressure within each feature window.

These features provide useful information when classifying vertical motions, such as walking on stairs and riding an elevator.

6.5 Evaluation

We evaluated the performance of our motion classification method with real user data.

6.5.1 Methodology

To test the performance of the motion classification method, we collected 37 motion sequences from a test user. The dataset was captured in the MIT Stata Center, where the user traversed in a single or multiple floors of the building. The user made occasional transits through elevators and stairs. The total length of the dataset is 68 minutes.

The user held the device (a N900 smartphone with Sensorbox attached) loosely in front while recording first-person video. The recorded videos were used for obtaining the ground-truth motion labels. We developed a motion annotation tool (Fig. 6-2) to support annotation of the recorded user motion data at 100 ms, a finer resolution than the segmentation interval of the classifier (333 ms).

The logged sensor data were offloaded to a desktop computer for offline analysis. The digital sensor values were converted to physical units without any further calibration other than use of the factory default conversion scales and offsets. (We could do this because our motion classifier performs well without such calibration, as it does not rely on any precise physical values in computing feature functions.) Furthermore, the feature functions are discretized, reducing the necessity for keeping precise physical values.

For CRF implementation, we use Wapiti [122], discriminative sequence labeling software developed for NLP tasks. For the evaluation, we perform cross-validation tests, in which one sequence was held out for test and the model was trained using the remaining 36 sequences (leave-one-sequence-out test). The performance metrics were computed from the aggregate results from the cross-validation tests.

The primary metrics for the evaluation are precision and recall. For each class, the precision is the fraction of the correct predictions over all predictions classified to that class. The recall is the fraction of the correct predictions over all examples of that class. That is:

$$\text{precision} = \frac{\text{true positive}}{\text{true positives} + \text{false positives}} \tag{6.3}$$

$$\text{recall} = \frac{\text{true positive}}{\text{true positives} + \text{false negatives}}. \tag{6.4}$$

The F-measure combines precision and recall by the harmonic mean, denoting an overall performance of the classifier for each class:

$$\text{F-measure} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}. \tag{6.5}$$

The classification results were evaluated in per-frame basis. In computing precision and recall, we compute "misalignment-compensated" metrics by allowing misalignments up to two frames. That is, a prediction results for a frame is regarded as being correct if a true label with the same class can be found within two frames from it. performance of

6.5.2 Classification Performance

Figures 6-3 and 6-4 show the aggregate classification performance computed from the cross-validation tests. The overall per-frame precision and recall (after misalignment compensation) were 95.2% and 93.8%, respectively (88.7% and 88.8% without compensation).

Most user motions except a few combinations were classified accurately. However, rare motions (motions that are rarely found in the dataset) have relatively high classification error

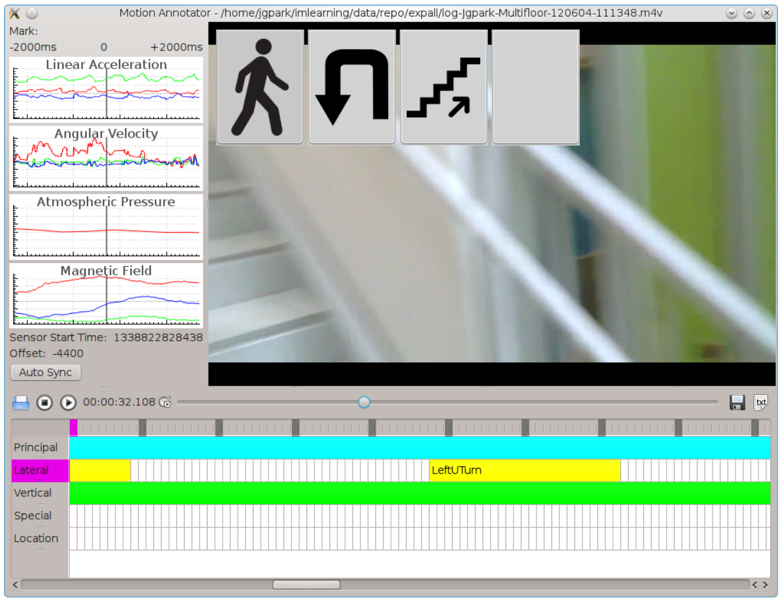


Figure 6-2: Motion annotation software with supporting GUI to aid ground-truth annotation of user motions. The true motion matching an instantaneous video frame can be annotated at fine resolution (100 ms) using the timeline at the bottom of the interface. Sensor measurements are displayed on the left. The program can also be used for visual inspection of the classification results with the graphical icons shown on the top.

```
+---------------------------------+--------------------------------------------------------------------------------------------------------------------+
|         (Walking,Straight,Flat) |<5731>  53   86   62    .   41    9   35    .    .   15    4    .    3    .    1    .    2    .    .    .    3    . |
|        (Standing,Straight,Flat) |   45<1857>   .    1    .    .    .    1   11   10   21    5    .    .    3    .    .    2    .    .    1    .    2 |
|         (Walking,LeftTurn,Flat) |   77    1 <491>   1    .    1    .    3    .    .    1    4    .    .    .    .    .    .    .    .    .    .    . |
|        (Walking,RightTurn,Flat) |   72    .    . <491>   .    1    .    1    .    .    6    .    .    .    .    .    .    .    .    .    .    .    . |
|         (Sitting,Straight,Flat) |    2  162    .    . <140>   .    .    .    .    .    .    .    .    .    .    .    .    2    .    .    1    .    . |
|      (Walking,Straight,StairUp) |   48    1    1    .    . <227>   .    .    .    .    .    .    6    .    .    1    .    .    .    .    .    .    . |
|    (Walking,Straight,StairDown) |   34    .    .    .    .    . <192>   2    .    .    .    .    .    2    .    .    4    .    4    .    .    .    . |
|                        DoorOpen |   63   12    4    4    .    .    . <105>   .    .   44    .    .    .    .    .    .    .    .    .    .    .    . |
|  (Standing,Straight,ElevatorUp) |    .   16    .    .    .    .    .    . <195>   .    .    .    .    .    .    .    .    .    .    .    .    .    . |
|(Standing,Straight,ElevatorDown) |    .   10    .    .    .    .    .    .    . <178>   .    .    .    .    .    .    .    .    .    .    .    .    . |
|                     ButtonPress |   16   21    .    8    .    .    .   30    .    .  <31>  10    .    .    .    .    .    .    .    .    .    .    . |
|       (Standing,LeftUTurn,Flat) |    3    8    .    .    .    .    .    .    .    .    5  <73>   .    3    6    .    .    .    .    .    .    .    5 |
|    (Walking,RightUTurn,StairUp) |    2    .    .    3    .    6    .    .    .    .    .    .  <88>   .    .    .    .    .    .    .    .    .    . |
|        (Walking,LeftUTurn,Flat) |    2    .    .    .    .    .    .    .    .    .    .    .    .  <42>   .    8    .    .    .    .    .    .    . |
|      (Standing,RightUTurn,Flat) |    1    3    .    .    .    .    .    .    .    .    .    .    .    .  <42>   .    .    .    .    .    .    4    . |
|     (Walking,LeftUTurn,StairUp) |    2    .    3    .    .    6    .    .    .    .    .    .    .   16    .  <19>   .    .    .    .    .    .    . |
|  (Walking,RightUTurn,StairDown) |    .    .    .    .    .    .    3    .    .    .    .    .    .    .    .    .  <42>   .    .    .    .    .    . |
|     (SittingDown,Straight,Flat) |    5    2    .    2    3    .    .    .    .    .    .    .    .    .    .    .    .  <27>   .    1    .    .    . |
|   (Walking,LeftUTurn,StairDown) |    .    .    .    .    .    .    4    .    .    .    .    .    .    .    .    .    .    .  <35>   .    .    .    . |
|       (Standing,RightTurn,Flat) |    2    8    .   14    .    .    .    .    .    .    .    .    .    .    .    .    .    1    .   <5>   .    .    . |
|      (StandingUp,Straight,Flat) |   13    3    .    .    2    .    .    .    .    .    .    .    .    .    .    .    .    3    .    .   <8>   .    . |
|       (Walking,RightUTurn,Flat) |    4    .    .    5    .    .    .    .    .    .    .    .    .    .    2    .    .    .    .    .    .  <16>   . |
|        (Standing,LeftTurn,Flat) |    .    9    7    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .   <7>|
+---------------------------------+--------------------------------------------------------------------------------------------------------------------+
```

Figure 6-3: Confusion matrix. Row = reference (true); column = test (predicted). Motion labels are sorted in descending order of the true number of frames in which they occurred.

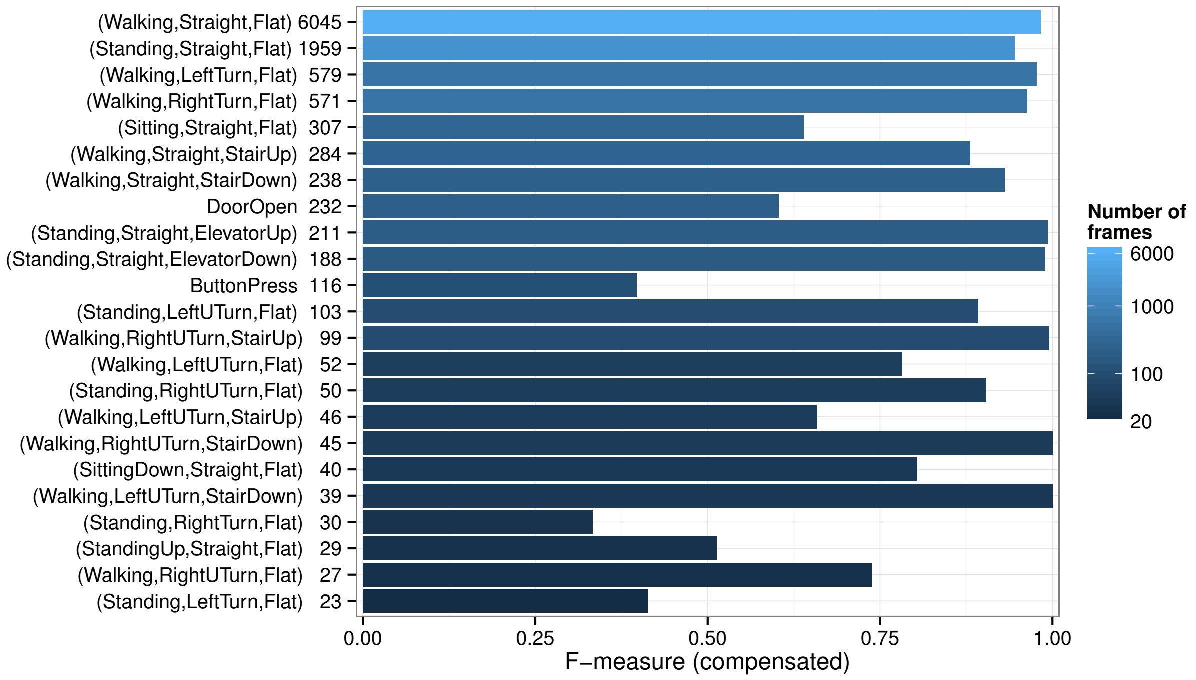


Figure 6-4: Classification performance. (Compensated) F-measures are shown for each motion label. The motion labels are shown along with the corresponding number of frames (1 frame = 333ms) in the dataset.

(bottom of Fig. 6-4). For instance, our dataset contained only 23 frames (7.67 seconds) of (Standing,LeftTurn,Flat) instances ("Turning left while standing on a flat surface"), for which the F-measure was only about 41%. It is because, in the learning phase, the model weights are adjusted to decrease the overall classification error in the parameter estimation, which puts less importance on rare motions that do not contribute much to the aggregate error.

The classification performance is low for pairs of motions that are very similar to each other. In our dataset, two cases are found (Fig. 6-3): DoorOpen vs. ButtonPress, and Standing vs. Sitting (i.e. (Standing, Straight, Flat)) vs. (Sitting, Straight, Flat)). Indeed, these pairs are often nearly indistinguishable; both DoorOpen and ButtonPress involve a short, transient lateral movement, in which a person stretch out his/her hand to manipulate a door or a button.

Also, both standing and sitting are stationary motions with no net movement. They are sometimes distinguishable by the classifier because they often accompany different sorts of motions (i.e. ButtonPress is followed by atmospheric pressure change due to elevator movement, DoorOpen is usually continued to walking, Sitting and Standing accompany StandingUp or SittingDown before *or* after the motion.), but the observation of such clues by the classifier is not always guaranteed.

6.5.3 Feature Windows

We evaluated the effectiveness of long-range, overlapping features by training and testing the classifier with different subsets of feature windows.

We compared our choice of feature windows (all 1, 2, 4, 8 and 16 frames) to the case in which either only the shortest possible windows (1 or 4 frames, depending on the feature) or only the longest windows (16 frames) are used (Fig. 6-5). The overall compensated F-measures for partial windows were 89.0% (90.2% precision, 88.5% recall) and 81.4% (86.9% precision and 79.9% recall) respectively. These are significantly lower than that of all-windows classifier (94.5% F-measure).

Each choice of feature windows degrades performance for certain classes of motions. When only short windows were used, classification error was higher for long motions: U-turns were confused with right turns; vertical motions, especially stair motions (StairUp and StairDown) were misclassified because the short windows did not contain enough number of samples to compute barometric pressure gradient, an essential feature for vertical motions. In contrast, when only long windows were used, there was remarkable performance degradation for transient motions, in particular, turns. The classification accuracy for access motions (DoorOpen and ButtonPress) degraded as well. The result demonstrates that having various feature window sizes is essential to achieve high accuracy in fine-grained motion classification tasks.

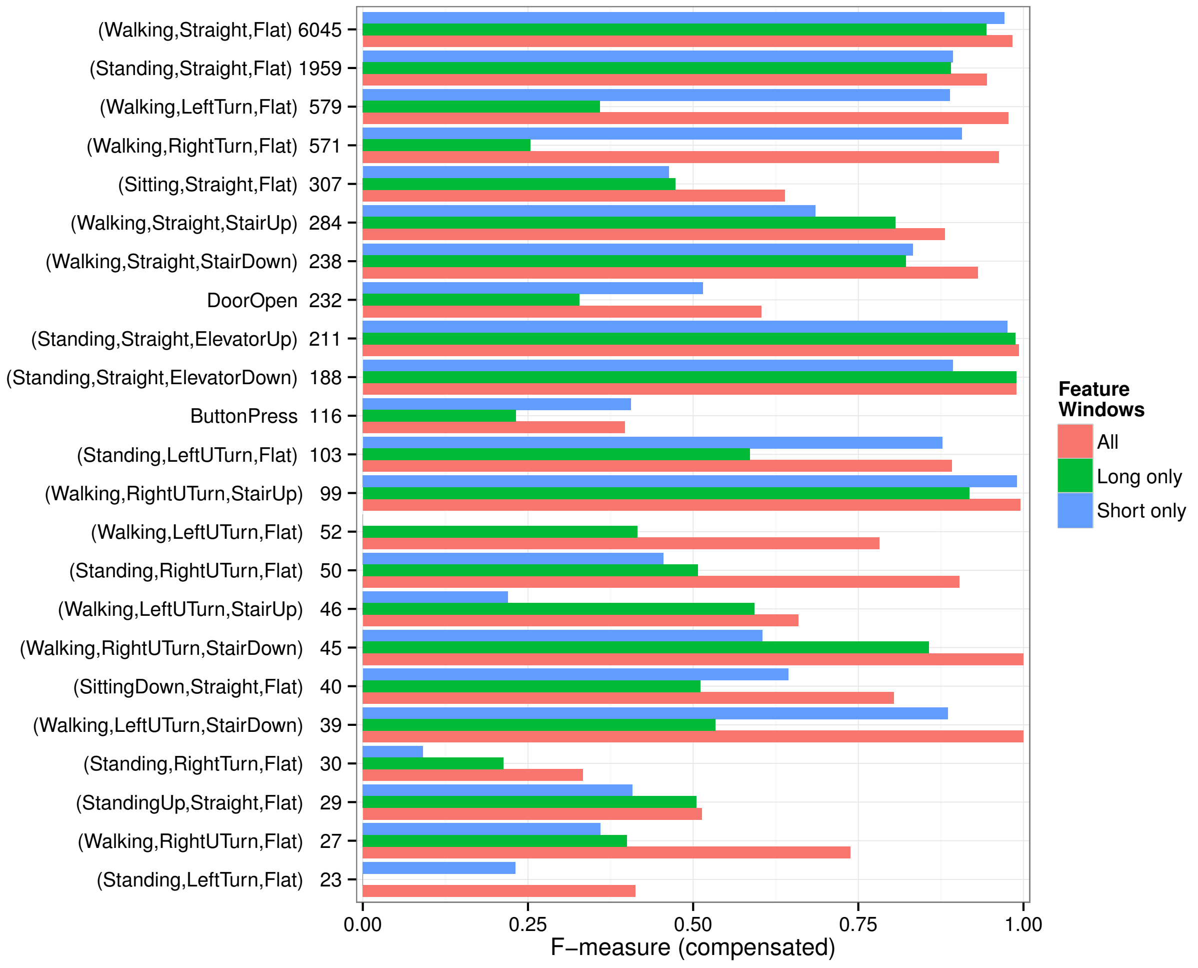


Figure 6-5: Effect of feature window size. (Compensated) F-measures are shown per motion label, when only shortest windows ("Short only"), only longest ones ("Long only"), or all ("All") are used in classification.

