iNav: A Hybrid Approach to WiFi Localization and Tracking of Mobile Devices

by

Lev Popov

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2008

Author .....
Department of Electrical Engineering and Computer Science
May 23, 2008

Certified by .....
Samuel Madden
Associate Professor
Thesis Supervisor

Accepted by .....
Arthur Smith
Chairman, Department Committee on Graduate Theses

iNav: A Hybrid Approach to WiFi Localization and Tracking of Mobile Devices

by

Lev Popov

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2008, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

This thesis presents iNav - a hybrid system for 802.11-based localization targeted at low-power mobile devices. WiFi localization enables numerous location-based services and applications without requiring a separate GPS module, thus offering device cost and power consumption savings.

iNav is a WiFi localization system targeted at low-power mobile devices, capable of utilizing multiple data sources to produce location estimates with accuracy higher than that of pure WiFi estimates. iNav uses a stochastic location estimation algorithm based on particle filters to integrate streams of WiFi access point observations and 3-axis accelerometer data. The system is tailored towards localization of vehicles and relies on a road network map to increase localization accuracy. iNav is designed with low-power devices in mind, and is capable of computing real-time location estimates on embedded devices like the iPhone.

Thesis Supervisor: Samuel Madden
Title: Associate Professor

Acknowledgments

I would like to thank my thesis supervisors, Samuel Madden and Hari Balakrishnan, for the great guidance and encouragement throughout my thesis work. I would like to thank Jakob Eriksson, whose collaboration and advice was invaluable in developing iNav. Finally, I would like to thank my family, friends, and fellow lab-mates for all their support.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

With the growing number of WiFi-equipped consumer devices - cell phones, personal media players, and even digital cameras - the prospect of using the existing WiFi infrastructure as a localization platform for location-based services is becoming increasingly feasible. WiFi localization systems work by using WiFi access points (APs) with known positions as radio beacons for triangulation. This thesis presents iNav — a WiFi localization system targeted at low-power mobile devices, capable of utilizing multiple data sources to produce location estimates with accuracy higher than that of pure WiFi estimates. iNav uses a stochastic location estimation algorithm based on particle filters[2, 15] to integrate streams of WiFi access point observations and 3-axis accelerometer data. The algorithm relies on a road network map to increase localization accuracy; because of this reliance it is primarily targeted for estimating vehicle locations and trajectories. The system is designed with low-power devices in mind and is capable of computing real-time location estimates on embedded devices like the iPhone.

This chapter explains the motivation for creating iNav, highlighting the advantages of iNav over alternative localization solutions. The next chapter provides background on the systems and algorithms that iNav builds upon. Chapters 3 and 4 explain the system design and implementation in detail, followed by Chapter 5 which presents an evaluation of iNav. Finally, chapter 6 discusses potential applications for the system, considerations for large-scale deployment, and possible improvements.

1.1 Advantages Over Alternative Localization Systems

This section compares iNav to alternative localization systems, explaining the advantages of iNav's approach.

1.1.1 GPS

While offering relatively high localization precision, GPS has several shortcomings which make it a poor fit for a number of applications. The first disadvantage of GPS is availability — GPS often fails when the device is indoors, in an urban canyon, or carried in a pocket with no clear visibility of the sky. These scenarios are usually not a problem for WiFi-based localization schemes. The last case — visibility to the sky — is particularly advantageous for WiFi-equipped devices like cell phones. While GPS systems require constant visibility to the sky (especially in cases of continuous tracking), cell phones using WiFi technology will continuously log location information with or without visibility to the sky.

Second, the Time to First Fix (TTFF) for GPS modules can be excessive for applications requiring immediate location information on a cold start — for example location-aware search or directory services. Since it is impractical from a power-management perspective to have the GPS module be active at all times, the user may have to wait up to a minute to get a response from the system for each search. With WiFi localization the position estimate is available as soon as a WiFi scan is complete (typically within a second from cold start), making it more practical for these types of applications.

Finally, cost is also a disadvantage of GPS when compared to WiFi localization — GPS modules are more expensive to add than WiFi radios.

1.1.2 Skyhook Wireless/Navizon

Skyhook Wireless and Navizon are two of the popular localization platforms using WiFi/GSM signals. These systems operate in a client-server setup, with the mobile device acting only as a sensor, and all of the location estimation computation taking place on a central server. While this configuration is attractive because it places very little complexity on the device itself, it suffers from estimation accuracy and latency issues, preventing these systems from being used in a large class of applications.

Because a communication round-trip between the device and the server is required for every location query, the latency of the location estimates can be prohibitively high. Especially over a slow GPRS link, the round trip can take over 10 seconds, making such localization schemes a poor fit for applications like navigational aids. The requirement of a persistent data link to the central server also negatively impacts the cost and availability. Running the localization computation locally on the device avoids these latency, availability, and cost issues.

Finally, these two systems offer relatively low positioning accuracy, since the location estimates are made based on individual WiFi scans. In comparison, iNav uses the entire observation history up to present, allowing for more accurate estimation.

1.2 Motivation for WiFi Localization

Using WiFi for localization is attractive for two reasons: its accuracy is good enough to power a wide class of location-based services, and its low cost and ease of deployment make it a very feasible proposition.

1.2.1 Location-Based Services and Applications

While existing WiFi-localization schemes do not offer the precision of a satellite positioning system, even the coarse-grained location estimates they produce can be practical for a large number of applications and services. Some of the applications enabled by WiFi-localization schemes for vehicles — such as iNav — are navigational

aids, vehicle tracking, and traffic estimation.

All three of these applications do not require very high localization precision — reporting the correct street on which vehicle is currently moving on, or even a street block, is usually good enough. While WiFi localization precision does not allow for a realtime turn-by-turn navigation system, it does enable a system that can report upcoming turns within the next minute and providing a map of immediate surroundings — still a significant navigational aid.

Approximate location information can be used for almost all vehicle tracking applications: reporting expected bus/shuttle arrival times, tracking package delivery vehicles, and monitoring locations of cars in a taxi fleet to optimize cab dispatching. For these applications, the WiFi radio can perform double duty — both acting as a location sensor, and functioning as a means for uploading the location information from the tracked device.

Finally, if the vehicle tracking information is allowed to be aggregated, it can be used for traffic density estimation in urban environments. Given a sufficient number of probe vehicles reporting their location to a central server, congestion estimates can be computed for the urban road network.

1.2.2 Advantages of WiFi-Localization

The main advantage of WiFi-localization is the low cost of deployment — both in terms of the cost of required infrastructure and the cost of consumer devices that can utilize such system. Most urban centers and their suburban surroundings have a very high density of deployed WiFi access points (AP); a typical one hour drive through the Boston metropolitan area will pass within range of 1000–2000 access points. The high density of already deployed public and private access points eliminates the need for any additional infrastructure to make a WiFi localization system functional. The only required infrastructure development is collecting war-driving data that is necessary to map the already-deployed access points.

The cost of adding a WiFi radio in a consumer electronics device is relatively low as well. Because many devices on the market already include a WiFi antenna, the

only component needed to enable location-based services for these devices is software.

1.2.3 Advantages of Accelerometer-Assisted Systems

iNav is capable of utilizing accelerometer input to improve location estimation accuracy. This section offers the intuition for why combining accelerometer and WiFi data is a good fit.

While accelerometer does not allow for absolute positioning, it can be used to infer relative movement between locations with good accuracy. In contrast, WiFi location estimates offer absolute positioning, but the noise and gaps in WiFI coverage make the accuracy of the estimates fairly low. Combining accelerometer and WiFi information helps cover the gaps that the individual data sources suffer from. WiFi localization can provide individual location estimates, while the accelerometer data can be used both to compute location estimate when a gap in WiFi coverage is encountered, and to disambiguate between possible locations when WiFi observations are noisy.