6.6 Related Work

6.6.1 Conditional Random Fields

Conditional random fields [95] are a discriminative probabilistic framework for segmenting and labeling sequential data. CRFs achieve tractable inference while avoiding unwarranted independence assumptions over the observation sequence, by modeling a conditional probability of a label sequence given an observation sequence. This is in contrast to generative probabilistic models which model a joint probability between a label and an observation, which is often not necessarily required for classification tasks. CRFs have been used for sequence labeling tasks in many areas, including natural language processing [123], robotics [124, 125], ubiquitous computing [126] and computational biology [127].

6.7 Conclusion

In this chapter, we presented a classification method that infers low-level user motion type from proprioceptive sensor measurements of a quality available in commodity mobile devices. We define a factorized representation of user motions, in which a typical navigation activity is decomposed into four components — principal, lateral, vertical and auxiliary. The CRF-based motion classifier is formulated with the feature templates of varying window sizes, and show high accuracy for a test motion dataset.

As future work, to capture the hierarchical nature of human motions, the motion model can be potentially expanded to a rich, hierarchical structure. This, in turn, would require a parsing algorithm for non-flat, hierarchical structures. Accordingly, the map matching algorithm can be reformulated to match such rich motion models.

Chapter 7

Route Network Generation

As outlined in Chapter 5, finding the trajectory of a user based on his/her motions and the observation of navigational cues requires a matching spatial representation of indoor spaces, namely *route networks*. A route network is a graph representation of all possible routes within an indoor environment. This chapter begins by presenting an overview on how the map source data, MIT Floor plan XML, is structured (§ 7.1). Then we describe our route network generation algorithm, which analyzes the floor plan data to construct a graph representation of user paths in indoor environments (§ 7.2). Also, we explain how spaces in MIT floor plans can be categorized according to the degree of motion (§ 7.3).

7.1 The MIT Floor Plan XML

Automatic construction of the route graphs for our test bed, the MIT Stata Center, builds on previous work of MIT Building Model Generation group, which has built a comprehensive model of the MIT campus from CAD drawings of the building floor plans [73, 128–131]. Whiting and Battat [73, 130, 131] created an exhaustive representation of indoor spaces of the MIT campus, extracting geometric and topological information from the source floor plan data represented in AutoCAD DXF format. The generated floor plan representation, which we call *MIT floor plan XML*, is formatted as XML documents. These XML documents contain geometrical (centroid coordinates and contours) and topological data (adjacency and hierarchical layout) of floors and spaces of MIT buildings. Since our route graph generation process is based on the MIT floor plan XML representation, we describe its structure in some detail in this section.

The floor plan data corpus is organized to follow the natural hierarchy of a typical campus or a building space: it is embodied in a tree structure whose root is the base name of the entire corpus (‘`MIT`’). The tree model of XML data format is a natural encoding for such a structure. Each node in the tree has child nodes for the spaces that it physically contains.

For example, MIT Building 32 (the Stata Center) contains its nine floors as its children nodes, each of which, in turn, contains spaces that belong to it (e.g. offices, corridors and conference rooms) as its children in the tree.

7.1.1 Spaces

The basic unit of the floor plan representation is a *space*, a physical area within an extended indoor environment. Figure 7-1 shows a typical XML element for a space in the corpus. Each space element is attributed with a space name (e.g. "32-300S13") and a space type (e.g. "STAIR"), and has three child elements that are pertinent to its metric-topological properties: <contour> element describing its geometry, <portal> element describing its adjacency relationship to nearby spaces (see 7.1.2, and <triangle> element describing its triangulation structure (see 7.1.3).

The <contour> element describes the geometry of a space. Each space is represented as a simple polygon parameterized by its centroid coordinates (<centroid>) and the list of contour point coordinates that defines the sides of the polygon (<point>). If a space contains a round contour, it is approximated as a polyline. The bounding box surrounding the contour of the space is also given (<extent>).

Spaces are separated by either physical walls or wall-like structures, or by implicit

```
<space type="STAIR" name="32-300S13">
  <contour>
    <centroid x="710448.767146" y="496440.149751"/>
    <extent maxy="496449.009485" miny="496431.111303"
            maxx="710455.963341" minx="710441.497540"/>
    <point x="710441.497540" y="496446.049074"/>
    <point x="710444.834064" y="496447.569616"/>
    ...
    <point x="710441.497540" y="496446.049074"/>
  </contour>
  <portal type="stair" class="vertical" target="32-200S13" direction="DOWN"/>
  <portal type="implicit" class="horizontal" target="32-331CA.b">
    <edge maxparam="1.000000" minparam="0.000000" index="0"/>
  </portal>
  <triangle v1="0" v0="1" v2="2"/>
  <triangle v1="0" v0="2" v2="10"/>
  ...
</space>
```

Figure 7-1: XML representation of 32-300S13

boundaries between two spatially contiguous, yet functionally separate regions. In the latter cases, a space such as a large lecture room or a big open office space is further divided into multiple *subspaces*. The current version of the MIT floor plan XML assumes a flat hierarchical structure between spaces, and does not define a separate lower level for these subspaces. Therefore, a space node in the XML may correspond to either a physically isolated area surrounded by walls and doors or a subspace that is implicitly separated. In either case, the space node representation given in this section is universal, and each space node has its own geometric and topological information.

The basic semantic information of a space is encoded in the *space type* (`type` attribute in XML representation), which is defined by MIT Department of Facilities. This type classification provides useful information for finding user trajectories based on his/her activities, because certain classes of motion activities are strongly tied to specific subsets of space types. Section 7.3 further describes how this space type information can be grouped and used in activity-based trajectory matching.

Figure 7-2 shows a graphical presentation of `32-300S13` and surrounding area.

7.1.2 Portals

The adjacency relationship between two spaces is expressed by the use of *portals*. A portal connects two adjacent spaces, having a source and a “target” (destination) space. In the XML representation, a portal element is subsumed within the XML element of the source space, and has a `target` attribute specifying the destination space (see Fig. 7-1). Every portal is considered directional; if the corresponding connection is traversable from either direction, like most doors, it is expected to have a pair of portals, one for each source space, in the corpus.

Canonical examples of portals include: doors, implicit connection between two subspaces, and vertical transitions such as elevators and stairs. According to the characteristics of the transition, portals are classified into two classes: horizontal and vertical portals

A *horizontal* portal involves a connection between a pair of spaces that are at the same floor level, such as intra-floor transitions or inter-building transitions through a path on the same height. If a physical barrier (e.g. a door) exists on the connection, the corresponding portal is typed *explicit*; otherwise it is *implicit*. The latter case arises when a big open space is subdivided into smaller pieces because each smaller division (subspace) has a different function, or the original space is too large or complex-shaped to be kept as a single simple space.

This classification of horizontal portals implies certain characteristics about user motions. For example, detection of a door opening action implies that the user is located at an explicit portal, but the same is not true for implicit portals. Also, user movement is more restricted through explicit portals than implicit portals because passing an explicit portal means going through a doorway, which in general is a narrow passage way.

In the XML representation, every horizontal portal is parameterized by an index of the contour edge of the source location and position parameters describing the exact position of the portal on that edge (see Fig. 7-1). For explicit portals, the position parameter (`param` attribute) is a single dimensionless number in [0,1] describing the center position of the corresponding door. On the other than, implicit portals are parameterized by a (min, max) pair of dimensionless numbers (`minparam` and `maxparam` attributes), which represents the position and fraction of traversable segment along the corresponding contour line. Knowing the exact positions of portals is important for generating route graphs because they define the actual portion on a space contour that humans traverse when moving between adjacent places.

Inter-floor connections via stairs and elevators are classified as *vertical* portals. Other than a target space, every vertical portal is attributed with two properties that define the type

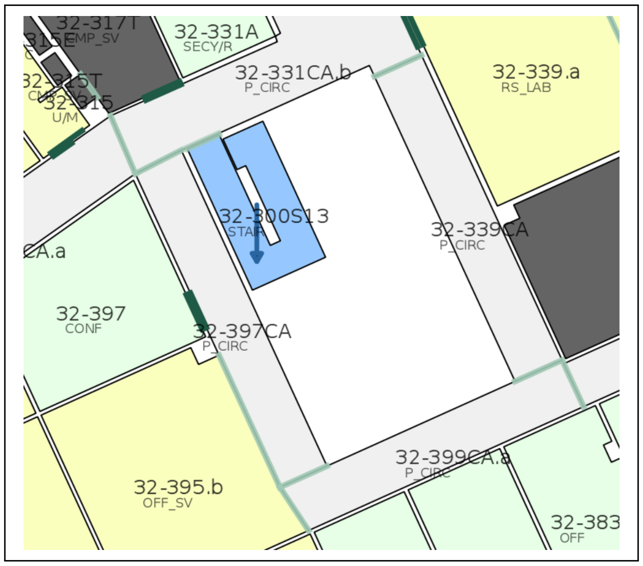


Figure 7-2: Graphical representation of the area around 32-300S13. Each space is given a name and a type, such as CONF (conference room), STAIR (stairs) or P_CIRC ("private circulation", or corridors), and color-coded by its category (see § 7.3). A circular corridor at the center is divided into four smaller sections, each of which has its own element in the XML document. Portals representing connection between adjacent spaces are shown in different colors; explicit portals (doors) are shown as a pair of thick dark-green markers (e.g. between 32-397 and 32-397CA); implicit portals (e.g. virtual boundary between 32-397CA and 32-331CA.b) are marked with light grey-green on the traversable portion of space contours; vertical portal representing stair transition from 32-300S13 toward the second floor is shown as a blue arrow.

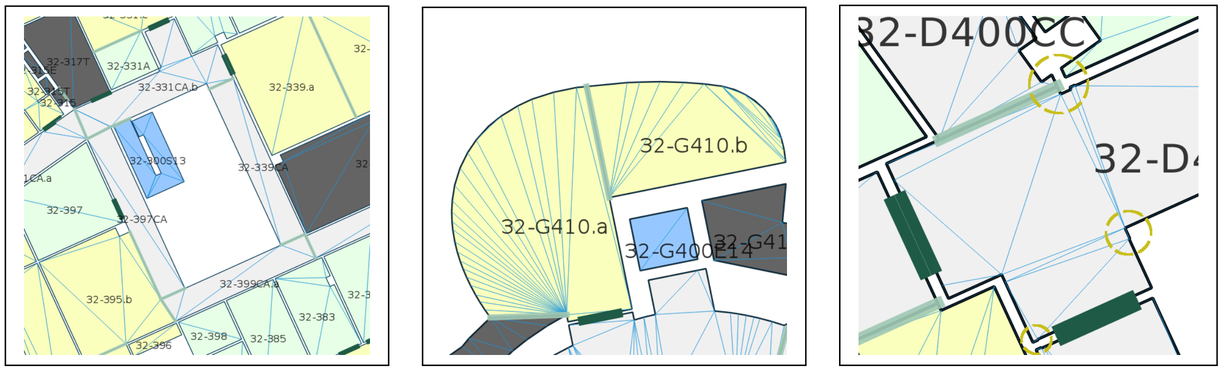
(‘`STAIR`’/‘`ELEV`’) and the direction (‘`UP`’/‘`DOWN`’) of the associated transition respectively. These properties are crucial for recovering user trajectory for multi-floor traversals as they provide highly informative constraints about user motion at the moment of vertical transitions.

7.1.3 Triangulation

Our route network generation process is based on prior work by Whiting et al. [73], which created walking routes through each space based on its triangulation. Specifically, it used a constrained Delaunay triangulation to approximate the medial axis of a space, by connecting adjacent midpoints of interior edges (edges of the triangles that are not a part of the space contour) [132]. This method was shown to work well for navigation in complex indoor environments such as the MIT Stata Center. In creating a route network (§ 7.2), we start with this scheme to create a raw route graph for every space in the map.

To support this process, a constrained Delaunay triangulation was computed for every space in the map. A set of triangles consisting the triangulation of the space was created for every space node in the floor plan XML (see Fig. 7-1). These triangles cover the entire space domain without overlapping.

A few triangulation examples are shown in Figure 7-3. For usual convex/non-convex polygonal spaces, the Delaunay triangulation captures the “shape” of the space well, providing the basis for route graph generation (Fig. 7-3a). On the other hand, if the space is round-shaped (Fig. 7-3b), or contains subtle, non-smooth segments, which are often artifacts



(a) (b) (c)

Figure 7-3: Triangulation examples. Constrained Delaunay triangles cover the space domain and provide the basis for route graph generation including non-convex-shaped spaces such as `32-331CA.b` (a). However, if a space contour includes (b) round edges or (c) subtle, non-smooth lines (yellow, dashed circles), the resulting triangulation may be unnecessarily complex.

resulted from imperfect processing of CAD drawings (Fig. 7-3c), the computed triangulation that is constrained to contain domain edges, can be unnecessarily complex. For motion-based matching, however, it is important to have a graph representation that can concisely describe actual human motions described in simple, abstract terms, such as “walks” and “turns”. If we were to connect the interior edges from the raw triangulation, the resulting route network would not be a proper representation of user paths in indoor environments, because human movement in indoor spaces is affected by the overall shape of the space that he/she is in (which the triangulation attempts to capture), not by the small details of the contour. A route network generation process must accommodate such intricacies.

7.2 Route Network Generation

Creating a route network from the MIT floor plan XML consists of a series of operations. The entire process is outlined below:

1. Source XML validation and correction
2. Per-space graph creation (§ 7.2.1)
3. Intra-floor graph merge and refinement (§ 7.2.2)
4. Inter-floor graph merge (§ 7.2.3)

First, the source XML documents, generated by Whiting and Battat [73, 130, 131], are automatically validated and corrected against common errors to ensure that general assumptions required for route network generation are met. Although the source XML documents are already high quality in general, checking the source file through the validation routine improves the correctness of a resulting route network. The validation steps include: removing any duplicate contour points, portals and triangles; checking location parameters of portals to ensure that they are logically correct; matching position, size and type of pairs of matching portals; and other minor corrections to improve the overall correctness of the source floor plan representation.

The validated XML floor plans may still leave places for further improvements and corrections. Spaces evolve over time either by renovation or placement of movable furniture. Further, the partitioning of open spaces using semi-permanent fixtures is not generally reflected in the floor plan. However, these partitions, made by room partitions or large fixtures, serve as “permanent” walls in practice and constrain human to walk around them. We made occasional manual corrections on the source XML documents to take these semi-permanent divisions into account as well as to deal with uncaught errors.

Using the validated XML floor plans, a route network is created in a bottom-up manner, from individual spaces to the entire corpus. The route graph generation algorithm takes the source XML documents and proceeds by creating a *per-space* route graph for every space. These per-space graphs are later assembled into a larger graph for each floor, which is then combined into a route graph for the entire building (the MIT Stata Center in the present work), which includes vertical transitions linking different floors. If a route network needs to be created across multiple buildings, these per-building graphs can be further combined in the same manner. The following sections describe this process in detail.

7.2.1 Per-Space Graph Creation

The following outlines the subgraph generation procedure for each space:

1. **Triangulation:** Compute the constrained Delaunay triangulation of the space domain.
2. **Medial Axis Approximation:** Connect midpoints of the interior edges along adjacent triangles to approximate medial axis.
3. **Simplification:** Simplify the generated raw graph:
   (a) Remove nodes that are too close to any contour line within a threshold.
   (b) Recursively merge two nodes that are too close within a threshold.
   (c) Simplify chains using Douglas-Peucker algorithm.
4. **Portal Node Generation:** Add one (explicit portal) or multiple (implicit portal) nodes per portal.
5. **Linking to Portals:** Make at most k connections from each portal node to the closest interior nodes.

As explained in Section 7.1.3, Steps 1 and 2 approximate the medial axis of a space domain. For simple shapes, this usually results in a satisfactory representation of walking routes through the input space. Many space contours in practice, however, have complex shapes with round edges and other subtleties. Raw graphs computed from the triangulation of such shapes may contain unnecessary artifacts and defects that do not correspond to actual walking behavior in indoor environments. Moreover, these unnecessary nodes in the route network increase the complexity of the route network, degrading time and space efficiency of map matching on the graph.

To remove these artifacts, we apply three cartographic generalization schemes on the raw graph (Step 3). First, nodes that are too close to any space contour line within a predefined threshold (set to 2 ft = 0.6 m for the Stata map) are removed (Step 3a). Second, interior nodes that are closely located with each other are recursively merged until all nodes are separated by some threshold, because they represent approximately the same position and do not need to be distinguished from each other for the map matching purpose (Step 3b). Third, linear chains with many intermediate nodes, such as a path on a curved bridge or a path along a round contour, are simplified using Douglas-Peucker algorithm [133], a well-known curve simplification method. The nodes removed in Step 3 usually correspond to the triangulation artifacts originating from a “skinny” triangle which has whose one edge which is a small part of a round corner or a small dent.

It should be noted that these simplification steps must preserve the connectivity of a target subgraph. For example, when Step 3a is applied on a narrow strip of corridor, all the interior node of the space triangulation may be located within the threshold from any of the contour lines. Simply applying Step 3a for this case will result in a disconnected, or even an empty graph. The simplification steps must not proceed in such cases to ensure the connectivity of the final graph.

Figures 7-4a and 7-4b illustrate the simplification steps. Spaces `32-331.a` and `32-331.b` have non-rectangular shapes. As a result, triangulation-based route graph generation introduces many redundant nodes that must be removed for efficient map matching (Fig. 7-4a). This simplification results in a simpler graph by merging or removing redundant nodes while ensuring connectivity (Fig. 7-4b).

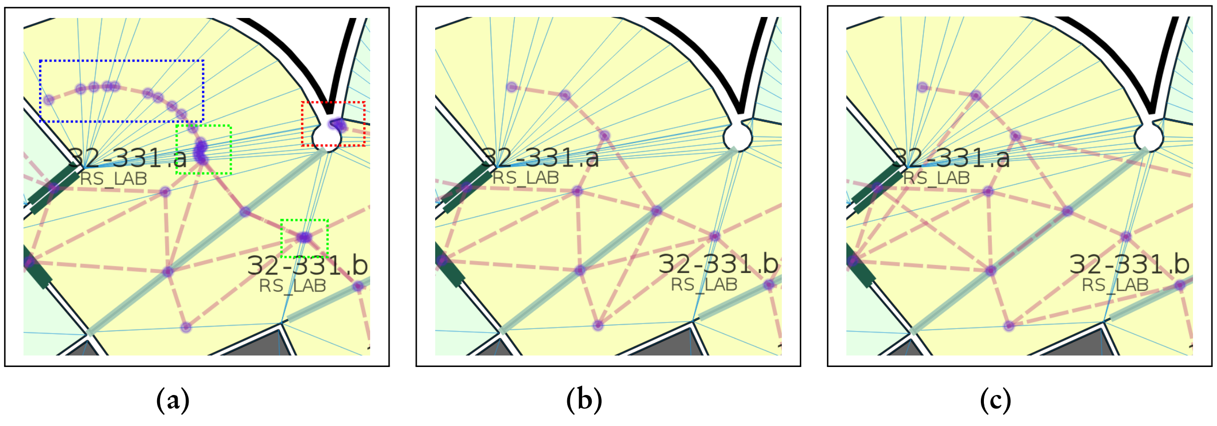


Figure 7-4: Simplification and refinement of route network. In (a), a raw graph generated from triangulation of the space may contain redundant nodes as a result of the complex triangulation structure. The graph contains nodes that are too close to a contour (red dashed box), nodes that are in close proximity with each other (green dashed box), and a chain of nodes induced from a round edge that is approximated by a polyline (blue dashed box). The simplification steps reduce the complexity of the graph shown in (b). In (c) further refinements are applied to connect missing intra- or inter-space straight-line paths while maintaining simplicity of the graph.

The last steps of the process are to generate nodes from portals and connect them to adjacent interior points (Steps 4 & 5). Whiting et al. [130] created one node per portal and connected it to the interior point that shares the same triangle. The resulting path segment around the portal often looks unnatural, because the chosen interior point is not always the closest one, depending on the triangulation structure of the space. While this scheme is valid for navigation purposes in [130], it is less suitable for map matching purpose, where the route graph must represent all the possible indoor traversals up to an approximation. Instead we opt to connect each portal node to at most *k* closest interior nodes, with *k* being set to 2 for the Stata corpus. Also, because an implicit portal often represents a wide, imaginary boundary between two subspaces in a bigger, open space, we place multiple portal nodes on every implicit portal, one for every fixed width (15 ft for the Stata corpus). We found that this scheme represents natural user movements around portals reasonably well.

7.2.2 Intra-Floor Graph Merge and Refinement

The per-space graphs constructed from triangulation for each space on the same floor are merged to form a bigger route network. The union of all spaces in the same floor is taken as the route graph representing that floor. Two graphs with adjacent spaces are joined by linking corresponding horizontal portal node pairs. Vertical portal links are considered in the next step (§ 7.2.3).

Rather than using the combined graph as it is, we refine the graph further to render a more faithful representation of indoor walking routes, by applying the following steps in order:

1. Remove nodes that belong to “inaccessible” spaces.

2. Connect pairs of nodes between which a feasible straight-line path is missing.

The first step improves the graph for indoor map matching use. A large building such as the Stata Center often has a small fraction of spaces that are not practically accessible to daily users, such as electronic/janitor closets or computer rooms. Those rooms are, in general, only accessible by specialists on specific occasions, and usually not of interest to normal walkers. Therefore, subgraphs whose parent space is classified as one of those “inaccessible” types (see Sec. 7.3 for the classification of spaces) are simply removed from the route network to achieve an efficient map matching process.

The second step mainly involves creating edges between pairs of nodes from different parent spaces. Because per-space graph generation in the previous section concerns only individual spaces, a simple concatenation of those graphs will not contain any edges that cross traversable space boundaries (i.e. implicit portals) and span across multiple adjacent spaces. This step ensures that there exists edges crossing adjacent spaces in the final route graph (Fig. 7-4c).

7.2.3 Inter-Floor Graph Merge

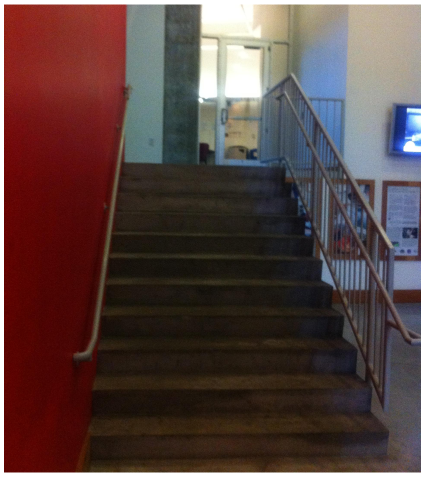
The last stage of the route network generation pipeline is combining route graphs from different floors. As in the intra-floor graph merge, each matching pair of vertical portals is connected to form a route network representing the entire building map. In contrast to the intra-floor merge, in which a matching pair of portal nodes is merged into a single node entity, the inter-floor graph merge simply adds an edge between a matching pair of vertical portals. This is because these matching pairs do not represent the same physical location unlike horizontal portal pairs. These edges have "vertical" attribute and used exclusively for matching with vertical user movements.

One exception to this rule is half-stairs (Fig. 7-5). Often, a building contains a mezzanine, or has irregular floor heights for some of its floors. In other cases, an inter-building connection between two adjacent buildings having different floor heights requires a small height adjustment using small staircases. These "half-stairs" have relatively short vertical heights and are often included exclusively in one of the two of the floor plans of the connecting floors. Therefore, from the viewpoint of floor plans, which is essentially a 2.5-D representation, they are not different from routes on a flat surface except that the corresponding space is classified as ``Stair''. However, the actual transition across half-stairs involves a short vertical movement, which can sometimes be detected using barometric and acceleration sensors.

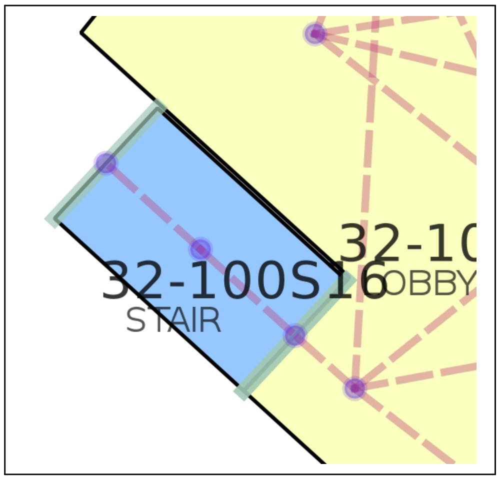
Therefore, in creating route networks, we give a dual representation for these half-stairs. In the graph generation, they are processed in the same manner as non-vertical spaces, in which per-space route graphs are generated and linked by merging matching horizontal portal nodes on the implicit boundary. Even when the matching pair of portals resides in two different floor plans, they are merged "horizontally" (by merging nodes instead of linking with a vertical edge.) However, interior edges for these half-stairs are set as a dual state ("horizontal/vertical"), which allows the map matching algorithm to match both short vertical transition observations and flat-surface walking on them.

7.3 Space Semantics

The route graph generation method described so far concerns only metric-topological aspects of human walking. The other missing component, *semantic constraints*, can be derived from the semantic information of the spaces. We augment the constructed route networks with the most basic form of semantic information, the type, which indicates usage (hence movement pattern therein) of the spaces.



(a)



(b)

Figure 7-5: A half-stair connecting the first and the second floor of the MIT Stata Center. This half-stair (a) is exclusively contained in the first floor floor plan (b).

The MIT Department of Facilities defines a set of fine-grained space types for all indoor spaces in MIT campus. Following that convention, the MIT Floor Plan XML data also records one of 91 space type codes for each space as an attribute for the `<space>` element (Whiting [130], Appendix B).

Since our work draws information from user motions and activities, knowing the usage type of spaces and encoding the information in the route graph facilitates inferring user path from motions. The type of a space is an important predictor about what kinds of motion behaviors are usually expected to be exhibited within that space. Hence, this knowledge can be incorporated into the map matching framework by writing down "compatibility functions" between space types and observed activities (Ch. 8).

Instead of using the fine-grained types, we categorized the 91 low-level types into six different *space categories*. The categorization is largely based on the "degree of movement" that is generally expected within that space. A space with Room category is usually a small office rooms where a person usually spends time sitting most of the time, whereas in a Corridor space, a person is usually walking. Common category lies between Room and Corridor categories in terms of movement patterns and is applicable for many common areas such as lobbies.

On the other hand, there are some special types of space that are usually inaccessible to normal persons, including electronic/janitor closets and server rooms. These spaces are marked as Inaccessible and the associated subgraphs are removed from the final route graph, as explained in Section 7.2.2.

Elevators and Stairs, categorized as Vertical, are of special interest because the movement through these spaces involves vertical motions. Because a typical building contains only a handful of these vertical portals, they serve as salient features for the map matching algorithm.

Every node in the route graph contains a reference to its parent space and shares the same space type. In addition, portal nodes have another attribute describing the class (horizontal vs. vertical) and the type (implicit vs. explicit, or elevator vs. stair) of the associated portal. This enables a map matching algorithm to utilize space category and portal information in map matching process.

Table 7.1 shows a list of low-level space types for each space category, from low-degree-of-motion to higher ones. The list is not exhaustive, but contains most space types found in the Stata Center corpus.

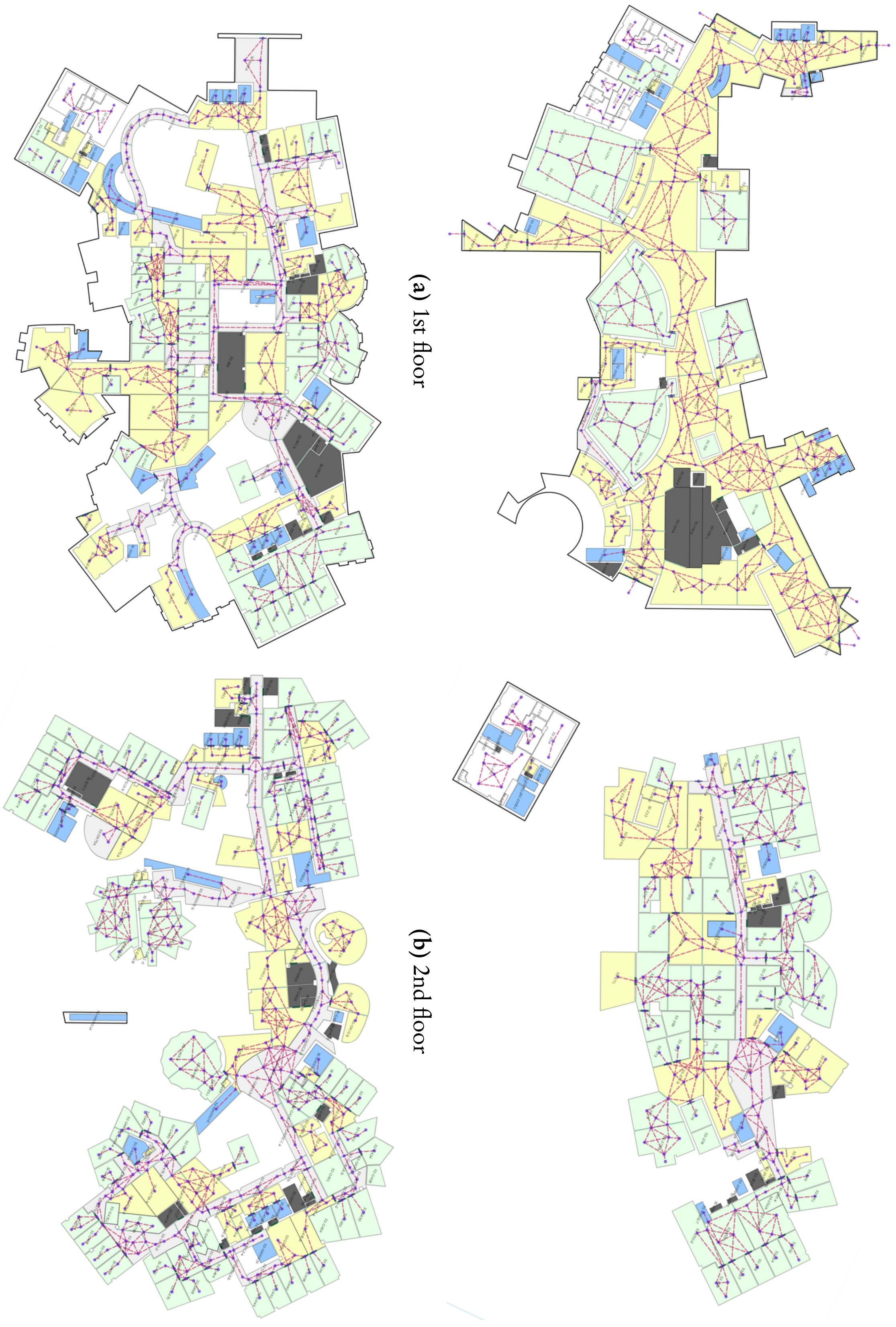
Category	Low-level types in this category	Degree of movement
Inaccessible	`ELEC` (electronic closet), `CMP SV` (computer service), `CMPUTR` (computer facility), `U/M` (utility/mechanical), `JAN CL` (janitor closet), `FOODSV` (food service), `SHAFT` (shaft space)	Inaccessible
Room	`RES LO` (research lab office), `OFF` (office), `CONF` (conference room), `STUDY` (study room), `SECY/R` (secretary/reception), `CLASS` (classroom), `LECT H` (lecture hall)	Mostly sitting
Common	`RS LAB` (research lab), `OFF SV` (office service), `CLA SV` (class service), `M LAV` (male lavatory), `F LAV` (female lavatory), `LOUNGE` (lounge), `LAB SP` (lab support shop), `LOBBY` (lobby), `FOOD` (food facility), `EXHIB` (exhibition area), `RECREA` (recreation facility)	Sitting/standing/short walking
Corridor	`LAB SV` (lab service), `P CIRC` (private circulation), `CORR` (corridor)	Mostly walking
Vertical	`ELEV` (elevator), `STAIR` (staircase)	Vertical transition
Other	Other unclassified types	Unknown

Table 7.1: Space categorization.

7.4 Conclusion

This chapter explained how we generated a route network, a graph representation of all possible user paths within an indoor environment. We gave a brief explanation of the MIT Floor Plan XML, and described a bottom-up procedure for the route network generation. First, small graphs were created for each space, and then, they were linked along horizontal and vertical connections to form a complete route network for the entire building. We also described how we classified the categories of spaces from the fine-grained space types available in floor plans. We show the final route networks in Figure 7-6.

Even though we generated route networks from legacy floor plans in a fully automatic manner, we anticipate that a high-quality route network can be created only with a small amount of manual labor with the aid of appropriate user interface. A "raw" route network can be first generated from floor plans, then a system administrator or end-users can add or revise the route network to improve its quality. As the end-goal of the method described in this thesis is to match user motions described in *human terms* on the prior map, the networks processed in this way might capture realistic human movements in indoor environment better than the automatically generated ones. We leave this hypothesis as future work.


(a) 1st floor

(b) 2nd floor

(c) 3rd floor

(d) 4th floor

Figure 7-6: Route graph examples of the MIT Stata Center. Space categorization: green – Room, yellow – Common, grey – Corridor, blue – Vertical, dark grey – Inaccessible. The vertical links corresponding elevator/stair transitions between floors are not shown.

Chapter 8

Motion Compatibility–Based Trajectory Matching for Indoor Localization

This chapter presents an indoor localization algorithm that recovers user path by matching user motion estimates on the prior map. The HMM-based matching method draws motion and map input generated in the previous chapters. We start by formulating the problem and laying out assumptions (§ 8.1), then describe a matching model that defines compatibility between motions and paths (§ 8.2). Next, we describe how the user trajectory can be recovered and model parameters can be estimated by HMM algorithms (§ 8.3). The experimental result from a test deployment demonstrates the performance of the method (§ 8.4).

8.1 Problem Statement

We formulate location discovery from motion traces as a sequence labeling problem. We model the user's indoor walking motion as a series of discrete motions consisting of straight-line walking segments, stops, turns, vertical movements, and actions to gain access (such as opening doors). Given a sequence of user motions, we compute the probabilistic path compatibility between trajectory hypotheses on the map and the hypothesized user path that gave rise to the input motion sequence (§ 8.1.1). Solving this problem amounts to labeling each input motion with a map location, while maximizing *compatibility*: the likelihood of the output trajectory given the input data. Given an input motion sequence, our method finds the most likely trajectory of location labels (§ 8.2).

We assume that each user carries a sensor-instrumented mobile device in his/her hand, held roughly level and pointed roughly forward. The device captures and time-stamps sensor data, which is then input to a classifier which produces as output a sequence of low-level motion types and durations.

We described our model of user motions in Section 6.1. We also assume that user paths have these properties:

1. **Smoothness**: The user maintains each motion for a certain characteristic duration, and does not change it too frequently.

2. **Continuity**: Consecutive motion segments agree at their endpoints (i.e. the user cannot "teleport" from one place to another).

3. **Monotonicity**: Users will tend not to travel a distance longer than necessary, or change floors more often than necessary, to reach any goal location.

We use these assumptions to precondition imperfect input motion sequences, as well as to formulate an efficient matching algorithm without redundancy. If a certain detection motion does not last for its normally expected duration (e.g. an elevator ride that lasts only two seconds), it is deemed erroneous, and either corrected or removed before matching (*smoothness*). When considering transitions from a certain place, the matching algorithm considers only nearby places for the next segment, resulting in physically plausible paths (*continuity*). Finally, the method excludes inefficient motion patterns (*monotonicity*).