1.3 iNav Overview

iNav produces location estimates with a particle filter-based algorithm, using WiFi observations for its sensor model, and a street map combined with accelerometer data for its transition model.

iNav offers three main contributions. First, iNav implements a particle filter that is capable of integrating accelerometer data together with WiFi observations to compute location estimates that are more accurate than the ones based on WiFi alone. Second, iNav presents an optimized WiFi-localization system that can utilize large amounts of AP location and street map data while running on a resource-limited device such as a cell phone. Finally, iNav offers an evaluation of accelerometer-assisted WiFi-localization system in environments with varying levels of WiFi coverage.

Chapter 2

Background

This chapter presents several pieces of earlier work on WiFi-localization upon which iNav is based and offers a brief survey of the well established WiFi-localization algorithms. Chapter 3 describes in detail how these algorithms are applied in iNav, and Chapter 4 explains the specifics in implementing these algorithms.

2.1 Related Work

Place Lab system[10, 4] and its derivatives are presently some of the most commonly used 802.11 based positioning systems. Place Lab relies on a database of WiFi access points with known positions acting as beacons for tracked devices. Projects like UCSDs Active Campus[7] proposed various approaches to estimating location given observed beacon set, such as relying on RF signal strength, and modeling of RF signal propagation in indoor environments. The requirement for having a database with APs with known locations is a major restriction for Place Lab, but some recent work has partially alleviated this issue. By using self-mapping algorithms that can determine access point locations given a small initial training set[11], Place Lab has been able to more easily estimate specific locations.

Bahl's et al RADAR[3] system uses a fundamentally different approach by recording radio signal fingerprints (for example signal strengths of visible 802.11 APs) for a set of locations, and matching these against the training fingerprint database to esti-

mate positions of tracked devices. This approach requires a large amount of detailed training data, but is capable of providing highly accurate position estimates, with median errors of just 2–3 meters. Section 2.2.2 describes this approach in detail.

Additional work has been done attempting to improve the basic beacon and fingerprint based positioning schemes by taking into account the past observations for the tracked object in addition to the present readings. The scheme designed by Ladd et al[9] and the Locadio project[8] at Microsoft Research both take this approach — using hidden Markov models to compute current location probabilities given past observations prior. Using sequential Monte Carlo algorithms such as particle filters[5] is another possible technique, attempted by [12] and [13]. Section 2.3 provides additional information on particle filters.

iNav uses a particle filter-based algorithm for its location estimation. iNav's departure from prior work is in attempting to use a particle filter to integrate separate streams of WiFi access point observations and accelerometer data. iNav's use of a street map to assist the particle filter, and the system's focus on low-powered mobile devices separates it from earlier work as well.

The following two section present a brief survey of the well established WiFi-localization algorithms.

2.2 Single-Point Estimation Algorithms

This section describes the basic single-point location estimation algorithms for WiFi-positioning systems. These algorithms take as an input a single WiFi fingerprint represented as a set of $(MAC, RSSI)$ pairs, and produce a location estimate (a $(Latitude, Longitude)$ pair) as the output. These algorithms are trained on a large database mapping WiFi fingerprints to locations where they've been made. Typically, such databases are accumulated by war-driving — having a vehicle, equipped with a WiFi scanner and a GPS, drive around the area of interest while recording WiFi observations labeled with the corresponding GPS location.

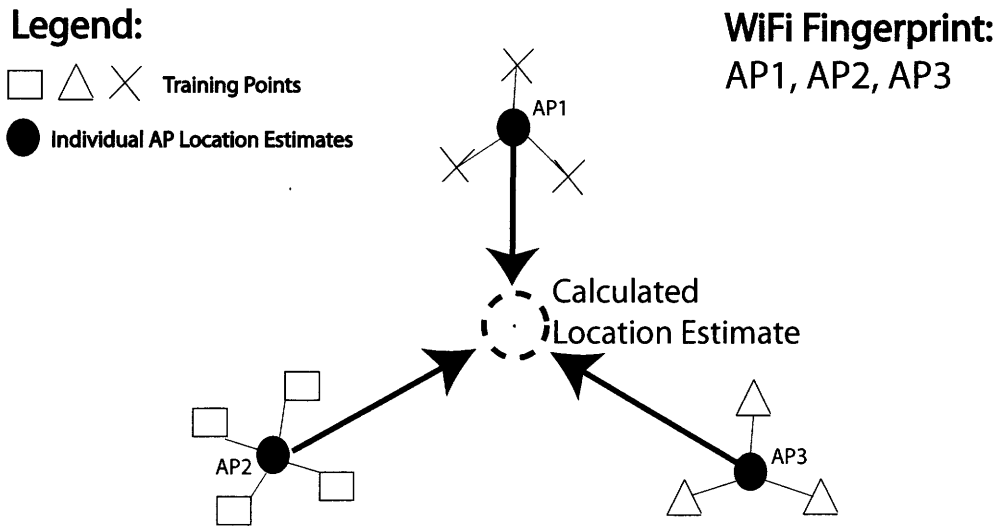


Figure 2-1: The centroid algorithms

2.2.1 Centroid Algorithm

Centroid is one of the simplest single-point estimation algorithms. From the training data, the algorithm tries to estimate the locations of all the individual access points in the training set, using a simple centroid of all the observations for each access point. Then, when a location estimate needs to be computed for an input WiFi fingerprint, the algorithm takes the centroids of all the access points seen in the fingerprint, and computes a centroid of those, returning it as the estimate. The pseudocode for the centroid algorithm is listed in Algorithm 1. The algorithm is attractive because of its extremely low processing and training storage requirements, but offers relatively low precision.

Several variations of the Centroid algorithm exist, which are described below and illustrated in Figure 2-1.

RSSI-Weighted Centroid

RSSI-Weighted Centroid is a variation of the centroid algorithm. Instead of computing the output location estimate centroid uniformly over the centroids of APs seen in

input : training map $D : AP \rightarrow \{Loc_1 \ldots Loc_k\}$, WiFi fingerprint $\{AP_1 \ldots AP_n\}$
output: estimated location

```
estimate ← new Location(0,0)
points ← 0
for i ← 1...n do
    for loc ∈ D[AP_i] do
        points ← points + 1
        estimate ← estimate + loc
estimate ← estimate/points
return estimate
```

Algorithm 1: Centroid Algorithm

the input fingerprint, the AP centroids are weighted by their observed signal strength. This variation benefits from RSSI information by shifting the centroid towards the access points that are likely to be closer than the others.

Bounding-Box Center

The Bounding-Box Center algorithm computes a bounding box of observations and selects its center instead of computing a centroid of the observations. This variation is less affected by irregularities in training data, such as when a particular access point is visible from two streets, but much more frequently on one than the other, causing the centroid of the AP observations to shift towards the more trafficked street.

2.2.2 Fingerprinting Algorithm

The fingerprinting algorithm is based on the RADAR work of Bahl et al[3], and is a fundamentally different approach to single-point location estimation. This algorithm discretizes the location space into individual locations, and stores a database of all the training fingerprint-location pairs. When a location estimate needs to be made, the algorithm scores the input fingerprint against its training fingerprint database, looking for the best match. The location of the best-matched fingerprint is reported as the output location estimate. Figure 2-2 illustrates the algorithm, and the pseudocode is shown in Algorithm 2.