The present thesis focuses on finding the location trajectory for a *single* user path. We anticipate that our method can be generalized to handle multiple paths from different users jointly, by taking interactions between paths into account. We discuss this possibility in Section 8.5.

8.1.1 Path Compatibility

A sequence of observed motions, as described in Section 6.1, implies a *motion path* (Figs. 8-1a & 8-1b). The motion path from imperfect observations is not identical to the original

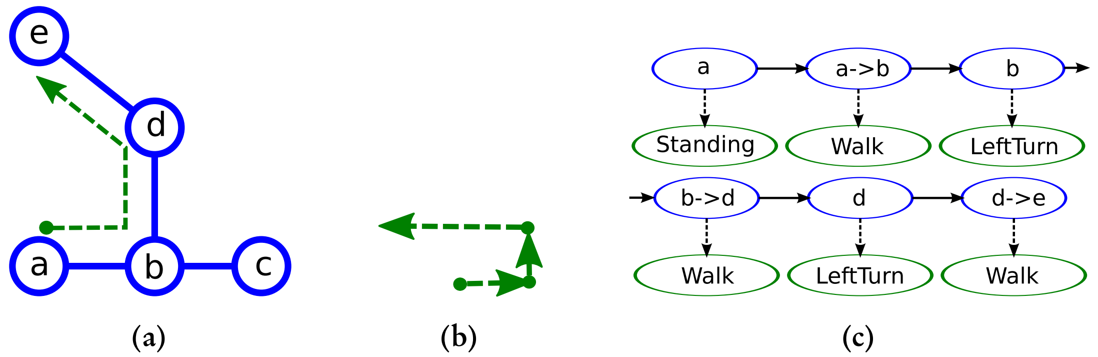


Figure 8-1: Path compatibility and matching: (a) path of a user moving a to e (green, dashed) on a route network (blue); (b) implied motion path from motion observations with imperfect length and angle estimates; (c) corresponding sequence labeling problem.

path from which motions were generated. Since motion estimates inferred by an automatic motion recognition algorithm are noisy, the properties attached to the motions can also be noisy and coarse-grained.

In this setting, the trajectory matching process from a sequence of motion observations can be thought of as finding the best (chain) subgraph in the route network whose *path compatibility* is maximal to the conceived path from motions. We say that a motion path is compatible with a subgraph if the path can be embedded into the subgraph in a way that observes (or weakly violates) the constraints inherent in the path and subgraph. That is, compatibility holds if each motion path segment can be assigned to some vertex or edge of the subgraph. For example, in Figure 8-1, the subgraph `a-b-d-e` in Figure 8-1a is compatible with the motion path of Figure 8-1b, because the path can be assigned to the subgraph with tolerable geometric distortion and without violating continuity. Clearly, the more components in the motion path, the more restricted its compatibility.

We analyze path compatibility at three levels: *geometric*, *topological*, or *semantic* compatibility. Geometric compatibility imposes metric constraints, such as length of walking segments and turn angles. Topological compatibility concerns about correspondence between space layouts and the continuity of motions. Semantic compatibility states that a certain class of motions can occur only in spaces with the matching type. These notions of compatibility are encoded in our trajectory matching model in Section 8.2.

8.2 Trajectory Matching Model

This section describes our matching model formulated from the elements in the previous section. The model encodes the notion of path compatibility (§ 8.1.1) as a form of sequence labeling problem, in which each unit motion is assigned to some node or edge, which represents user location and direction, of the route network (Fig. 8-1c).

8.2.1 Hidden Markov Models

We represent the stated sequence labeling problem as an instance of Hidden Markov Models (HMMs), a well-known probabilistic sequential model [134]. Let $x_t \in X$ denote the state representing the "location," and $y_t \in Y$ denote the input motion observation at time t, $1 \leq t \leq T$, where T is the length (the total number of unit motions) of the input motion sequence, with index 0 used for the (possibly unknown) initial state. Our goal is to assign "location labels", i.e. direction-parameterized nodes or edges in the route graph, to the state sequence $x_{1:T} = \{x_1, x_2, ..., x_T\}$, while maximizing path compatibility with the input motion sequence $y_{1:T} = \{y_1, y_2, ..., y_T\}$.

The HMM provides a scoring mechanism to determine the compatibility between X and Y by defining the following joint distribution for a sequence of T observations:

$$p(x_{0:T}, y_{1:T}) = p(x_0) \prod_{t=1}^{T} p(x_t|x_{t-1}) p(y_t|x_t) \tag{8.1}$$

where the model consists of three components: *transition probabilities* $p(x_t|x_{t-1})$; *emission probabilities* $p(y_t|x_t)$; and an *initial distribution* $p(x_0)$. With no knowledge about the initial location, $p(x_0)$ is a uniform distribution over states.

HMMs achieve their computational efficiency by limiting interaction between X and Y; i.e., the current state x_t depends only on the previous state x_{t-1} (Markovian state evolution), and the current observation y_t is conditionally independent of the other states given the current state x_t. These restrictions, as expressed in Equation (8.1), have important implications for the HMM transition and emission models: every user motion can be decomposed into a directional and a non-directional component, where directional properties (e.g. heading change by a turn) relate *two* states in time, while non-directional information (e.g. walk distance) defines compatibility between the associated motion and a *single* state. Hence, we rewrite Equation (8.1) to represent this factorization as an input-output HMM [135]:

$$\begin{aligned} & y \to (c, z) \\ p(x_{0:T}, y_{1:T}) &= p(x_{0:T}, c_{1:T}, z_{1:T}) \\ &= p(x_0) \prod_{t=1}^{T} p(x_t|x_{t-1}, c_t) p(z_t|x_t) \end{aligned} \tag{8.2}$$

where c is the “control” component governing state transition according to the observed direction, and z is the “measurement” component that determines single-state compatibility. Sections 8.2.4 to 8.2.6 describe how each type of motion y defines its own transition and emission model. The following section describes our representations of states X and input motions Y.

8.2.2 Input Model

An input is a sequence of motion descriptors, each of which models a unit action performed by the user while traveling indoors, as explained in Section 6.1. In this thesis, we use the following subset of natural motion descriptors: {Rest, Standing, Walking, Turning, Stair Walking, Elevator Ride, Door Open}. Other labels produced by the motion tagger (Ch. 6), including Sitting, Rising, and Button Press, were removed, since they add little information to the descriptors above.

The motion labeler associates a direction with Turning and vertical motions: {Left, Right, Left U-Turn, Right U-Turn} or {Up, Down} respectively. Also, every motion has a duration, from which some important geometric quantities can be estimated: walking distance or turning angle. We do not estimate other physical quantities from sensor measurements, because values from the low-cost sensors in off-the-shelf mobile devices would require careful calibration to be usable. Instead, we opt to infer physical values only from motion durations, by assuming constant walking speed and discretizing heading to eight (egocentric) cardinal directions. Even though the resulting estimates have limited accuracy, our matching method is flexible enough to handle the significant uncertainty arising from estimation error. Moreover, our framework does not exclude the use of direct measurements.

8.2.3 State Model

Our state model represents instantaneous location and heading at the completion of each unit motion. The state model is derived from the route network (Ch. 7). Straight-line walking or vertical transitions are matched to route network edges, while other actions are considered to occur at point locations, so are matched to route network nodes.

Heading

To represent instantaneous heading, we generate multiple states from each edge or node, one for each discrete heading (Fig. 8-2). For edges, because a user can walk from either direction, we derive *two* directional *edge-states* for each. Vertical connections between different floors are treated analogously, having two directional edge-states per connection. For nodes, because

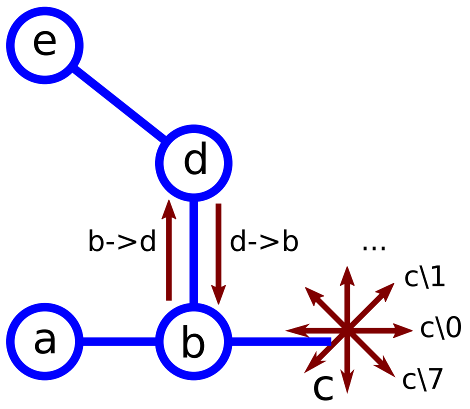


Figure 8-2: States for route network in Fig. 8-1a. Edge `b-d` gives two edge-states `b->d` and `d->b`, and node `c` gives 8 node-states `c\0` to `c\7`.

the user states after different rotation angles do not represent the same state in our problem (suppose the user starts walking after a rotation, then the next state will be dependent on the last heading angle), we quantize relative heading to produce eight different *node-states* for each graph node.

"Express" Edges

A straight-line walk that spans more than one edge in the route graph must be matched to a series of edges, instead of one. For example, in Figure 8-2, walking from node `a` to `c` matches a series of two edge-states, `a->b` and `b->c`. To deal with this problem, the state graph is augmented with *express* edges. An express edge is generated from a series of edges that together form a nearly-straight path (e.g. `a->c` in Fig. 8-2). Express edges are computed by performing breadth-first search from each network node, while testing if a series of edges can be combined into a single straight path via the Douglas-Peucker criterion: tangential deviation from the approximated line below a threshold [133]. We call the original non-express edges *local* edges, an analogy adopted from subway networks. When generating edge-states, both types of edges are treated in the same manner.

Properties

Each derived node- and edge-state inherits properties from the corresponding element in the route network and parent space. For instance, edge-states have a length property, the distance from one end to the other. Node-states are annotated with the type of the map object (e.g. room, door, stair, elevator) from which they arise. These annotations are used later during compatibility determination.

8.2.4 Matching Model: Horizontal Motions

This section describes the matching models for horizontal motions. The transition models, depending on the quantized angle input, have an identical form for all horizontal motions. We then give specifics for each class of horizontal motion.

Transition Model

The transition model $p(x_t|x_{t-1}, c_t)$ determines possible current states from the previous state, based on the directional component c_t that the observed motion indicates. Since a user path must be continuous, we allow transitions only to nearby states that can be reached in *one* action. A stationary user on a node-state may change direction while staying in the same location (e.g. standing turn), or start to walk along a connected edge in the route graph. A walking user on an edge-state, on the other hand, must arrive at a node state; at one turn would be required to reach a different edge. Hence, we design the transition probability to have non-zero mass $p(x_t|x_{t-1}, c_t) \neq 0$ (making the corresponding graph element *"reachable"*), only under the following conditions:

- x_{t-1} = a node-state on node A
  $\Rightarrow x_t \in$ { all node-states on A *or* edge-states starting from A }
- x_{t-1} = an edge-state from node A to node B
  $\Rightarrow x_t \in$ { all node-states on B }.

In the absence of directional information, every reachable next state is assigned the same transition probability. However, the HMM formulation requires the sum of transition probabilities from each state to be one ($\sum_{x_t} p(x_t|x_{t-1}) = 1$). Some algorithms distribute one unit of probability over outgoing transitions [96]. However, in complex indoor environments where the number of outgoing edges differs significantly from place to place, this formulation inappropriately assigns high transition probabilities to low-degree connections. We overcome this problem using the approach of VTrack [109], which assigns a global constant for each transition probability, and uses a dummy state to keep the summed probability equal to one.

Specifically, let ζ be the maximum out-degree in the route graph. Then the *base transition probability* for each state reachable from a given state is $1/\zeta$. We incorporate directional information from a motion observation by discounting the base probability by a Gaussian *angle compatibility function* of the difference between the observed turn angle from the motion, ψ_{c_t}, and the expected turn angle between two states, $\psi_{x_{t-1} \to x_t}$:

$$f_{\text{angle}}(x_t, x_{t-1}, c_t) = \exp\left\{ -\frac{(\psi_{c_t} - \psi_{x_{t-1} \to x_t})^2}{\sigma_a^2} \right\} \in (0, 1] \tag{8.3}$$

where σ_a is the *angle compatibility parameter* controlling the extent to which the matching algorithm allows angle mismatch (a higher value allows more matching freedom).

Summarizing, the transition probability for a motion with control component c_t (having turn angle ψ_{c_t}) is defined as:

$$p(x_t|x_{t-1}, c_t) = \begin{cases} \frac{1}{\zeta} \cdot f_{\text{angle}}(x_t, x_{t-1}, c_t) & x_t \text{ reachable from } x_{t-1} \\ 1 - \nu & x_t = \text{"dead-end" state} \\ 0 & \text{otherwise} \end{cases} \tag{8.4}$$

where $\nu \leq 1$ is the sum of transition probabilities to reachable next states:

$$\nu = \sum_{x_t:\text{reachable}} p(x_t|x_{t-1}, c_t). \tag{8.5}$$

As in VTrack, the remaining probability mass $1-\nu$ flows to the dead-end state, which has no further outgoing transitions and is incompatible with any further input motion. Any trajectory arriving at the dead-end state will have zero compatibility during decoding, and thus no chance of selection.

Emission Probability

The emission probability $p(z_t|x_t)$ is determined indirectly by setting the posterior probability of a state given the observation, $p(x_t|z_t)$, which is more natural to consider in our framework. For a fixed, known input motion z_t, specifying the compatibility functions between x_t and z_t in the posterior form $p(z_t|x_t)$ is equivalent to setting them in the original form $p(z_t|x_t)$ under proper normalization. For convenience, we refer to the posterior form as an "emission probability."

Rest and Standing

Whenever a non-turning motion, such as Rest, Standing, or (straight-line) Walking is observed, the turning angle observation is set to zero, $\psi_{c_t} = 0$. This effectively penalizes path hypotheses having a non-zero turn at that time with compatibility as determined by Equation (8.3).

Since neither Rest nor Standing involve any transitional motions, the emission probability (posterior probability) simply ensures that the current state must have type node-state, not edge-state:

$$p(x_t|z_t = \text{stationary}) \propto \begin{cases} 1 & x_t = \text{node-state} \\ 0 & x_t = \text{edge-state} \end{cases} \tag{8.6}$$

A more sophisticated model would distinguish ways of being stationary from context, exhibiting a stronger preference for certain space types when Sitting (vs. Standing) is observed.

Straight-line Walking

Like stationary motions, the transition probability for Walking follows the turn angle–based formula (Eq. 8.4) with zero observed turn angle $\psi_{c_t} = 0$.

For emission probability, unlike stationary motions, the walking motion must be matched on a local or express edge-state, not a node-state. We also incorporate walking distance compatibility here. We model human walking speed as a normal distribution centered at a constant μ_s with variance σ_s^2, i.e. as $\mathcal{N}(\mu_s, \sigma_s^2)$. The distribution of a "projected" walking distance for a straight-walk of duration Δt is then $\mathcal{N}(\mu_s \Delta t, \sigma_s^2 \Delta t^2)$. However, we observed that for short walking motions, duration estimates derived from the motion labeler's automatic segmentation were often inaccurate, causing underestimation of true walking duration. Thus using the estimate as is will produce an inappropriately small variance of the projected distance, leading to overly narrow regions of compatibility compared to the granularity of the route network map. Therefore, we set a minimum variance for the projected walking distance, $\sigma_d^{\min}$, to prevent the variance from collapsing. The *distance compatibility function* between a straight-line walk motion with duration Δt and an edge-state x_t is then defined as:

$$f_{\text{dist}}(x_t, z_t = \text{walk}) = \exp\left\{ -\frac{(l_{x_t} - \mu_s \Delta t)^2}{2\sigma_d^2} \right\} \tag{8.7}$$

$$\sigma_d = \max(\sigma_s \Delta t, \sigma_d^{\min}) \tag{8.8}$$

where l_{x_t} is the length of the edge-state x_t. We set $\sigma_d^{\min}$ to 3 ft (≈ 0.91 m) for our corpus to match the approximate granularity of the route network. Finally, the emission probability for a straight-line walk is:

$$p(x_t | z_t = \text{walk}) \propto \begin{cases} f_{\text{dist}}(x_t, z_t) & x_t = \text{edge-state} \\ 0 & x_t = \text{node-state.} \end{cases} \tag{8.9}$$

Turn

We model turns as point motions in which the user changes only heading direction while staying on one route graph node (and experiencing no net translation). Each turn observation is quantized to a multiple of $\pi/4$ to produce a value ψ_{c_t}. The transition and emission models are identical to those for stationary motions (Eqs. 8.4 & 8.6) except that each turn involves a non-zero heading change.

8.2.5 Matching Model: Vertical Motions

Vertical motions, including stair and elevator transitions, are associated with vertical edges that connect different floors in the route graph.

Elevators

Elevator ride motions provide three pieces of information to be exploited during matching: *type*, *direction*, and *length*. First, only vertical edges associated with an elevator can be matched to an elevator ride motion. Second, the ride direction (up or down) determines the direction of the edge-state to be matched in the transition model. Last, the number of floor transitions, which constrains the length of the vertical edge, is determined from the ride duration. The transition model for vertical motions (including elevator ride and stair walk) is:

$$p(x_t|x_{t-1},c_t) = \begin{cases} \frac{1}{\eta} & x_{t-1} \rightarrow x_t \text{ is the direction matching } c_t \\ 0 & \text{otherwise} \end{cases} \tag{8.10}$$

where η is the maximum out-degree among vertical transitions in the route graph (analogous to ζ for horizontal transitions). Essentially, the transition model determines next possible states based on the space type (vertical) and direction.

For the emission probability, the number of floor transitions or "vertical moving distance" is probabilistically determined by a compatibility function analogous to the distance compatibility function (Eq. 8.7). The number of floor transitions is estimated from the duration, and matched with the floor difference implied by each vertical edge, using a Gaussian compatibility function.

Stairwells

Like elevator rides, stair ascents and descents are matched by their associated type and properties. In principle, traversing a staircase could be treated in the same manner as level walking, if the detailed shape of all staircases in the corpus were known. For instance, if the precise shape of each spiral stair including the number of treads and spiral direction were known *a priori*, we could match every fine-grained sub-motion of a stair walk to an individual stair segment. However, our floor plans do not model stairwells with such precision, representing them instead simply as rooms with type "stair."

In light of this limitation, we instead summarize a series of fine-grained stair motions into a single, abstract stair motion starting at one end of a stairwell and ending at the other. Our system parameterizes stair transits by vertical direction (up / down), spiral direction (clockwise / counter-clockwise / straight), and duration as in elevator rides. (We manually annotated the spiral direction of each staircase in our corpus.) These properties are used in matching similarly to elevator ride motions; vertical direction is incorporated in the transition model, while length and spiral sense are used in the emission model. We model half-stairs differently from full-stairs, because the shorter transition through a half-stair can be easily missed by the motion labeler, causing a missed detection. We avoid this by allowing half-stairs to match straight-walk motions.

8.2.6 Matching Model: Special Cases

Door Open

We use detected door-open actions to constrain the user path when matching. Every door in the floor plan has an associated node in the route graph, from which states are generated. The detection of a door-open action indicates that the user *is likely to be* in one of these node-states.

However, we do not treat the door-open observation as a hard constraint; as are half-stairs, door-open actions are often confused with similar actions by the low-level motion labeler. Instead, the matching algorithm has the flexibility to violate the door constraint if necessary. To that end, we assign a small non-zero probability to non-door states even when a door-open action is detected:

$$p(x_t|z_t = \text{door open}) \propto \begin{cases} \alpha & x_t = \text{door node-state} \\ 1 & x_t = \text{non-door node-state} \\ 0 & x_t = \text{edge-state} \end{cases} \tag{8.11}$$

where $\alpha >> 1$ is a ratio encoding a preference for door node-states when a door-open action is detected. Larger values of α make the matcher less likely to violate door constraints.

Long Walks

Though we introduced express edges to handle straight-line walks spanning multiple edges, very long walks might not be handled well even with this mechanism. Suppose for example that the user has walked for a minute, traversing a 100-meter corridor. If the path is slightly curved, the motion classifier would fail to detect the curvature. This single straight-line Walking observation would have to be matched to a long series of successive edges in the route network. Automatically creating express edges for such long edges would introduce many additional edges to the state graph, increasing the computational complexity of matching. A simple solution to this problem is to split lengthy walks into smaller intervals, each of which can then be matched to an express or local edge. Our method uses a threshold of 15 seconds (about 20 meters of walking).

8.3 Algorithms

In Section 8.2, we modeled the trajectory matching problem from a user motion sequence using the HMM formulation. In this section, we present algorithms for recovering user trajectories and estimating model parameters from unannotated user motion data.

8.3.1 Matching Algorithm

To decode a user trajectory in the HMM model with Equation (8.2.1), we use standard methods: forward-filtering to compute the distribution of the most recent state x_T (as in conventional particle filters for localization), forward-backwards algorithm to compute "smoothed" distributions of x_t in the past ($1 \leq t \leq T$), and the Viterbi algorithm to find the "most likely" trajectory $x_{1:T}$ [134].

In this thesis, we use the Viterbi algorithm to compute the most likely continuous trajectory. Smoothed estimates computed by the forward-backward algorithm are similar to those from Viterbi, but are not guaranteed to be spatially continuous. Unlike conventional particle filters that update only the last position estimate upon each new input, the most-likely-trajectory approach updates the entire path. Also, the Viterbi algorithm can easily be modified to yield the k-best state sequences instead of the single best, along with matching scores. The score gap between the best sequence and the rest can be used to gauge uncertainty in the current estimates.

In practice, the algorithm should be implemented to exploit sparsity of our state model rooted on route networks. Because a user path without a significant rest period must be continuous, the number of possible transitions from a specific location is physically bounded by the maximum out-degree in the state graph. With N states and an input sequence of length T, sparsity yields a time complexity of $O(NT)$ for the matching algorithm, rather than $O(N^2T)$ for non-sparse models. The complexity of computing transition and emission models is $O(N)$ per unit motion.

8.3.2 Parameter Learning

The matching models require specification of a few parameters. These include include physical parameters, such as walking speed constant μ_s (Eq. 8.7), and other parameters that determine association strength between motions and user paths (σ_a, σ_d, and α in Eqs. 8.3, 8.7 and 8.11, resp.). Some of these parameters have a physical interpretation, which provide a basis for setting a value in the model. For example, from prior knowledge of average human walking speed, we can determine μ_s empirically.

In this section, however, we show how to determine these parameters automatically from *unannotated* data – motion sequences with unknown location labels – using a variant of the Expectation-Maximization (EM) algorithm. This process can be used to learn individually-tuned parameters for a dataset from a single user, or alternatively to learn good parameters for a dataset captured from the motions of many users.

Intuitively, for a given motion dataset, some parameter settings will produce better trajectories than others in terms of explanatory power. For instance, the best path found by setting the walking speed to 5 km/h (average human walking speed) is more "plausible" than the path found by setting it to 1 km/h. Given n data sequences $\{y^i_{1:T_i} | 1 \le i \le n\}$, we search for the parameters Θ^* and paths ${x^i_{1:T_i}}^*$ that maximize joint probability of the HMMs:

$$\Theta^*, {x^1_{1:T_1}}^*, \ldots, {x^n_{1:T_n}}^* \leftarrow \underset{\Theta, x^1_{1:T_1}, \ldots, x^n_{1:T_n}}{\operatorname{argmax}} \prod_{i=1}^{n} p(x^i_{1:T}, y^i_{1:T}; \Theta). \tag{8.12}$$

This optimization problem is solved by the hard-EM algorithm (also known as Viterbi training or segmental K-means) [136]. It finds the best parameters (along with the best paths) in coordinate ascent manner. First, the parameters Θ are fixed, and the best paths are found using the Viterbi algorithm. Next, the estimated paths $x^i_{1:T_i}$ are treated as ground truth, and new parameters are estimated. This process is iterated until convergence:

1. Initial parameters: $\Theta(0)$.
2. Step $\tau = 1, 2, \ldots$, repeat until convergence ($\Theta_{\tau-1} \approx \Theta_\tau$):
   (a) Given $\Theta(\tau - 1)$, find paths $x^i_{1:T}(\tau)$ using the Viterbi algorithm;
   (b) Estimate new parameters $\Theta(\tau)$ from inputs and decoded labels at τ: $\{x^i_{1:T}(\tau), y^i_{1:T} | 1 \le i \le n\}$.

Since the optimization problem in Equation (8.12) is not convex in general, and hard-EM does not guarantee identification of the global optimum, care must be taken in using this approach. For example, if the corpus does not impose strong constraints enabling easy parameter learning (e.g., if much of the corpus consists of large open rooms and wide corridors), the process may settle on non-optimal parameters. Therefore, it is helpful to guide the learning process by providing a good set of initial parameters and by limiting the range of each parameter, using common-sense knowledge of human motions.

8.4 Evaluation

8.4.1 Experimentation Methodology

We collected 30 single- and multi-floor motion sequences over seven days. Of these, 21 traces contained at least one vertical transition (elevator or stair transit). The total length of the dataset is 58 minutes, with an average sequence length of about 2 minutes. Our deployment area, the MIT Stata Center, was a nine-story, 67,000 m^2 office and lab complex. We used four floors of the building, from the ground level up to the fourth floor. Coverage included most corridors and many office and lab spaces.

We developed data logging software running on a Nokia N900 mobile phone for motion sensing. The N900 host connects to an external sensing device, the Nokia Sensorbox, containing five consumer-grade MEMS sensors in a $4 \times 4 \times 1$ cm package: a tri-axial accelerometer, tri-axial gyroscope, barometer, thermometer, and tri-axial magnetometer. Our low-level motion classifier performed satisfactorily on uncalibrated sensor data (i.e. with unit conversion factors left as factory defaults). The data logging module connected to the Sensorbox via Bluetooth and continuously sampled time-stamped sensor data, which were then offloaded to a desktop computer for offline analysis. Our evaluation pipeline first predicted user motions from the sensor data in leave-one-trajectory-out manner, then matched each labeled motion sequence to a trajectory.

We determined ground truth trajectories by annotating video of the user and background recorded by the host smartphone. We used a custom GUI to indicate the type and duration of each low-level motion, and its location on the map; locations and motion types were interpolated at fine grain from these annotations.

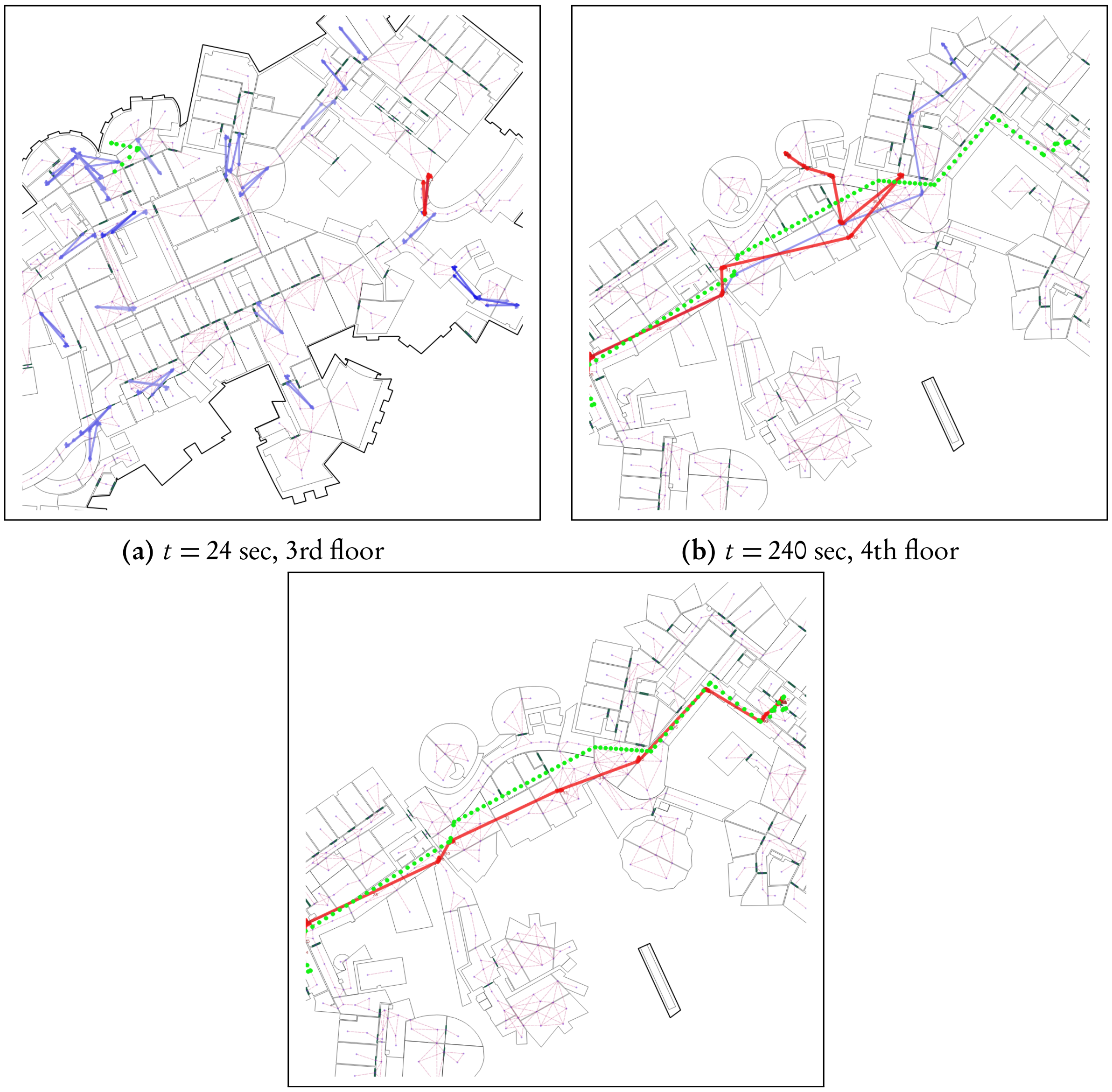
8.4.2 Matching Examples

We illustrate the algorithm's behavior using matching results from two test traces. In the first example (Figs. 8-3 & 8-4), the user started from an office on the third floor, transited to the fourth floor via elevator, walked across the fourth floor, and returned to the starting office via a different elevator. No information was known about the user's starting location.

Figure 8-3 shows a few snapshots of the trace over time. We compute trajectory error at time t (Fig. 8-4) by computing pointwise error at each frame (3 Hz) until t, then taking the median as the representative trajectory error at t. (The stair-step nature of the error plots arises from the fact that the algorithm computes a new trajectory estimate whenever a new unit motion becomes available.)

Initially, not enough information is available to constrain the user path; the user has made only one significant turn, classified as `Right U-Turn`. Hence, there were many plausible embeddings of this short motion sequence throughout the corpus (Fig. 8-3a). The method started to find the correct path before the user took an elevator to the fourth floor (“`ELEV:3->4`” in Fig. 8-4). The user next walked faster than average, resulting in a large disparity between the true and estimated walking distance. This made the algorithm select an incorrect, alternative path, which better matched the distance estimate while sacrificing angle compatibility (Fig. 8-3b). The correct path was eventually recovered as the user took the elevator, providing strong evidence of location. This example shows how the algorithm recovers the correct path history from a transient failure by incorporating stronger constraints.

We show another example in which the algorithm matches the true path after 45 seconds of walking (Fig. 8-5). Though the user walked on a single floor, the true path was sufficiently distinctive to enable rapid convergence, yielding the true trajectory after a few turns.



(a) $t = 24$ sec, 3rd floor

(b) $t = 240$ sec, 4th floor

(c) $t = 269$ sec, 4th floor

Figure 8-3: Matching example (green: ground truth; red: best path found; blue: other path candidates): (a) after only one right turn, many plausible paths; (b) before elevator transit, matching drifted due to noisy walking distance estimate; (c) after elevator transit, matching algorithm corrected the entire path.

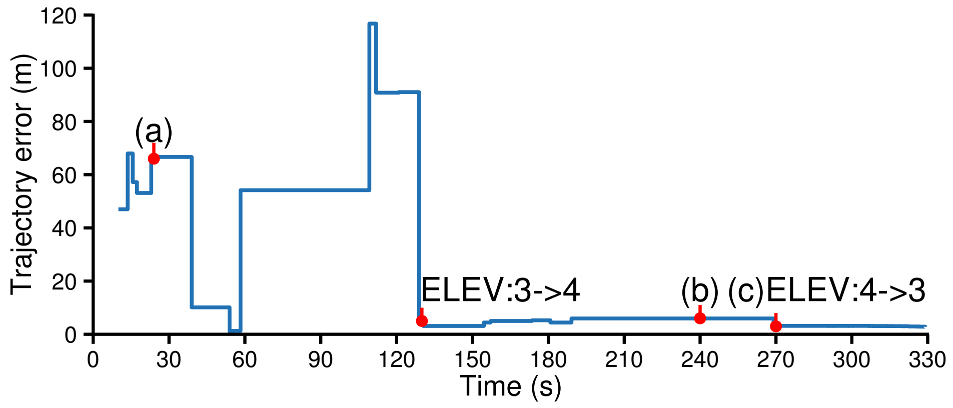


Figure 8-4: Trajectory error for the first example trace (§ 8.4.2).

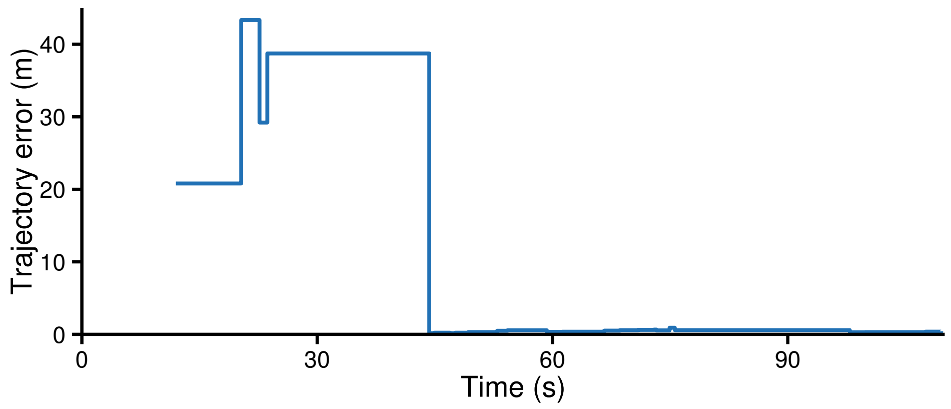
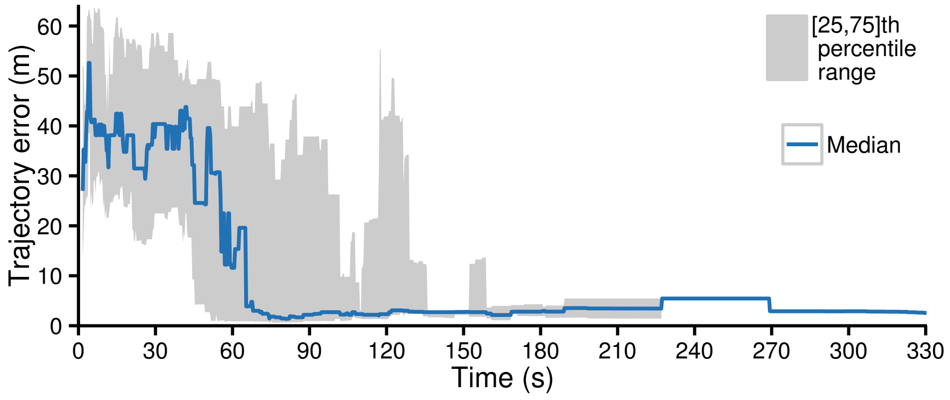


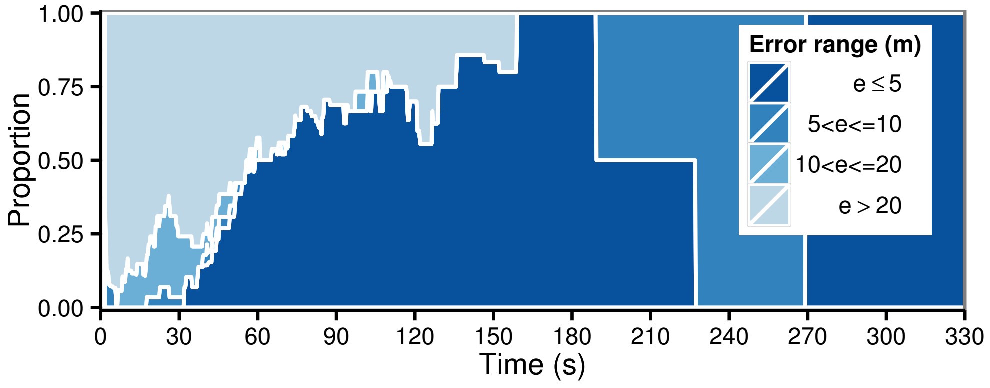
Figure 8-5: Trajectory error for the second example.