Compared to the centroid algorithm, fingerprinting has the potential to be much

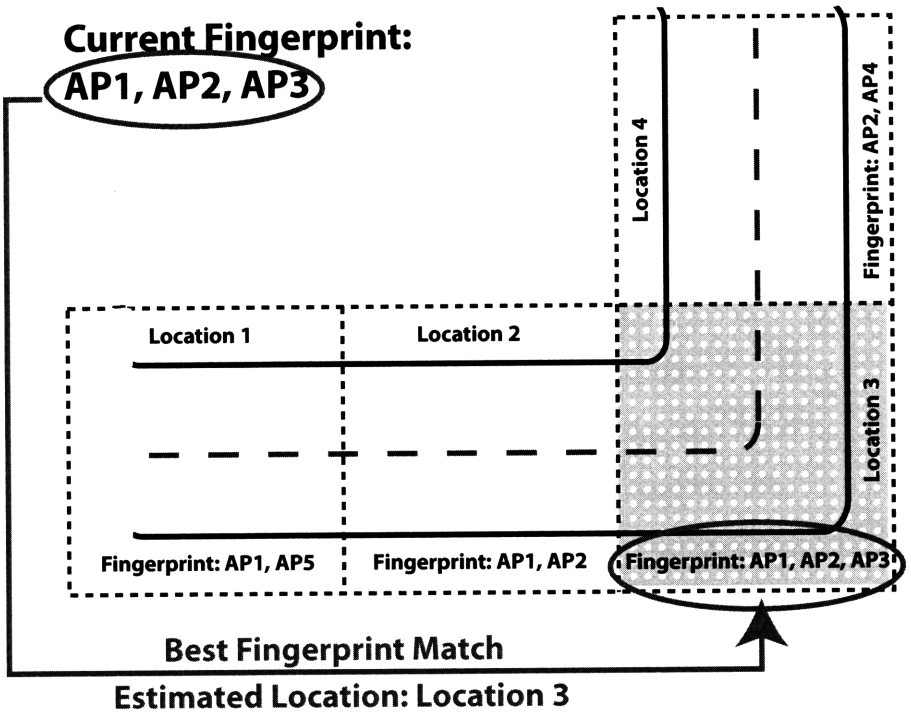


Figure 2-2: The fingerprinting algorithm

input : training map $D : \{AP_1 \dots AP_k\} \rightarrow loc$, WiFi fingerprint $\{AP_1 \dots AP_n\}$
output: estimated location

```
1 estimate ← new Location(0,0)
2 bestScore ← 0
3 for F ∈ D.keys do
4 |   score ← matchFingerprints(F, {AP_1 ... AP_n})
5 |   if score > bestScore then
6 |   |   bestScore ← score
7 |   |   estimate ← D[F]
8 |
9 return estimate
```

Algorithm 2: Fingerprinting Algorithm

more accurate, but also requires much more storage space for the training data and is considerably more computationally expensive.

2.3 Particle Filter Algorithm

In contrast with the algorithms from the previous section, particle filter computes a location estimate given the entire sequence of WiFi observations up to the present. The algorithm maintains a probability distribution of the current device location, and updates it at every time step as each new WiFi observation becomes available. The probability distribution is represented as a vector of weighted particles, and is initialized using the first WiFi observation, such that the particles represent the probability distribution of the current location given the first observation. After initialization, the algorithm proceeds enters the main cycle, with each iteration consisting of *Update*, *Reweight*, and *Resample* steps, explained below.

In *Update* step, all particle locations are updated according to particle filter's Mobility Model. Possible Mobility Models vary in sophistication, ranging from naive models that simple translate each particle a random distance from its previous location, to much more complex models that simulate cars driving on streets.

To maintain the probability distribution of the current location, the algorithm reweighs the particles based on their new locations and the transitions made in the *Update* step. Using the probabilistic interpretation of particle weight, the update equation for the weight $w_i(t)$ of particle i with location $x_i(t)$ at time t when WiFi observation $O(t)$ is made, is

$$w_i(t) = w_i(t-1)P(x_i(t)|O(t))P(x_i(t)|x_i(t-1))$$

The new weight $w_i(t)$ of the particle i is its old weight $w_i(t-1)$ multiplied by the probability that the tracked device is at particle's location $x_i(t)$ given current WiFi observation $O(t)$, multiplied by the probability of transitioning from particle's old location $x_i(t-1)$ to its new location $x_i(t)$. Particle filter's Sensor Model is

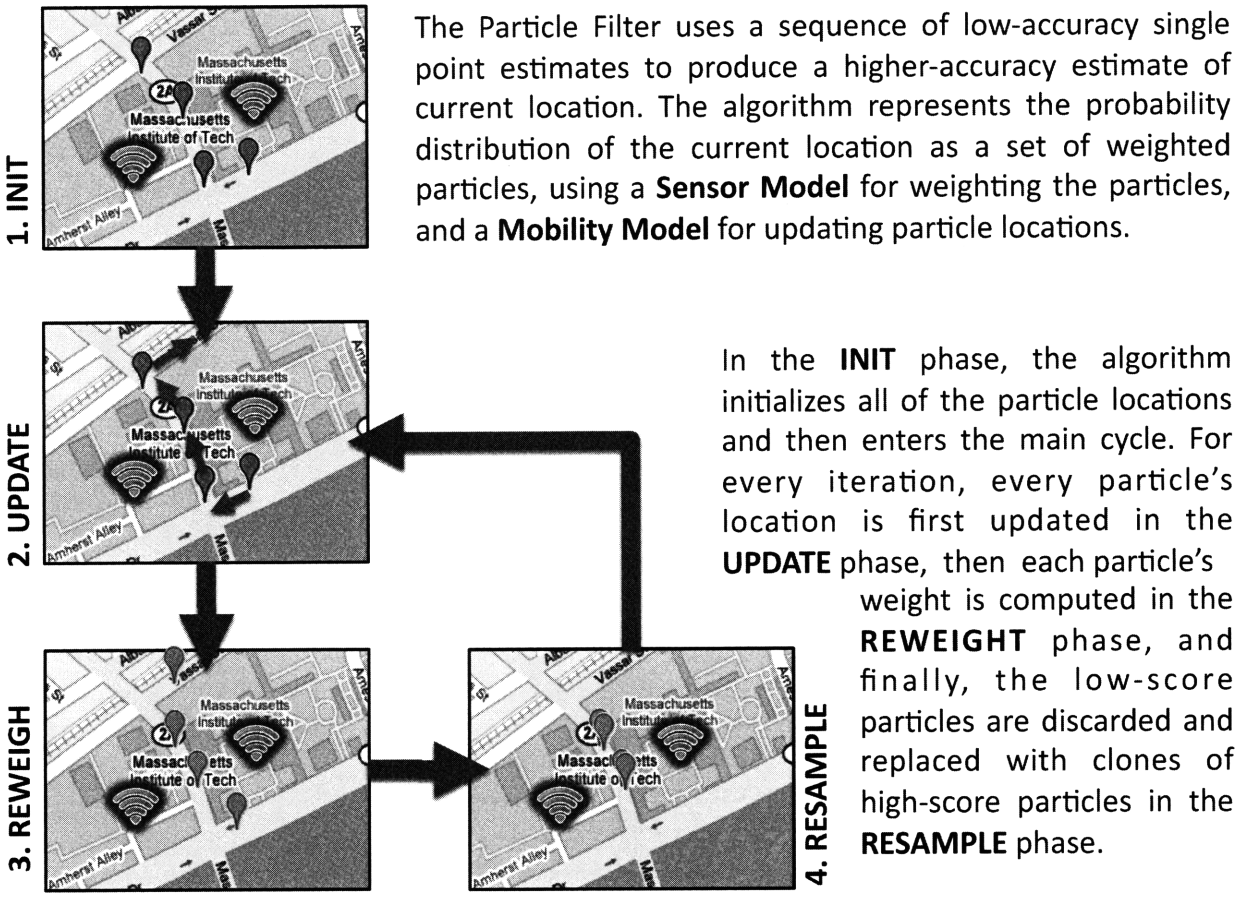


Figure 2-3: Particle filter used for vehicle location estimation.

responsible for estimating $P(x_i(t)|O(t))$ and the Mobility Model is responsible for estimating $P(x_i(t)|x_i(t-1))$.

At the end of the *Reweight* step, particle weights are normalized so that they all add up to one.