8.4.3 Ensemble Trajectory Error

We evaluated the matcher's performance by computing trajectory error over all sequences in the dataset. Figure 8-6a shows the progression of trajectory error statistics (median and interquartile range) over time; Figure 8-6b depicts the distribution of results over different error ranges. The x-axis includes time spent for all motions, not only for walking. The matching algorithm is typically highly uncertain until it reaches a "tipping point" at which enough information is available to constrain the user path with high accuracy. For more than half of the test traces, the algorithm started to match an input motion sequence on the correct path within about one minute, and for almost all traces within about two minutes, similar to the results of Rai et al. [17]. Note that the exact tipping point is also a function of the characteristics of the corpus and the underlying motion labeling algorithm, rather than solely of the matching algorithm.



(a) Trajectory error (median and interquartile range) over time.



(b) Trajectory error distribution over time.

Figure 8-6: Overall trajectory error

8.4.4 Salient Features

Certain motion classes, when detected, provide strong constraints on the user path. Navigation-related events such as vertical transitions or door-open actions limit the user path to a small number of candidates, as there are only a few ways in which such motions can occur within the provided route graph.

We confirm this intuition by measuring the trajectory error after the user experienced a certain number of each kind of *salient* motion: turn, vertical transitions (elevators / stairs), and door-open actions. For each trajectory, we computed the total number of (ground-truth) salient motions that had been experienced in that trajectory as of each sensor frame. Note that the low-level motion classifier sometimes fails to detect these features; missed actions will not contribute to matching.

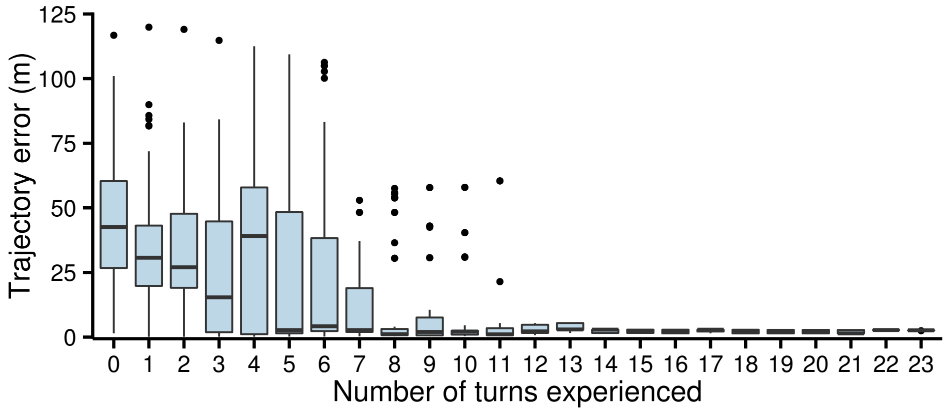
The results (Figs. 8-7) confirm that distinctive motions facilitate matching by providing additional constraints on possible user paths. However, different motions contribute by different degrees; constraints provided by vertical transitions are strongest among the motion classes tested (Fig. 8-7b). This is because vertical motions were detected very reliably by our classifier, and stairwells and elevators occur rarely in our building (as in most buildings) compared to ordinary rooms. On the other hand, door-opens, which were often missed or misclassified, were a less reliable source of information (Fig. 8-7c). Turns were weakest among three, because the angle compatibility function (Eq. 8.3) allows relatively broad freedom in matching, enabling many plausible paths on the map until a certain number of distinct turns were accumulated (Fig. 8-7a).

8.4.5 Prior Information

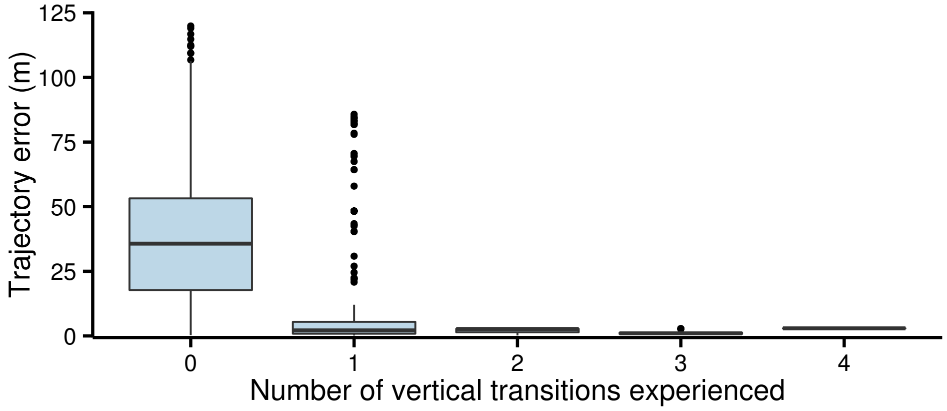
Prior information other than motion sequences could be valuable for matching. If available, such information could potentially make the matching process converge faster to the true user path. Our trajectory matching formulation admits various types of prior information to be incorporated in the model, e.g. by setting an initial state distribution or by pre-filtering candidate locations.

We assessed the method under a hypothetical scenario in which the starting floor (but not the precise location) is known beforehand. Floor information may be available from a different sensor (e.g. WiFi, or barometer [102]), or by taking user attributes (e.g. office number or calendar contents) into account. To test this scenario, we initialized the state distribution $p(x_0)$ uniformly over the known starting floor.

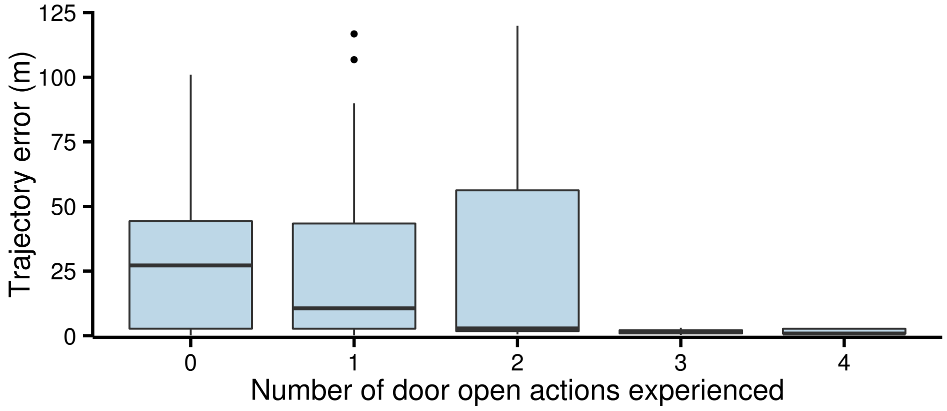
Figure 8-8 compares error with and without starting floor information. With the initial floor known, the bootstrapping time required to yield median trajectory error below 5 meters was 51 seconds, 14 seconds faster than the 65 seconds required when the initial floor was unknown. This improvement was due to better matching for previously ambiguous test



(a) Turns



(b) Vertical transitions



(c) Door-open actions

Figure 8-7: Error decreases when salient motions are observed.

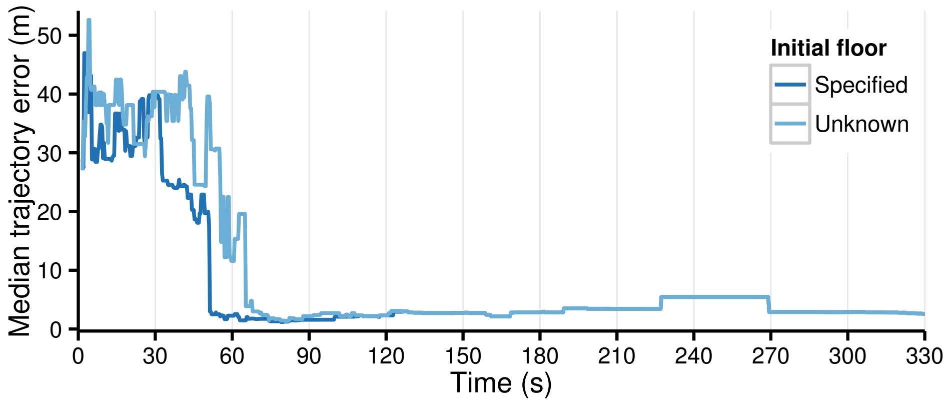


Figure 8-8: Trajectory error is lower when the initial floor is specified.

paths that did not traverse multiple floors. We expect starting floor information to be particularly salient when the user motion trajectory has no or few vertical transitions and different floors have spatially similar layouts.

8.4.6 Computation Time

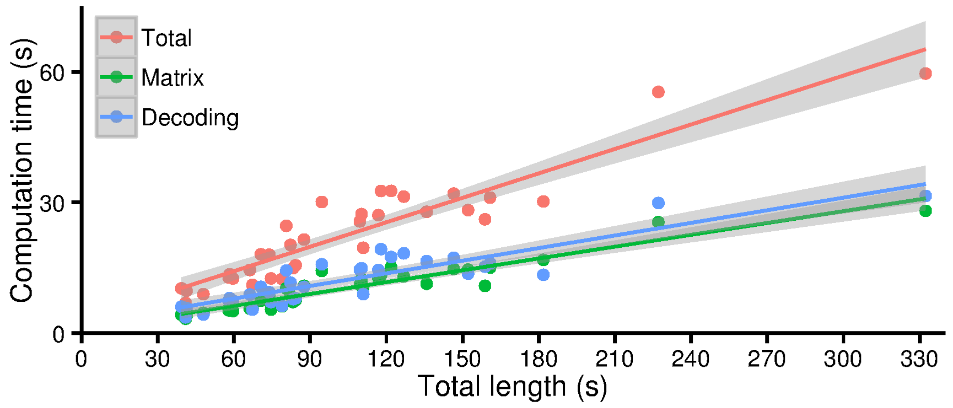
We studied the computational complexity of our matching algorithm. As explained in Section 8.3.1, the Viterbi decoding algorithm with a sparse model has time complexity of $O(NT)$ rather than $O(N^2T)$, for N states and T time steps. The complexity of transition and emission matrix computation is also $O(NT)$, or $O(N)$ per unit motion. Online, computation of the model matrices is amortized; each matrix is computed only once and stored for reuse in later executions.

Our unoptimized Python/SciPy implementation exhibited update times linear in the input sequence length (Fig. 8-9a). Its running time scaled linearly with the number of states (Fig. 8-9b), which we varied by selecting subsets of the map.

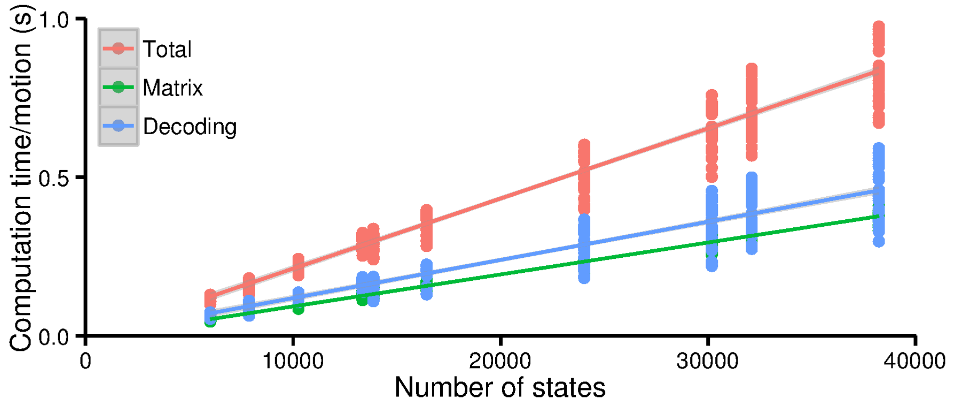
8.5 Discussion

Our trajectory matching model defines a compatibility measure between descriptive motion sequences and paths, enabling trajectory recovery to be cast as a sequence labeling and matching problem.

Our method has several limitations. Essentially, its performance is determined by the balance between the expressive power of the matching model and the uncertainty of the prior map and input motions. When this uncertainty outweighs the model expressiveness, the method may take an unreasonably long time, or even fail, to find the correct path.



(a) Computation time as a function of input length.



(b) CPU time per unit motion increases with the number of states.

Figure 8-9: Time for Matrix and Viterbi (“Decoding”) computations.

In our experiments, we had an instance in which no turns were detected by the motion classification algorithm when the user walked over a gently curving bridge. The bridge did not contain any sharp turns (which caused the motion classifier to fail to detect turns) but the net difference in heading before and after crossing the bridge was nearly 90 degrees. In that case, the motion over the bridge was recovered as a single, long straight-line walk, which the map matching algorithm tried to match to long corridors rather than to the true bridge path. Our input model of egocentric observations did not handle this case properly. We anticipate that this problem could be alleviated with additional sensor data (such as compass heading).

Also, as demonstrated in Section 8.4.5, ambiguous environments such as wide corridors may cause the method to fail, especially when the input motion has short duration and no prior information is available. In many of these instances, the amount of information that can be extracted from the input is low compared to the high uncertainty that the ambiguous environment implies. In our case, the ground floor of the corpus had a wide "student street" and large, open spaces. Motion traces without strong evidence (e.g. vertical transitions) tend to produce uncertain trajectory estimates.

In future work, we anticipate that the current model can be expanded in a variety of ways to become more expressive while handling uncertain cases more gracefully. For example, the matching model can exploit, or even infer, a user's activity pattern in indoor environments. Because people tend to traverse and use spaces in similar fashion, there exists a natural association or "activity map" between activities and space types. While at present we use such cues only for vertical motions, associating them with stairs or elevators, this mapping can be generalized to handle other motion classes e.g., sitting in an office, or walking in a corridor, by defining emission probabilities that capture the corresponding associations. Conversely, the trajectory matching algorithm could be used to learn (unknown) associations from user data by bootstrapping, as it can proceed given only geometric and topological compatibilities, without requiring or using semantic information. This learned association could then be used to facilitate later matching processes, creating a closed loop between trajectory estimation and activity map learning.

Another way to extend the model would be through joint estimation of multiple paths from a single user or multiple users. At a single-user level, each user tends to repeat some paths, exhibiting a few distinctive motion patterns, or often returns to a few select locations. At a multi-user level, each user will encounter or accompany others, having either brief or substantial overlap with other users' paths. We anticipate that such information can be collected from multiple paths, with the aid of device proximity sensors (e.g. ad hoc WiFi or Bluetooth), and can be incorporated into the model as a form of "second-order" compatibility.

8.6 Conclusion

We described a fine-grained indoor localization and trajectory estimation method based on the notion of path compatibility: probabilistic agreement between user motions and a prior map encoded as a route graph. We defined user motion traces as a sequence of navigation-related actions, which were automatically estimated by a classification algorithm from proprioceptive sensor data. The method assumes route networks (derived automatically) from legacy floor plans of the deployment area. With these inputs, we defined an HMM-based matching model that encodes the compatibility between motions and locations, and used the model to decode the most likely trajectory given an input motion sequence. We also showed how to learn the model parameters from unannotated data.

Testing on user data from four floors of a deployment area demonstrates that our method can recover the user's location trajectory to within a few meters using only proprioceptive sensor data from a commodity mobile device. However, our method fails where the uncertainty of the user motion and/or prior map is large. As future work, we anticipate that the current model can be extended to handle more general classes of motions and to incorporate more expressive semantic associations between motions and spaces.

Chapter 9

Conclusion

This thesis presented two approaches to the indoor localization problem. Both approaches are based on the notion of place and motion signatures, and they model how a user moves and interacts with localization systems to discover his/her location in an indoor environment.

In the first approach, the organic indoor location discovery, we presented representations of and algorithmic solutions to the fundamental requirement of organic localization systems — an ability to bootstrap fingerprint database quickly from the beginning, while excluding faulty user inputs. We presented a Voronoi diagram–based approach to prompt new user inputs, and a clustering-based approach to filter erroneous user binds. In addition, we analyzed the device diversity problem, which arises in practical organic location systems in which users use diverse devices, and proposed design guidelines for organic localization algorithms.

In the second approach, we explicitly modeled how humans describe their movements in indoor environments, and presented a probabilistic matching model that formalizes such intuition. We presented a CRF-based motion classifier, which produces accurate motion estimates from uncalibrated, low-cost MEMS sensor inputs. We also automatically generated route networks, a graph representation of possible walking paths in an indoor space, from our deployment building, the MIT Stata Center. We showed that our motion compatibility–based trajectory matching algorithm recovered user trajectories from motion estimates on the route network. We demonstrated the performance of the method using real user data.

Through our work in this thesis, we placed an emphasis on a previously uncaptured, yet important element of the system: *users*. We believe that future localization methods, especially those for complex indoor environments, must explicitly model human users as a central part of the localization system.

Even with the attempts in this thesis, however, indoor localization is still an unsolved, challenging problem. The scope and complexity of indoor spaces are incomparably greater than those of outdoor spaces; indoor spaces are often irregularly shaped and rapidly changing over time to meet users' needs; walls, furniture, electronic devices and human movements often obscure faithful sensing by mobile sensors, preventing positioning methods from recovering accurate location estimates. These problems can be solved only by expanding our understanding of how humans, not only devices, move in and interact with indoor environments.

9.1 Future Work

We conclude the thesis by outlining a few possible future directions extending the presented work. In general, we anticipate that indoor localization can be significantly improved by actively utilizing interaction between end-users. The methods presented in the thesis can be expanded to incorporate diverse sources of such "second-order" information.

9.1.1 Rich Sensor Maps

In the first part of the thesis, we used WiFi fingerprints as place signatures of indoor spaces. However, sensor features that can be associated with physical locations are not limited to WiFi. We envision that place signatures can be enriched by various types of sensors, including:

- Local magnetic disturbance
- Local inertial sensing data (e.g., from floor or ground profile)
- Barometric pressure profile
- User mobility pattern
- Frequently visited places

Place signatures enriched with various sources of fingerprints, or *rich sensor maps*, can be used to find positions of different types of devices with heterogeneous sets of sensors. Moreover, they can be voluntarily collected and shared across end-users in the same manner as we have shown in Part I for WiFi fingerprints.

9.1.2 Encounters and Significant Places

Comparison of two or more WiFi signal measurements gives physical proximity information between those measurements [72]. Similarly, Bluetooth scans also give proximity information to other Bluetooth devices in the area. Such *encounter* information can constrain two or more individual trajectories that would otherwise be unrelated. Thus, it can be used for a joint inference of multiple user paths, expanding the horizon of motion compatibility–based indoor localization that we presented in Part II.

The notion of *significant places* brings important high-level information that constrains the possible locations of the user. Most people spend a majority of their time at a few places such as their home and in the workplace. The number of places that an ordinary person normally visits is usually small. This implies that if a recurring pattern of sensor and WiFi signals is found, it is likely that it matches one of the user's significant places. Automatic recognition of such patterns, as demonstrated in previous work [4], can reveal a great amount of positioning information about where the user was likely to be during that specific interval of time, especially when combined with auxiliary information such as directories or user calendars.

Bibliography

[1] Jun-geun Park, Ben Charrow, Dorothy Curtis, Jonathan Battat, Einat Minkov, Jamey Hicks, Seth Teller, and Jonathan Ledlie. Growing an Organic Indoor Location System. In *Proc. 8th Ann. Int'l Conf. on Mobile Systems, Applications, and Services (MobiSys '10)*, pages 271–284, June 2010.

[2] Jun-geun Park, Dorothy Curtis, Seth Teller, and Jonathan Ledlie. Implications of Device Diversity for Organic Localization. In *Proc. 30th IEEE Int'l Conf. on Computer Communications (INFOCOM '11)*, pages 3182–3190, April 2011.

[3] Donald J. Patterson, Lin Liao, Dieter Fox, and Henry Kautz. Inferring High-Level Behavior from Low-Level Sensors. In *Proc. 5th Int'l Conf. on Ubiquitous Computing (UbiComp '03)*, 2003.

[4] Jeffrey Hightower, Sunny Consolvo, Anthony LaMarca, Ian Smith, and Jeff Hughes. Learning and Recognizing the Places We Go. In *Proc. 7th Int'l Conf. on Ubiquitous Computing (UbiComp '05)*, 2005.

[5] B. Hoffmann-Wellenhof, J. Collins, and H. Lichtenegger. *Global Positioning System: Theory and Practice*. Springer-Verlag, 1993.

[6] R. Want, A. Hopper, V. Falcão, and J. Gibbons. The Active Badge Location System. *ACM Transactions on Information Systems*, 10(1):91–102, 1992.

[7] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The Cricket Location-Support System. In *Proc. 6th ACM Ann. Int'l Conf. on Mobile Computing and Networking (MobiCom '00)*, pages 32–43, 2000.

[8] Nissanka B. Priyantha, Allen K. L. Miu, Hari Balakrishnan, and Seth Teller. The Cricket Compass for Context-Aware Mobile Applications. In *Proc. 7th ACM Ann. Int'l Conf. on Mobile Computing and Networking (MobiCom '01)*, pages 1–14, 2001.

[9] Vladimir Bychkovsky, Bret Hull, Allen Miu, Hari Balakrishnan, and Samuel Madden. A Measurement Study of Vehicular Internet Access Using in situ Wi-Fi Networks. In *Proc. 12th ACM Ann. Int'l Conf. on Mobile Computing and Networking (MobiCom '06)*, pages 50–61. ACM, 2006.

[10] Paramvir Bahl and Venkata N. Padmanabhan. RADAR: An In-Building RF-Based User Location and Tracking System. In *Proc. 19th IEEE Int'l Conf. on Computer Communications (INFOCOM '00)*, pages 775–784, March 2000.

[11] Andreas Haeberlen, Eliot Flannery, Andrew M. Ladd, Algis Rudys, Dan S. Wallach, and Lydia E. Kavraki. Practical Robust Localization over Large-Scale 802.11 Wireless Networks. In *Proc. 10th ACM Ann. Int'l Conf. on Mobile Computing and Networking (MobiCom '04)*, pages 70–84, September 2004.

[12] Moustafa Youssef and Ashok Agrawala. The Horus WLAN Location Determination System. In *Proc. 3rd Ann. Int'l Conf. on Mobile Systems, Applications, and Services (MobiSys '05)*, 2005.

[13] Theodore S. Rappaport. *Wireless Communications: Principles and Practice*. Prentice Hall, 2nd edition, 2002.

[14] Seth Teller, Jonathan Battat, Ben Charrow, Dorothy Curtis, Russel Ryan, Jonathan Ledlie, and Jamey Hicks. Organic Indoor Location Discovery. Technical Report MIT-CSAIL-TR-2008-075, MIT CSAIL, 2008.

[15] Lasse Klingbeil and Tim Wark. A Wireless Sensor Network for Real-time Indoor Localisation and Motion Monitoring. In *Proc. 7th Int'l Conf. on Information Processing in Sensor Networks (IPSN '08)*, pages 39–50, 2008.

[16] Oliver Woodman and Robert Harle. Pedestrian Localisation for Indoor Environments. In *Proc. 10th Int'l Conf. on Ubiquitous Computing (UbiComp '08)*, pages 114–123. ACM, 2008.

[17] Anshul Rai, Krishna Kant Chintalapudi, Venkata N. Padmanabhan, and Rijurekha Sen. Zee: Zero-Effort Crowdsourcing for Indoor Localization. In *Proc. 18th Ann. Int'l Conf on Mobile Computing and Networking (MobiCom'12)*, pages 293–304. ACM, 2012.

[18] Patrick Robertson, Michael Angermann, Bernhard Krach, and Mohammed Khider. Inertial Systems Based Joint Mapping and Positioning for Pedestrian Navigation. In *Proc. 22nd Int'l Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS '09)*, 2009.

[19] Martin Azizyan, Ionu Constandache, and Romit Roy Choudhury. SurroundSense: Mobile Phone Localization via Ambience Fingerprinting. In *Proc. 15th ACM Ann. Int'l Conf. on Mobile Computing and Networking (MobiCom '09)*, pages 261–272, 2009.

[20] Philipp Bolliger. Redpin – Adaptive, Zero-Configuration Indoor Localization through User Collaboration. In *Proc. 1st ACM Int'l Workshop on Mobile Entity Localization and Tracking in GPS-less Environments (MELT '08)*, pages 55–60, September 2008.

[21] William G. Griswold, Patricia Shanahan, Steven W. Brown, Robert Boyer, Matt Ratto, R. Benjamin Shapiro, and Tan Minh. ActiveCampus: Experiments in Community-Oriented Ubiquitous Computing. *Computer*, 37(10):73–81, 2004.

[22] C. Jernigan, C. Bayley, J. Lin, and C. Wright. Locale. `http://people.csail.mit.edu/hal/mobile-apps-spring-08/`, 2008.

[23] Anthony LaMarca, Yatin Chawathe, Sunny Consolvo, Jeffrey Hightower, Ian E. Smith, James Scott, Timothy Sohn, James Howard, Jeff Hughes, Fred Potter, Jason Tabert, Pauline Powledge, Gaetano Borriello, and Bill N. Schilit. Place Lab: Device Positioning Using Radio Beacons in the Wild. In *Proc. 3rd Int'l Conf. on Pervasive Computing (PERVASIVE '05)*, pages 116–133, May 2005.

[24] Ekahau. `http://www.ekahau.com`.

[25] Ezekiel S. Bhasker, Steven W. Brown, and William G. Griswold. Employing User Feedback for Fast, Accurate, Low-Maintenance Geolocationing. In *Proc. 2nd IEEE Int'l Conf. on Pervasive Computing and Communications (PerCom '04)*, pages 111–120, March 2004.

[26] Andrew Barry, Benjamin Fischer, and Mark Chang. A Long-Duration Study of User-Trained 802.11 Localization. In *Proc. 2nd Int'l Workshop on Mobile Entity Localization and Tracking in GPS-less Environments (MELT '09)*, pages 197–212, September 2009.

[27] Jeffrey Hightower, Gaetano Borriello, and Roy Want. SpotON: An Indoor 3D Location Sensing Technology Based on RF Signal Strength. Technical Report UW-CSE-2000-02-02, University of Washington, 2000.

[28] Asim Smailagic and David Kogan. Location Sensing and Privacy in a Context-Aware Computing Environment. *IEEE Wireless Communications*, 9(5):10–17, 2002.

[29] Hyuk Lim, Lu-Chuan Kung, Jennifer C. Hou, and Haiyun Luo. Zero-Configuration, Robust Indoor Localization: Theory and Experimentation. In *Proc. 25th IEEE Int'l Conf. on Computer Communications (INFOCOM '06)*, pages 1–12, 2006.

[30] Paramvir Bahl, Venkata N. Padmanabhan, and Anand Balachandran. Enhancements to the RADAR User Location and Tracking System. Technical Report MSR-TR-2000-12, Microsoft Research, 2000.

[31] Milos N. Borenovic and Aleksandar M. Neskovic. Comparative Analysis of RSSI, SNR, and Noise Level Parameters Applicability for WLAN Positioning Purposes. In *Proc. IEEE Region 8 EUROCON*, 2009.

[32] Mauro Brunato and Roberto Battiti. Statistical learning theory for location fingerprinting in wireless LANs. *Computer Networks*, 47(6):825–845, 2005.

[33] Sebiatian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.

[34] Moustafa A. Youssef, Ashok Agrawala, and A. Udaya Shankar. WLAN Location Determination via Clustering and Probability Distributions. In *Proc. 1st IEEE Int'l Conf. on Pervasive Computing and Communications (PerCom '03)*. University of Maryland, College Park, 2003.

[35] Moustafa Youssef and Ashok Agrawala. Handling Samples Correlation in the Horus System. In *Proc. 23rd IEEE Int'l Conf. on Computer Communications (INFOCOM '04)*, volume 2, 2004.

[36] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Monte Carlo Localization for Mobile Robots. In *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA '99)*, 1999.

[37] Joydeep Biswas and Manuela Veloso. WiFi Localization and Navigation for Autonomous Indoor Mobile Robots. In *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA '10)*, 2010.

[38] David Madigan, Eiman Elnahrawy, and Richard P. Martin. Bayesian Indoor Positioning Systems. In *Proc. 23rd IEEE Int'l Conf. on Computer Communications (INFOCOM '04)*, volume 2, pages 1217–1227, 2004.

[39] Julia Letchner, Dieter Fox, and Anthony LaMarca. Large-Scale Localization from Wireless Signal Strength. In *Proc. Nat'l Conf. on Artificial Intelligence (AAAI-05)*, page 15, 2005.

[40] Nirupama Bulusu, John Heidemann, and Deborah Estrin. GPS-less Low-Cost Outdoor Localization for Very Small Devices. *IEEE Personal Communications*, 7:28–34, 2000.

[41] Tian He, Chengdu Huang, Brian M. Blum, John A. Stankovic, and Tarek Abdelzaher. Range-Free Localization Schemes for Large Scale Sensor Networks. In *Proc. 9th ACM Ann. Int'l Conf. on Mobile Computing and Networking (MobiCom '03)*, pages 81–95, 2003.

[42] Slobodan N. Simic and Shankar Sastry. Distributed Localization in Wireless Ad Hoc Networks. Technical report, UC Berkeley, 2001.

[43] Suprakash Datta, Chris Klinowski, Masoomeh Rudafshani, and Shaker Khaleque. Distributed Localization in Static and Mobile Sensor Networks. In *Proc. IEEE Int'l Conf. on Wireless and Mobile Computing, Networking and Communications (WiMob '06)*, pages 69–76, 2006.

[44] Jeffrey Hightower and Gaetano Borriello. Location Sensing Techniques. Technical Report UW-CSE-01-07-01, University of Washington, 2001.

[45] Chris Savarese, Jan Rabaey, and Koen Langendoen. Robust Positioning Algorithms for Distributed Ad-Hoc Wireless Sensor Networks. In *Proc. USENIX Ann. Technical Conference (USENIX '02)*, pages 317–328, 2002.

[46] Dragos Niculescu and Badri Nath. DV Based Positioning in Ad Hoc Networks. *Telecommunication Systems*, 22:267–280, 2003.

[47] Andreas Savvides, Chih-Chieh Han, and Mani B. Strivastava. Dynamic Fine-Grained Localization in Ad-Hoc Networks of Sensors. In *Proc. 7th ACM Ann. Int'l Conf. on Mobile Computing and Networking (MobiCom '01)*, pages 166–179. ACM Press New York, NY, USA, 2001.

[48] Juan Liu, Ying Zhyang, and Feng Zhao. Robust Distributed Node Localization with Error Management. In *Proc. 7th ACM Int'l Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '06)*, pages 250 – 261, 2006.

[49] Srdjan Čapkun, Maher Hamdi, and Jean-Pierre Hubaux. GPS-free Positioning in Mobile Ad hoc Networks. *Cluster Computing*, 5(2):157–167, 2002.

[50] David Moore, John Leonard, Daniela Rus, and Seth Teller. Robust Distributed Network Localization with Noisy Range Measurements. In *Proc. 2nd ACM Int'l Conf. on Embedded Networked Sensor Systems (SenSys '04)*, pages 50–61, New York, NY, USA, 2004. ACM Press.

[51] Jun-geun Park, Erik Demaine, and Seth Teller. Moving-Baseline Localization. In *Proc. 7th Int'l Conf. on Information Processing in Sensor Networks (IPSN '08)*, pages 15–26, 2008.

[52] Nissanka B. Priyantha, Hari Balakrishnan, Erik D. Demaine, and Seth Teller. Mobile-Assisted Localization in Wireless Sensor Networks. In *Proc. 24th IEEE Int'l Conf. on Computer Communications (INFOCOM '05)*, volume 1, pages 172–183, March 2005.

[53] Lance Doherty, Kristofer S. J. Pister, and Laurent El Ghaoui. Convex Position Estimation in Wireless Sensor Networks. In *Proc. 20th IEEE Int'l Conf. on Computer Communications (INFOCOM '01)*, volume 3, pages 1655–1663, 2001.

[54] Pratik Biswas, Tzu-Chen Liang, Kim-Chuan Toh, Ta-Chung Wang, and Yinyu Ye. Semidefinite Programming Approaches for Sensor Network Localization With Noisy Distance Measurements. *IEEE Transactions on Automation Science and Engineering*, 3(4):360–371, 2006.

[55] Andreas Savvides, Wendy Garber, Sachin Adlakha, Randolph Moses, and Mani B. Srivastava. On the Error Characteristics of Multihop Node Localization in Ad-Hoc Sensor Networks. In *Proc. 2nd Int'l Workshop on Information Processing in Sensor Networks (IPSN '03)*, pages 317–332. Springer, April 2003.

[56] Randolph L. Moses, Dushyanth Krishnamurthy, and Robert M. Patterson. A Self-Localization Method for Wireless Sensor Networks. *EURASIP Journal on Applied Singal Processing*, 4:348–358, 2003.

[57] Jose A. Costa, Neal Patwari, and Alfred O. Hero. Distributed Weighted-Multidimensional Scaling for Node Localization in Sensor Networks. *ACM Transactions on Sensor Networks*, 2(1):39–64, 2006.

[58] Yi Shang, Wheeler Ruml, Ying Zhang, and Markus P.J. Fromherz. Localization from Mere Connectivity. In *Proc. 4th ACM Int'l Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '03)*, pages 201–212. ACM Press New York, NY, USA, 2003.

[59] Yi Shang and Wheeler Ruml. Improved MDS-Based Localization. In *Proc. 23rd IEEE Int'l Conf. on Computer Communications (INFOCOM '04)*, volume 4, pages 2640–2651, 2004.

[60] Alexander T. Ihler, John W. Fisher III, Randolph L. Moses, and Alan S. Willsky. Nonparametric Belief Propagation for Self-Calibration in Sensor Networks. In *Proc. 3rd Int'l Symposium on Information Processing in Sensor Networks (IPSN '04)*, pages 225–233, 2004.