The final step of the algorithm cycle is the *Resample* step. In this step, particle distribution is resampled, discarding particles with low weights and replacing them with high-weighted particles. A preset percentage of particles is discarded, with each removed particle being replaced by a clone of a random particle sampled from the weighted particle distribution.

Figure 2-3 illustrates the execution of the particle filter, and Algorithm 3 shows the pseudocode.

input : WiFi observation sequence $W : t \rightarrow \{AP_1 \ldots AP_k\}$, number of particles N, number of particles to resample at each step K, mobility model M, sensor model S

output: estimated location sequence

```
   /* Initialize particles                                                    */
 1 particles ← new List
 2 for i ← 0...N do
 3 |   particles.append(S.sampleParticle(W[0]))
 4 /* enter main Update-Reweight-Recycle loop                                */
 5 for t ∈ W.keys do
 6 |   sum ← 0
 7 |   for i ← 0...N do
   |   |   /* Mobility Model updates particle location and returns the
   |   |      probability of the transition                                  */
 8 |   |   transProb ← M.updateParticle(particles[i])
 9 |   |   locProb ← S.scoreParticle(particles[i],W[t])
10 |   |   particles[i].weight ← particles[i].weight *transProb * locProb
11 |   |   sum ← sum + particles[i].weight
12 |   /* Normalize weights                                                  */
13 |   for i ← 0...N do
14 |   |   particles[i].weight ← particles[i].weight /sum
15 |   /* Resample particles                                                 */
16 |   for i ← 0...K do
17 |   |   i ← random(0,N)
18 |   |   good ← weightedSampleParticle(particles)
19 |   |   particles[i].cloneParticleFrom(good)
20 |   /* Output estimate                                                    */
21 |   estimates.append(weightedAverage(particles))
22 return estimates
```

Algorithm 3: Particle Filter Algorithm

Chapter 3

iNav Design

This chapter explains iNav's system architecture in detail, describing how algorithms from Chapter 2 are used to create a hybrid WiFi/accelerometer localization scheme. The chapter begins with a general overview of the system design and then examines each of the components individually.

3.1 System Overview

iNav is composed of four major components, illustrated in Figure 3-1. At the center of iNav is the particle filter that is responsible for maintaining system's belief state about the tracked vehicle's location. The system takes continuous streams of WiFi and accelerometer observation data as inputs, which are then routed to the appropriate processing modules. The incoming WiFi fingerprints are directed to the WiFi Model, and the accelerometer readings are processed by the Accelerometer Model. Once the current WiFi/accelerometer observations are processed by their respective models, iNav's particle filter is stepped through the *update/reweight/resample* cycle and a location estimate is produced by the system.

The WiFi Model is used as the sensor model of the particle filter, while the Accelerometer Model is used an an input to the Mobility Model. This mapping is natural — the accelerometer data provides information on how the vehicle transitions from its previous location to the current location, allowing the Mobility Model to estimate

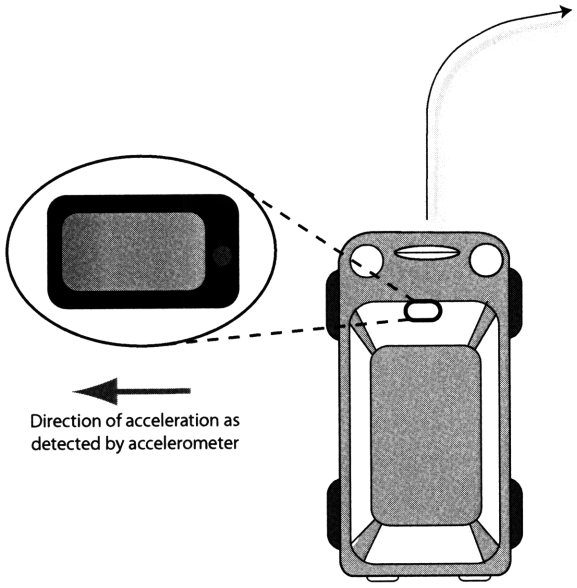


Figure 3-2: Accelerometer used for identifying vehicle turns

full 6 degrees of freedom accelerometer with very high accuracy and sampling rate. While a simple 3-axis accelerometer is a lot more difficult to use in this fashion, a great amount of device/vehicle motion information can still be extracted from its readings. The features that can be extracted with a 3-axis accelerometer are best compared to what a blindfolded person sitting in a car can detect. It is possible to tell when the vehicle is stopped or moving, when it is accelerating or slowing down, and when it is turning (along with the direction of the turn).

The Accelerometer Model used by iNav extracts all of these features. By having the device accelerometer axis aligned with the car body (or having the information of device's orientation relative to the car, and assuming that it remains fixed), changes in car's speed can be determined by looking at longitudinal acceleration values, and turning can be detected by spikes in lateral acceleration. Figure 3-2 illustrates this.

iNav processes the accelerometer data with a simple averaging time window. Turns are labeled by comparing the lateral acceleration window average with a preset threshold value, and stops are identified by lack of variance in accelerometer samples over the time window (also compared to a preset threshold value). Algorithm 4 shows the pseudocode for the accelerometer processing logic. The intuition behind the averag-

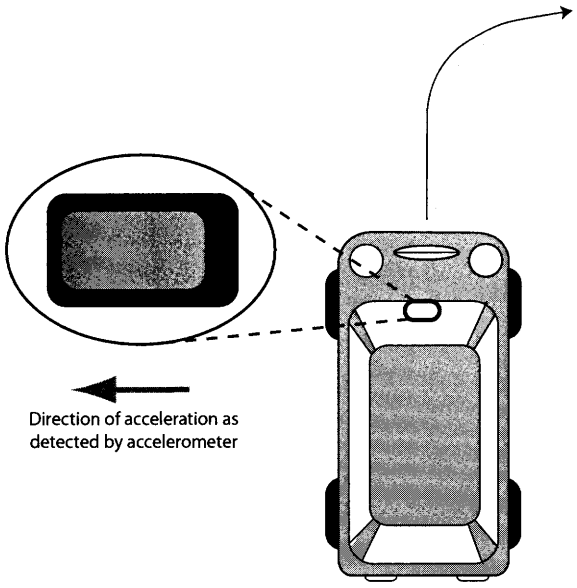


Figure 3-2: Accelerometer used for identifying vehicle turns

full 6 degrees of freedom accelerometer with very high accuracy and sampling rate. While a simple 3-axis accelerometer is a lot more difficult to use in this fashion, a great amount of device/vehicle motion information can still be extracted from its readings. The features that can be extracted with a 3-axis accelerometer are best compared to what a blindfolded person sitting in a car can detect. It is possible to tell when the vehicle is stopped or moving, when it is accelerating or slowing down, and when it is turning (along with the direction of the turn).

The Accelerometer Model used by iNav extracts all of these features. By having the device accelerometer axis aligned with the car body (or having the information of device's orientation relative to the car, and assuming that it remains fixed), changes in car's speed can be determined by looking at longitudinal acceleration values, and turning can be detected by spikes in lateral acceleration. Figure 3-2 illustrates this.

iNav processes the accelerometer data with a simple averaging time window. Turns are labeled by comparing the lateral acceleration window average with a preset threshold value, and stops are identified by lack of variance in accelerometer samples over the time window (also compared to a preset threshold value). Algorithm 4 shows the pseudocode for the accelerometer processing logic. The intuition behind the averag-

ing is that a turn is characterized by continuous presence of lateral acceleration over multiple seconds — an averaging time window will capture these occurrences given an appropriate size of the window. The threshold and window size values are calibrated from the training data. Window size of 2 seconds, and average lateral acceleration threshold cutoff of 0.88 m/s^2 (roughly 1/10th of g) are used for turn detection in the evaluation system.