[61] Aline Baggio and Koen Langendoen. Monte-Carlo Localization for Mobile Wireless Sensor Networks. In *Proc. 2nd Int'l Conf. on Mobile Ad-hoc and Sensor Networks (MSN '06)*, pages 317–328. Springer, 2006.

[62] Lingxuan Hu and David Evans. Localization for Mobile Sensor Networks. In *Proc. 10th ACM Ann. Int'l Conf. on Mobile Computing and Networking (MobiCom '04)*, pages 45–57. ACM Press New York, NY, USA, 2004.

[63] John Krumm and John Platt. Minimizing Calibration Effort for an Indoor 802.11 Device Location Measurement System. Technical Report MSR-TR-2003-82, Microsoft Research, November 2003.

[64] Anthony LaMarca, Jeff Hightower, Ian Smith, and Sunny Consolvo. Self-Mapping in 802.11 Location Systems. In *Proc. 7th Int'l Conf. on Ubiquitous Computing (UbiComp '05)*, 2005.

[65] Jeffrey Junfeng Pan, Qiang Yang, and Sinno Jialin Pan. Online Co-Localization in Indoor Wireless Networks by Dimensionality Reduction. In *Proc. 22nd AAAI Conf. on Artificial Intelligence (AAAI-07)*, 2007.

[66] Krishna Chintalapudi, Anand Padmanabha Iyer, and Venkata N. Padmanabhan. Indoor Localization Without the Pain. In *Proc. 16th ACM Ann. Int'l Conf. on Mobile Computing and Networking (MobiCom '10)*, 2010.

[67] Joseph Huang, David Millman, Morgan Quigley, David Stavens, Sebastian Thrun, and Alok Aggarwal. Efficient, Generalized Indoor WiFi GraphSLAM. In *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA '11)*, pages 1038–1043, 2011.

[68] Sebastian Thrun and Michael Montemerlo. The Graph SLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures. *The International Journal of Robotics Research*, 25(5-6):403, 2006.

[69] Brian Ferris, Dieter Fox, and Neil Lawrence. WiFi-SLAM using Gaussian Process Latent Variable Models. In *Proc. 20th Int'l Joint Conf. on Artificial Intelligence (IJCAI-07)*, pages 2480–2485, 2007.

[70] Neil D. Lawrence. Gaussian Process Latent Variable Models for Visualisation of High Dimensional Data. In *Advances in Neural Information Processing Systems (NIPS-04)*, volume 16, pages 329–336. of, 2004.

[71] Skyhook wireless. `http://www.skyhookwireless.com/`.

[72] John Krumm and Ken Hinckley. The NearMe Wireless Proximity Server. In *Proc. 6th Int'l Conf. on Ubiquitous Computing (UbiComp '04)*, pages 283–300, September 2004.

[73] Emily Whiting, Jonathan Battat, and Seth Teller. Topology of Urban Environments. In *Computer-Aided Architectural Design Futures (CAADFutures)*, pages 114–128, 2007.

[74] James Cussens. Bayes and Pseudo-Bayes Estimates of Conditional Probabilities and Their Reliability. In *Proceedings of the European Conference on Machine Learning (ECML-93)*, pages 136–152, 1993.

[75] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer, Third edition, 1997.

[76] Nils Ole Tippenhauer, Kasper Bonne Rasmussen, Crhistina Pöpper, and Srdjan Čapkun. Attacks on public WLAN-based positioning systems. In *Proc. 7th Ann. Int'l Conf. on Mobile Systems, Applications, and Services (MobiSys '09)*, pages 29–40. ACM New York, NY, USA, 2009.

[77] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Inc., 1988.

[78] Nattapong Swangmuang and Prashant Krishnamurthy. Location Fingerprint Analyses Toward Efficient Indoor Positioning. In *Proc. 6th IEEE Int'l Conf. on Pervasive Computing and Communications (PerCom '08)*, pages 100–109, 2008.

[79] Yingying Chen, Konstantinos Kleisouris, Xiaoyan Li, Wade Trappe, and Richard P. Martin. The Robustness of Localization Algorithms to Signal Strength Attacks: A Comparative Study. In *Proc. 3rd IEEE Comm. Soc. Conf. on Sensor and Ad Hoc Communications and Networks (SECON '06)*, volume 4026, pages 546–563. Springer, 2006.

[80] Leonid Portnoy, Eleazar Eskin, and Salvatore J. Stolfo. Intrusion Detection with Unlabeled Data Using Clustering. In *Proc. ACM CSS Workshop on Data Mining Applied to Security*, 2001.

[81] Nattapong Swangmuang and Prashant Krishnamurthy. On Clustering RSS Fingerprints for Improving Scalability of Performance Prediction of Indoor Positioning Systems. In *Proc. 1st ACM Int'l Workshop on Mobile Entity Localization and Tracking in GPS-less Environments (MELT '08)*, pages 61–66. ACM New York, NY, USA, 2008.

[82] Hendrik Lemelson, Mikkel Baun Kjærgaard, Rene Hansen, and Thomas King. Error Estimation for Indoor 802.11 Location Fingerprinting. In *Proc. 4th Int'l Symposium on Location and Context Awareness (LoCA '09)*, page 138. Springer, 2009.

[83] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classfication*. Wiley-Interscience, 2nd edition, 2000.

[84] John Krumm and Eric Horvitz. LOCADIO: Inferring Motion and Location from Wi-Fi Signal Strengths. In *Proc. 1st Ann. Int'l Conf. on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04)*, pages 4–14, August 2004.

[85] Philipp Bolliger, Kurt Partridge, Maurice Chu, and Marc Langheinrich. Improving Location Fingerprinting through Motion Detection and Asynchronous Interval Labeling. In *Proc. 4th Int'l Symposium on Location and Context Awareness (LoCA '09)*, pages 37–51, May 2009.

[86] Arvin Wen Tsui, Yu-Hsiang Chuang, and Maurice Hao-Hua Chu. Unsupervised Learning for Solving RSS Hardware Variance Problem in WiFi Localization. *Mobile Networks and Applications*, 14(5):677–691, 2009.

[87] Mortaza S. Bargh and Robert de Groote. Indoor Localization Based on Response Rate of Bluetooth Inquiries. In *Proc. 1st ACM Int'l Workshop on Mobile Entity Localization and Tracking in GPS-less Environments (MELT '08)*, pages 49–54. ACM, September 2008.

[88] Yu-Chung Cheng, Yatin Chawathe, Anthony LaMarca, and John Krumm. Accuracy Characterization for Metropolitan-scale Wi-Fi Localization. In *Proc. 3rd Ann. Int'l Conf. on Mobile Systems, Applications, and Services (MobiSys '05)*, pages 233–245, June 2005.

[89] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley and Sons, 2006.

[90] Mike Y. Chen, Timothy Sohn, Dmitri Chmelev, Dirk Haehnel, Jeffrey Hightower, Jeff Hughes, Anthony LaMarca, Fred Potter, Ian Smith, and Alex Varshavsky. Practical Metropolitan-Scale Positioning for GSM Phones. In *Proc. 8th Int'l Conf. on Ubiquitous Computing (UbiComp '06)*, pages 225–242, September 2006.

[91] Kamol Kaemarungsi. Distribution of WLAN Received Signal Strength Indication for Indoor Location Determination. In *Proc. Int'l Symposium on Wireless Pervasive Computing (ISWPC '06)*, January 2006.

[92] Mikkel Baun Kjærgaard. Automatic Mitigation of Sensor Variations for Signal Strength Based Location Systems. In *Proc. 2nd Int'l Workshop on Location and Context Awareness (LoCA '06)*, pages 30–47, May 2006.

[93] Fangfang Dong, Yiqiang Chen, Junfa Liu, Qiong Ning, and Songmei Piao. A Calibration-Free Localization Solution for Handling Signal Strength Variance. In *Proc. 2nd Int'l Workshop on Mobile Entity Localization and Tracking in GPS-less Environments (MELT '09)*, pages 79–90, September 2009.

[94] Chenshu Wu, Zheng Yang, Yunhao Liu, and Wei Xi. WILL: Wireless Indoor Localization Without Site Survey. In *Proc. 31st IEEE Int'l Conf. on Computer Communications (INFOCOM '12)*, pages 64–72. IEEE, 2012.

[95] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proc. 18th Int'l Conf. on Machine Learning (ICML-01)*, pages 282–289, 2001.

[96] Lin Liao, Deieter Fox, Jeffrey Hightower, Henry Kautz, and Dirk Schulz. Voronoi Tracking: Location Estimation Using Sparse and Noisy Sensor Data. In *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS '03)*. Citeseer, 2003.

[97] J.O. Wallgrün. Autonomous Construction of Hierarchical Voronoi-Based Route Graph Representations. *Spatial Cognition IV. Reasoning, Action, and Interaction*, pages 413–433, 2005.

[98] Michael Peter, Norbert Haala, Markus Schenk, and Tim Otto. Indoor Navigation and Modeling Using Photographed Evacuation Plans and MEMS IMU. In *Special joint symposium of ISPRS Technical Commission IV & AutoCarto*, November 2010.

[99] C. Ascher, C. Kessler, M. Wankerl, and GF Trommer. Dual IMU Indoor Navigation with particle filter based map-matching on a smartphone. In *Proc. Int'l Conf. on Indoor Positioning and Indoor Navigation (IPIN '10)*, pages 1–5. IEEE, 2010.

[100] Fan Li, Chunshui Zhao, Guanzhong Ding, Jian Gong, Chenxing Liu, and Feng Zhao. A reliable and accurate indoor localization method using phone inertial sensors. In *Proc. 14th Int'l Conf. on Ubiquitous Computing (UbiComp '12)*, pages 421–430, 2012.

[101] Oliver J. Woodman. An Introduction to Inertial Navigation. Technical Report 696, University of Cambridge, Computer Laboratory, 2007.

[102] Maurice Fallon, Hordur Johannsson, Jonathan Brookshire, Seth Teller, and John Leonard. Sensor Fusion for Flexible Human-Portable Building-Scale Mapping. In *IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS '12)*, 2012.

[103] Morgan Quigley, David Stavens, Adam Coates, and Sebastian Thrun. Sub-Meter Indoor Localization in Unmodified Environments with Inexpensive Sensors. In *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS '10)*, 2010.

[104] Ling Bao and Stephen S. Intille. Activity Recognition from User-Annotated Acceleration Data. In *Proc. 2nd Int'l Conf. on Pervasive Computing (PERVASIVE '04)*, 2004.

[105] Nishkam Ravi, Nikhil Dandekar, Preetham Mysore, and Michael L. Littman. Activity Recognition from Accelerometer Data. In *Proc. Nat'l Conf. on Artificial Intelligence (AAAI-05)*, volume 20, page 1541, 2005.

[106] Jonathan Lester, Tanzeem Choudhury, Nicky Kern, Gaetano Borriello, and Blake hannaford. A Hybrid Discriminative/Generative Approach for Modeling Human Activities. In *Proc. 19th Int'l Joint Conf. on Artificial Intelligence (IJCAI-05)*, 2005.

[107] Andreas Krause, Daniel P. Siewiorek, Asim Smailagic, and Jonny Farringdon. Unsupervised, Dynamic Identification of Physiological and Activity Context in Wearable Computing. In *Proceedings of the Seventh IEEE International Symposium on Wearable Computers*, 2003.

[108] T̆am Huỳnh, Mario Fritz, and Bernt Schiele. Discovery of Activity Patterns using Topic Models. In *Proc. 10th Int'l Conf. on Ubiquitous Computing (UbiComp '08)*, pages 10–19. ACM, 2008.

[109] Arvind Thiagarajan, Lenin Ravindranath, Katrina LaCurts, Sivan Toledo, Jakob Eriksson, Samuel Madden, and Hari Balakrishnan. VTrack: Accurate, Energy-Aware Road Traffic Delay Estimation Using Mobile Phones. In *Proc. 7th ACM Int'l Conf. on Embedded Networked Sensor Systems (SenSys '09)*, 2009.

[110] John Krumm, Julie Letchnet, and Eric Horvitz. Map Matching with Travel Time Constraints. In *Society of Automotive Engineers (SAE) 2007 World Congress*, pages 2007–01, 2007.

[111] Jun Han, Emmanuel Owusu, Le T Nguyen, Adrian Perrig, and Joy Zhang. ACComplice: Location Inference using Accelerometers on Smartphones. In *Proc. 4th Int'l Conf. on Communication Systems and Networks (COMSNETS)*, pages 1–9. IEEE, 2012.

[112] Arvind Thiagarajan, Lenin Ravindranath, Hari Balakrishnan, Samuel Madden, and Lewis Girod. Accurate, Low-Energy Trajectory Mapping for Mobile Devices. In *Proc. 8th USENIX Conf. on Networked Systems Design and Implementation (NSDI)*, pages 20–20. USENIX Association, 2011.

[113] Harry Blum. A Transformation for Extracting New Descriptors of Shape. *Models for the perception of speech and visual form*, 19(5):362–380, 1967.

[114] Franz Aurenhammer. Voronoi Diagrams — A Survey of a Fundamental Geometric Data Structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.

[115] Amy A. Kalia, Gordon E. Legge, and Nicholas A. Giudice. Learning building layouts with non-geometric visual information: The effects of visual impairment and age. *Perception*, 37(11):1677–1699, 2008.

[116] A. J. Bernheim Brush, Amy K. Karlson, James Scott, Raman Sarin, Andy Jacobs, Barry Bond, Oscar Murillo, Galen Hunt, Mike Sinclair, Kerry Hammil, and Steven Levi. User Experiences with Activity-Based Navigation on Mobile Devices. In *Proc. 12th Int'l Conf. on Human Computer Interaction with Mobile Devices and Services (MobileHCI '10)*, pages 73–82. ACM, 2010.

[117] Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R. Walter, Ashis Gopal Banerjee, Seth Teller, and Nicholas Roy. Approaching the Symbol Grounding Problem with Probabilistic Graphical Models. *AI Magazine*, 2011.

[118] Michael Levit and Deb Roy. Interpretation of Spatial Language in a Map Navigation Task. *IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics*, 37(3):667–679, 2007.

[119] Cynthia Matuszek, Dieter Fox, and Karl Koscher. Following Directions Using Statistical Machine Translation. In *Proc. 5th ACM/IEEE Int'l Conf. on Human-Robot Interaction (HRI '10)*, pages 251–258. ACM, 2010.

[120] Charles Sutton and Andrew McCallum. An Introduction to Conditional Random Fields for Relational Learning. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*, chapter 4, pages 93–128. The MIT Press, 2007.

[121] Talat Ozyagcilar. Calibrating an eCompass in the Presence of Hard and Soft-Iron Interference. Freescale Semiconductor Application Note, AN4246, 2012.

[122] Thomas Lavergne, Olivier Cappé, and François Yvon. Practical very large scale CRFs. In *Proc. 48th Ann. Meeting of the Association for Computational Linguistics (ACL '10)*, pages 504–513, 2010.

[123] Fei Sha and Fernando Pereira. Shallow Parsing with Conditional Random Fields. In *Proc. 2003 Conf. of the North American Chapter of the Association for Computational Linguistics on Human Language Technology (HLT-NAACL '03)*, volume 1, pages 134–141. Association for Computational Linguistics, 2003.

[124] Nobuyuki Shimizu. Semantic Discourse Segmentation and Labeling for Route Instructions. In *Proc. COLING/ACL 2006 Student Research Workshop*, pages 31–36, 2006.

[125] Stephen Friedman, Hanna Pasula, and Dieter Fox. Voronoi Random Fields: Extracting the Topological Structure of Indoor Environments via Place Labeling. In *Proc. 20th Int'l Joint Conf. on Artificial Intelligence (IJCAI-07)*, 2007.

[126] Douglas L. Vail, Manuela M. Veloso, and John D. Lafferty. Conditional Random Fields for Activity Recognition. In *Proc. 6th Int'l Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS '07)*, pages 1–8. ACM, 2007.

[127] Qiang Huang, Ling-Yun Wu, and Xiang-Sun Zhang. An Efficient Network Querying Method Based on Conditional Random Fields. *Bioinformatics*, 27(22):3173–3178, 2011.

[128] Patrick James Nichols. Location-Aware Active Signage. Master's thesis, Massachusetts Institute of Technology, 2004.

[129] Vitaliy Y. Kulikov. Building Model Generation Project: Generating a Model of the MIT Campus Terrain. Master's thesis, Massachusetts Institute of Technology, 2004.

[130] Emily Whiting. Geometric, Topological & Semantic Analysis of Multi-Building Floor Plan Data. Master's thesis, Massachusetts Institute of Technology, 2006.

[131] Jonathan Battat. A Fine-grained Geospatial Representation and Framework for Large-Scale Indoor Environments. Master's thesis, Massachusetts Institute of Technology, 2010.

[132] Robert Joan-Arinyo, Lluis Pérez-Vidal, and E Gargallo-Monllau. An Adaptive Algorithm to Compute the Medial Axis Transform of 2-D Polygonal Domains. In *CAD Systems Development*, pages 283–298. Springer, 1997.

[133] David H Douglas and Thomas K Peucker. Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or Its Caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[134] Lawrence R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications In Speech Recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[135] Yoshua Bengio and Paolo Frasconi. Input–Output HMM's for Sequence Processing. *IEEE Transactions on Neural Networks*, 7(5):1231–1249, September 1996.

[136] Biing-Hwang Juang and Lawrence R. Rabiner. The Segmental K-Means Algorithm for Estimating Parameters of Hidden Markov Models. *IEEE Trans. on Acoustics, Speech and Signal Processing*, 38(9):1639–1641, 1990.