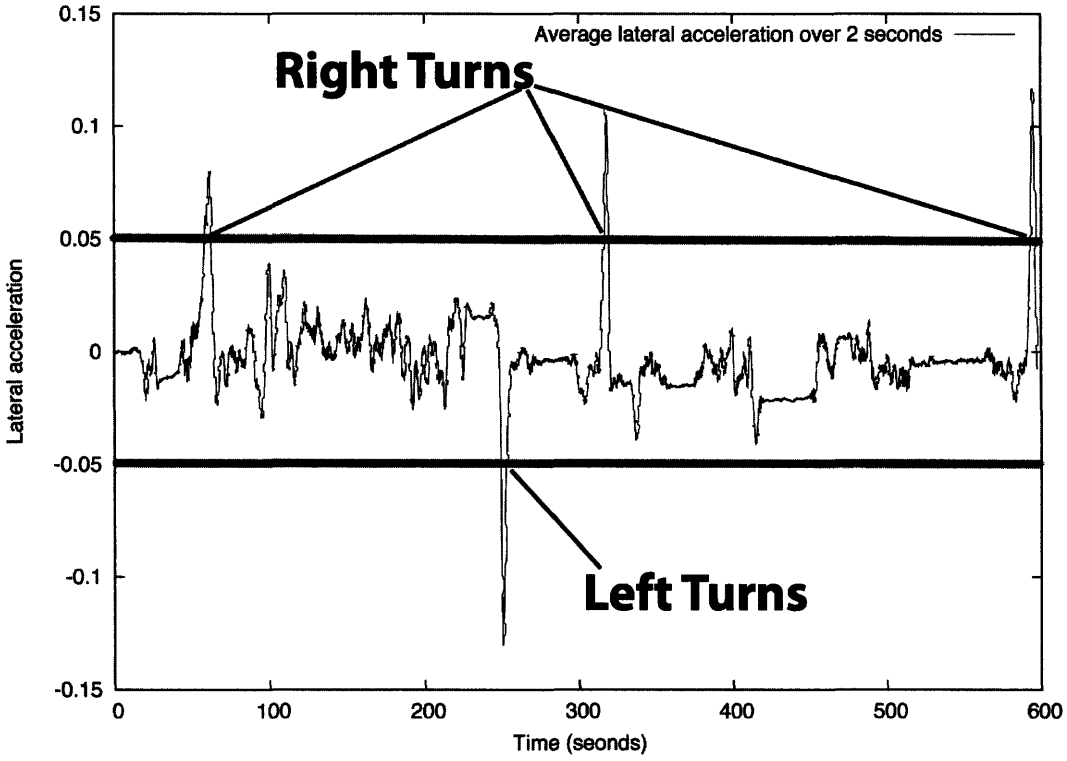


Figure 3-3: Lateral acceleration averages over time, with turn detection thresholds

Figure 3-3 shows a plot of time window-averaged lateral acceleration values, along with threshold values for identifying turns and stops.

The performance of iNav's Acceleration Model at correctly identifying vehicle motion features such as turns is evaluated in section 5.2.

input : lateral acceleration input sequence $A : t \rightarrow acceleration$, Turn detection threshold T, averaging window size W
output: sequence of times for which turns were detected

```
1 turns ← new List
2 average ← 0
3 for t ∈ A.keys do
4 |   average ← average + A[t]
5 |   average ← average − A[t − W]
6 |   if |average| > T then
7 |   |   turns.append(t)
8 |
9 return turns
```

Algorithm 4: Accelerometer Turn Detection Algorithm

3.3 iNav WiFi Model

The purpose of the WiFi Model in iNav is to process incoming WiFi observations, using the resulting information for the *intialization* and *reweight* phases of the particle filter. The WiFi Model requirements are similar to that of single-point estimation algorithms reviewed in section 2.2. While any of the single point estimation algorithms can be used for the WiFi Model, iNav uses a plain Centroid algorithm, for its computational simplicity and accuracy comparable to that of other single-point estimation algorithms.

The WIFi Model maintains a database of $(MAC- > Location)$ pairs for every known access point, along with a mapping between AP identifiers and the streets on which the AP was observed in the training dataset. The $(MAC- > Location)$ pairs are used for the Centroid algorithm, and the $(AP- > Street_0 \ldots Street_n)$ map is used for particle initialization, as described in Section 3.4.1.

When estimating $P(x_i(t)|O(t))$ for particle reweighing, the WiFi Model uses the single point estimate $Estimate(O(t))$ in the following equation:

$$P(x_i(t)|O(t)) = 1/Distance(x_i(t), Estimate(O(t)))$$

Particles farther than a certain distance (500 meters is used) are assigned zero score. This is done to allow the particle filter to enter a degenerate state and restart

when the particle distribution no longer matches the incoming WiFi observations.

3.4 iNav Particle Filter

The particle filter is at the center of iNav's design, and it is responsible for processing the outputs of all other modules and computing the final location estimate. iNav uses the particle filter algorithm exactly as presented in Section 2.3, with the details of the four phases — *initialization*, *update*, *reweight*, and *resample*, described below.

3.4.1 Particle Initialization

Particle initialization takes places as soon as the system receives the first WiFi fingerprint that the WiFi Model can recognize (i.e. it contains APs that are in the training database). To initialize particle locations, the WiFi Model looks up the set of streets on which any of the APs from the fingerprint was observed. The particles are initialized with locations uniformly sampled from this street set. Each particle's initial speed is picked randomly between zero and maximum allowed speed by the system (80mph).

3.4.2 Map-Assisted Mobility Model

The Mobility Model is responsible for the *update* phase of the particle filter algorithm, and for computing the transition probability estimate in the *reweight* phase. To reduce the search space that needs to be covered by the particles, all particle trajectories are restricted to streets on the map. At every *update* step, the particles move some distance along the streets (based on their current speed), making random turns when they hit intersections. Every particle keeps track of its speed, which is adjusted with random gaussian acceleration applied at every step.

The Mobility Model makes heavy use of the accelerometer data. Since the Accelerometer Model can identify when the vehicle is stopped or changing its speed, this information can be used to bias the motion of particles. The Mobility Model takes

processed accelerometer data as an input, and adjusts the particle speeds based on the accelerometer values. If the vehicle is determined to be stopped by the Accelerometer Model, then all particle speeds are set to zero. While the accelerometer data can also be used to determine whether the car is speeding up or slowing down, only very coarse-grained estimates are possible, making this information not very useful for the Mobility Model.

Optionally, the accelerometer data can be used to bias turn directions of the particles, although iNav does not. Instead, iNav uses the turn information reported by the accelerometer to compute the estimates of $P(x_i(t)|x_i(t-1))$ that the Mobility Model returns for the *reweight* step of the particle filter.

After updating particle locations, the Mobility Model determines whether each of the particles made a turn in the update step. This is determined by computing the angle between the previous street segment that the particle was on and the current street segment. If the angle is above a preset threshold (60 degrees is used), then the transition is labeled as having made a turn. For each particle that made a turn, the score reported by the Mobility Model is 1 if the turn matches what Accelerometer Model reports, and 0.1 otherwise. All particles that did not make a turn get assigned a score of 1.

3.4.3 Particle Reweighting and Resampling

The reweighting and resampling of particles in iNav is done in exactly the same way as described in Section 2.3. For reweighting, each particle's weight is multiplied by the scores returned by the WiFi Model and the Mobility Model, and all the weights are re-normalized afterwards.

For resampling, at every iteration iNav replaces randomly selected 10% of the particles with particles sampled from the weighted particle distribution. Section 4.2.1 provides details on the exact resampling algorithm implementation used by iNav.

3.4.4 Location Estimate Reporting

iNav reports the location estimate the same way as described in Section 2.3. A weighted average of all particle locations is taken to produce the estimate.

Additionally, iNav is capable of reporting an estimate of the street segment that the vehicle is currently on. The estimate is computed by selecting the street segment with largest cumulative particle weight.

3.4.5 Dealing With Degenerate Cases

A particle filter can suffer from degeneracy when all of its particles end up with zero weights. While it is possible to avoid this scenario by injecting newly initialized particles at every iteration of the algorithm, iNav does not attempt to do so. Instead, the system lets the algorithm reach a degenerate case if the evidence leads to zero scores for all particles, and simply stops reporting a location estimate. Once degeneracy is reached, the system restarts the particle filter, re-initializing all of the particles in the same way it did in section 3.4.1.

Chapter 4

iNav Implementation

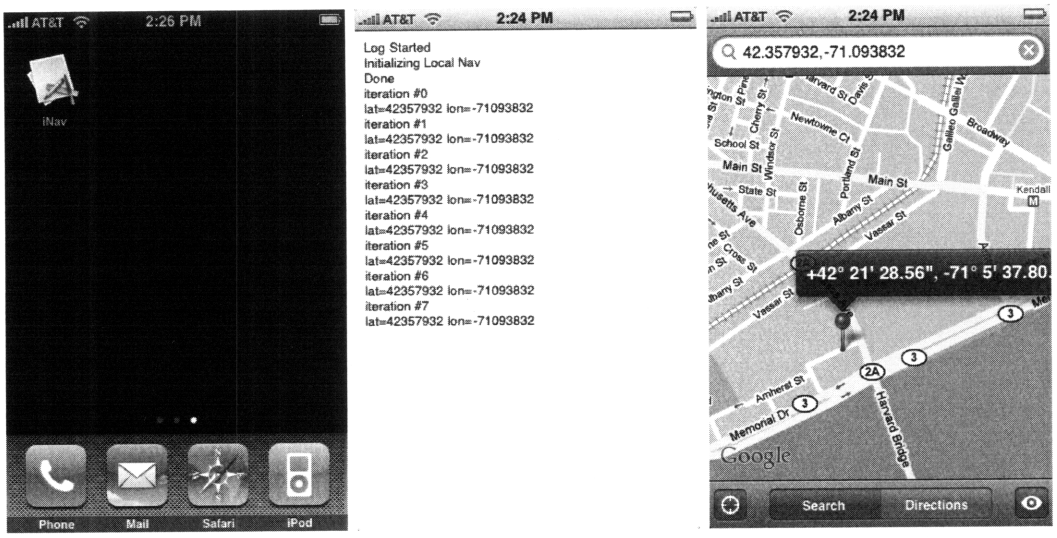
iNav has been designed to run on low-power mobile devices such as cell phones, and includes a number of optimizations to make a fairly sophisticated algorithm like particle filter run on in a very resource-limited environment. This chapter describes the specifics of iNav implementation and presents the optimizations that the system uses to achieve its performance.

4.1 Overview

The target platform for the current implementation of iNav is Apple iPhone. The iPhone has both a WiFi radio and a builtin 3-axis accelerometer, making it an ideal target device for iNav. iPhone also has plenty of computational resources available for a cell phone — a 400MHz ARM CPU, 128MB of RAM, and up to 16GB of storage space.

Figure 4-1 shows screenshots of the iNav client on the iPhone. The demo application runs the particle filter implementation, stepping the algorithm every time a new accelerometer and/or WiFi observation is recorded. The application is capable of sending the current location estimate to iPhone's Maps.app to visualize the device position.

While the code for acquiring WiFi and accelerometer readings is iPhone-specific, the rest of the codebase is platform-independent. In addition to the iPhone client,



(a) iNav application (b) iNav console (c) iNav output to Maps.app

Figure 4-1: iNav iPhone client

a build of iNav for Mac OS X-x86 exists. The iNav codebase is relatively small, consisting of less than 2000 lines of code. iNav is written mostly in C++ with some iPhone-specific code done in Objective C.

Source code from the Stumbler[1] project is used for WiFi scanning code on the iPhone. With the scanning code, WiFi observations are taken once every two seconds on average, The system samples iPhone's accelerometer readings at 50Hz.

4.2 Optimizations

This section discusses the optimizations that went into iNav to allow it to run in resource-limited environments, such as cell phones with limited amount of RAM and slow CPUs.

4.2.1 Fast Particle Resampling with Particle Distribution Bucketing

Particle resampling is one of the most processing-intensive steps in a particle filter. Given N total particles and K particles to be resampled, a naive implementation

would take $O(NK)$ time to sample a replacement particle for each particle that is discarded.

iNav introduces an optimization for the resampling process that discretizes particle distribution into B buckets, reducing the runtime complexity to $O(NlogN)$. The intuition for optimization lies in the idea that one could simplify the particle distribution by splitting it into a fixed number of buckets such that every bucket has particles with approximately same weight, and the cumulative weight of all particles in each bucket is the same as well. Then to sample a particle from the particle distribution, one could simply uniformly select a random bucket, and then select a random particle in the bucket.

To build up the buckets, all the particles are first sorted by their weight (taking up $O(NlogN)$ time which dominates the rest of the steps). Then, an indexing array *buckets*, defining the bounds of each bucket, is built by scanning through the sorted distribution once. Once all the buckets are built, the K particles are sampled as described in the previous paragraph. The pseudocode for the optimized resampling step is shown in Algorithm 5.

4.2.2 Flat-File DBs

Low memory utilization and quick startup times are crucial in applications targeting low-power mobile devices. iNav addresses these requirements by storing all of the system data (the access point table, and the street map) in flat-file databases residing on the phone's flash memory. Because all of the data remains on permanent storage only the particles themselves must be allocated in RAM at runtime, minimizing memory consumption. The startup times are minimized as well since no data needs to be loaded to RAM on a cold start. iNav can begin outputting location estimates as soon as the first WiFi scan is complete.

The flat-file databases are extremely simple. The databases themselves consist of fixed-size records with a unique fixed-sized key for each record. Keys are laid out in a sorted order forming an index. Binary search on the index is used for record retrieval, and no caching is performed, leaving that to the application.

```
input : Particle array particles, number of particles N, number of particles to
        resample K
output: Resampled array of particles particles
1  sorted ← sort(particles)
2  buckets ← new Array
3  sum ← 0
4  bucket ← 0
5  i ← 0
6  while i < N do
7  |   sum ← sum + sorted[i].weight
8  |   while sum >= bucket/B do
9  |   |   buckets[bucket] ← i
10 |   |   bucket ← bucket + 1
11 |
12 i ← 0
13 while i < K do
14 |   pick ← random(0, 1)
15 |   bucket ← pick * B
16 |   idx ← random(buckets[bucket], buckets[bucket + 1])
17 |   sorted[i].cloneFrom(sorted[idx])
18
```

Algorithm 5: Optimized Particle Filter Resampling Algorithm

For access points, the keys are MAC addresses, and the records include AP location estimates and streets on which the access point has been seen in the training data. For street map DB, intersection ID is the key, and the record consists of IDs of all connecting intersections.

4.2.3 Caching Strategies

Because all of the data remains on the slow permanent storage, caching is necessary to achieve acceptable system performance. The WiFi Model caches both the records for every access point looked up and the location estimate for every WiFi fingerprint processed. The fingerprint estimate caching ensures that the AP database is only accessed once per WiFi scan, which typically happens no more than once every second.

The Mobility Model maintains a cache of the recently looked up street intersections. The LRU policy is used for cache replacement. Because under typical running conditions intersections from a limited map area are accessed (the streets on which the current set of particles resides), LRU is a reasonable choice.

Chapter 5

Evaluation

This chapter presents a numerical evaluation of iNav's performance, both at the individual component level, and for the system as a whole. The methodology section explains the specifics of collecting training and testing data, and is followed by the component-level evaluations of the Accelerometer Model and the WiFi Model. The performance of the system as a whole is evaluated both in terms of distance error, and road segment identification accuracy,in section 5.4. Finally, the chapter concludes with an analysis of iNav resource utilization at runtime.

5.1 Methodology

5.1.1 Training Data

iNav relies on the vTrack[6] war-driving database for its training data, which is populated by records from 30+ taxi cars driving in the Greater Boston area. The database consists of WiFi fingerprints mapped to the locations where they have been observed, as marked by GPS. Since locations of access points whose observations span large areas cannot be accurately estimated by the centroid algorithm, these access points are filtered out from the training dataset. The data is cleaned to remove access points whose observation points span an area larger than a kilometer in the diagonal. The resulting cleaned training set includes over 450,000 unique access points.

iNav requires a road network map for the Mobility Model of its particle filter. NavTeq map data was used for the evaluation.

5.1.2 Testing Data

The testing data was collected over 3 hours of driving in Downtown Boston, Cambridge, Somerville, and Brookline areas. Figure 5-1 shows the GPS trace of the test vehicle driving, and the test data collection setup is illustrated in Figure 5-2. An iPhone running accelerometer and WiFi loggers is mounted on the vehicle's dashboard, and a separate Garmin eTrex Vista HC GPS unit is used for recording GPS trace of the car.

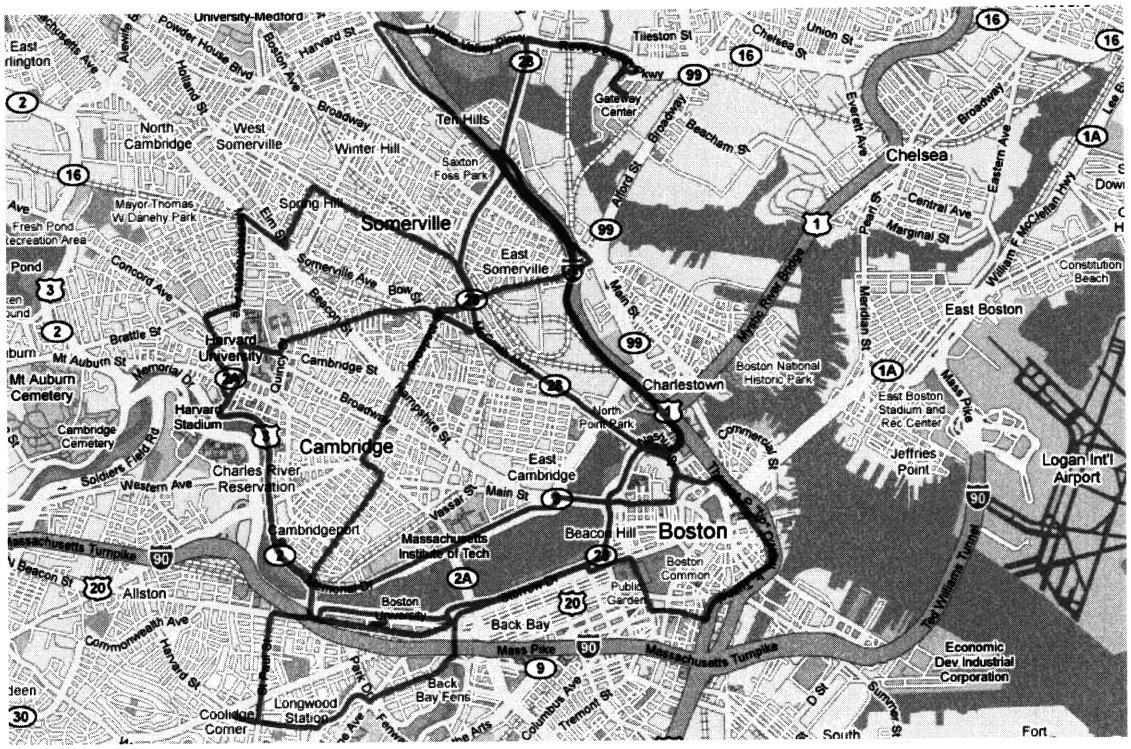
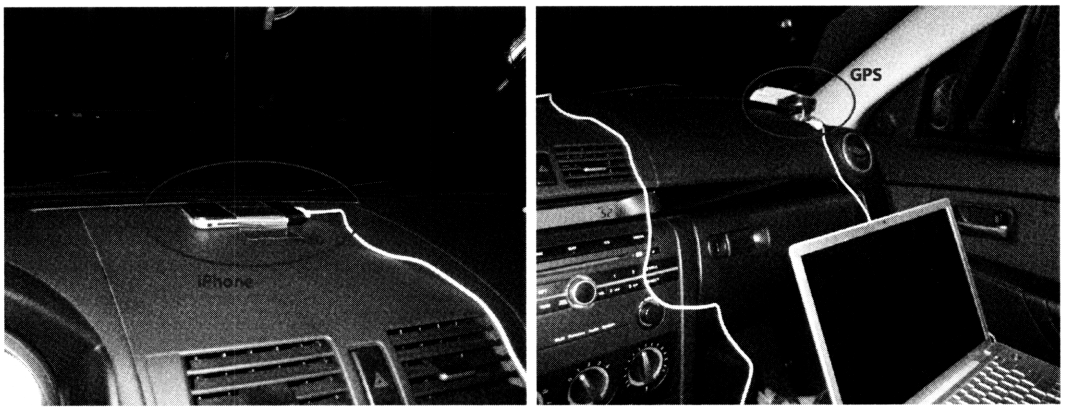


Figure 5-1: GPS trace of the test vehicle

While the iPhone is capable of logging WiFi fingerprints, the evaluation WiFi data was collected using a separate Meraki Mini box. Due to lack of control over access point caching in the current version of iPhone's Stumbler WiFi scanning code, the resulting WiFi fingerprints are not a good estimate of the current location of the

device. To avoid this problem for the evaluation, the WiFi scans from Meraki Mini were used instead. The Meraki Mini runs a modified firmware that performs WiFi scans every second, and periodically sends the traces to a central server where the fingerprints are processed and stored in a database.



(a) iPhone logging acceleration (b) GPS logging ground truth location

Figure 5-2: Data collection setup

Since accelerometer, GPS, and WiFi data are collected on three separate devices, the time synchronization issue must be addressed. The GPS and the iPhone get their clocks set by GPS satellites and cell towers respectively, resulting in a closely matching timestamps for both devices. The remaining third data source - the Meraki Mini is synchronized by recording the clock offset between the WiFi logging server and the iPhone. Using the offset, the WiFi timestamps are shifted appropriately, resulting in all three data sources matched in time.

Table 5.1: Test drive descriptions

1	Cambridge to Somerville. Mainly suburban areas with good WiFi coverage
2	Somerville to Brookline. Suburban areas with large WiFi gaps
3	Brookline to Downtown Boston. Dense urban areas with good WiFi coverage
4	Downtown Boston to Somerville. Mostly highway, with very large WiFi gaps
5	Cambridge to Somerville. Mainly suburban areas with occasional WiFi gaps

The test driving data is split into five separate drives each lasting between 10 and 20 minutes, with the split points picked to minimize gaps in WiFi coverage in the five

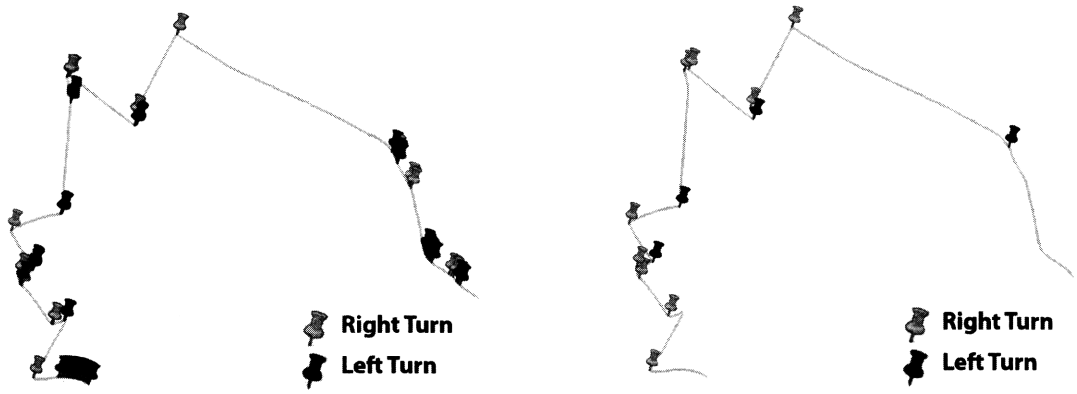
drives. The five drives are characterized in Table 5.1.

GPS traces for all five drives are used as ground truth in the evaluation.

5.2 Accelerometer Model Performance

Before evaluating the performance of the entire system, it is important to take a look at the performance of the individual components that make up the system. This section examines the performance of the Accelerometer Model.

The job of the Accelerometer Model is to identify when the vehicle is making a turn, using the accelerometer readings, and to score the particles based on how well their trajectories match the turn information. Figure 5-3 shows route turns for drive #5 identified by the Accelerometer Model, and turns extracted from the GPS trace.



(a) Turns identified by accelerometer (b) Turns identified by GPS

Figure 5-3: Accelerometer Model identifying turns and stops over a drive

To numerically assess the performance of the Accelerometer Model, the evaluation computes the percentages of intersections where the turns identified by the Accelerometer Model match those extracted from the GPS trace. For the evaluation, only intersections with more than two streets emanating are considered, since those are the only intersections at which a vehicle has a "choice" of where to go (i.e. turn left or right). Table 5.2 shows the numerical results for Accelerometer Model accuracy. First line shows the percentage of intersections where both GPS and the

Accelerometer Model report a turn in the same direction, out of all the intersections where turn is reported by the Accelerometer Model. The second line lists the percentage of intersections where both report no turn being made, out of all the intersections where Accelerometer Model reports no turn.

Table 5.2: Accelerometer Model performance at identifying route turns

Percentage of intersections where $G = t \mid A = t$	62%
Percentage of intersections where $G = f \mid A = f$	94%

Based on these results, the Accelerometer Model is very good at instructing the particle filter to not allow the particles to make turns when the vehicle maintains a straight trajectory.

5.3 WiFi Model Performance

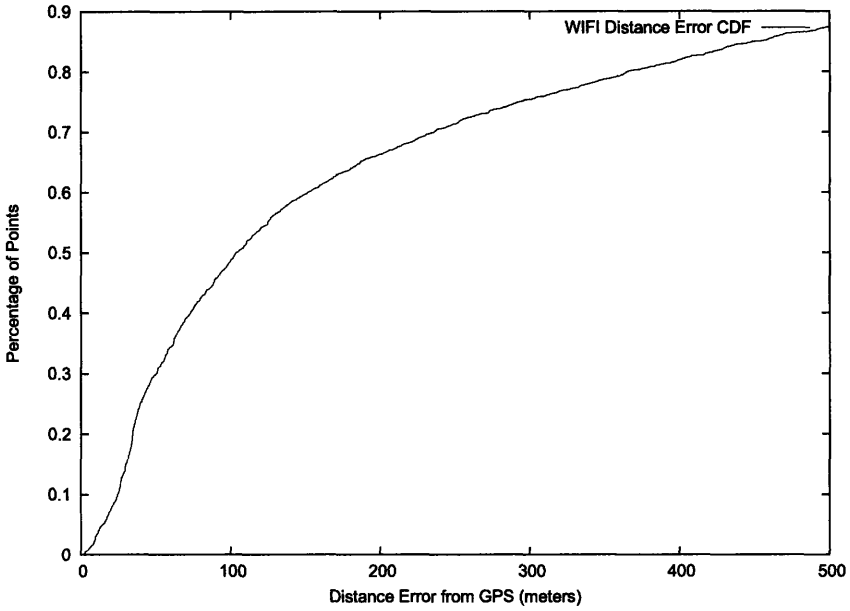
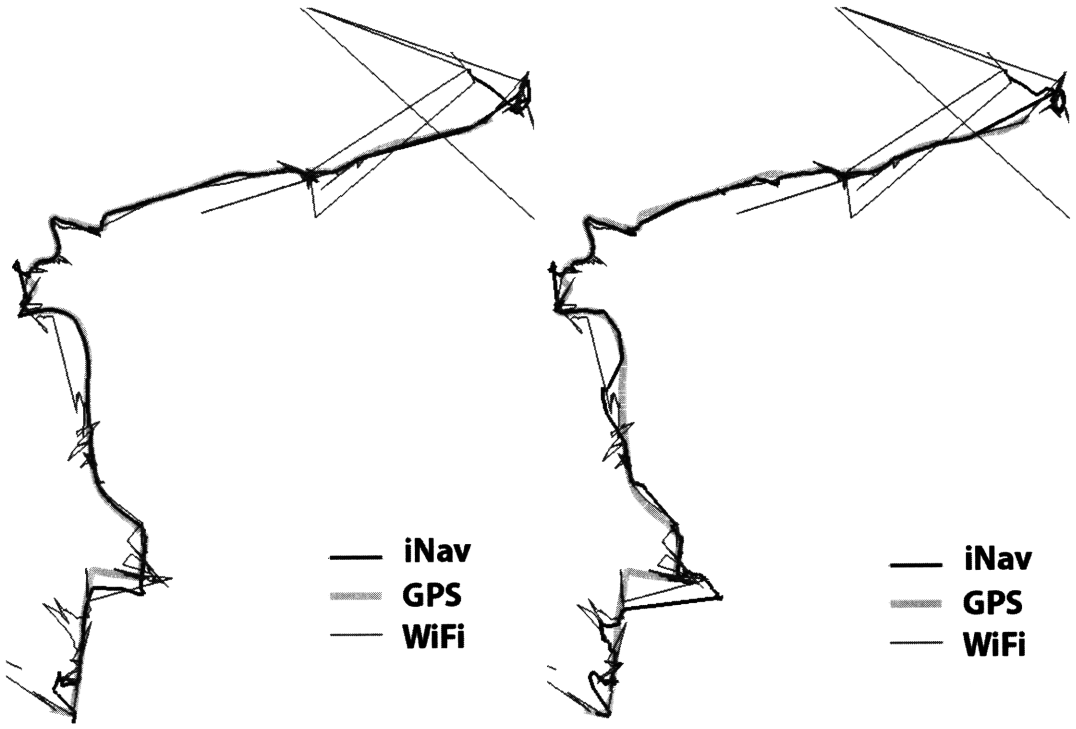


Figure 5-4: CDF of distance errors for WiFi Model location estimates

The WiFi Model scores particles based on how far away they are from the current single point estimate computed using the centroid algorithm. To evaluate the WiFi Model performance, distance errors between the centroid location estimates and the GPS location are computed for all WiFi observations. Figure 5-4 shows a CDF of distance errors for the location estimates produced by the centroid algorithm for all five test drives. The median error is 103.7 meters and the mean is 216.7 meters.

5.4 iNav Particle Filter Performance



(a) iNav with accelerometer

(b) iNav without accelerometer

Figure 5-5: iNav performance with and without accelerometer input

Having examined the performance of individual iNav components, this section now takes a look at the performance of the whole system. Figure 5-5 displays the output of iNav for drive #2, running with and without accelerometer input. In both configurations, iNav produces a trace that is much smoother than the centroid WiFi estimates, doing a better job at following the GPS trace. The output of iNav running

with the accelerometer input diverges less from the GPS trace, particularly in the areas with higher noise in WiFi centroid estimates.

For numerical evaluation, two separate metrics are used to examine iNav's performance. The first one is distance error from GPS - the same metric as the one used in previous section. While raw distance error is a good basic indicator of the system performance, for a number of applications, the accuracy of identifying the street on which the vehicle is currently traveling on is more important than the distance from true location itself. These applications include vehicle tracking, navigation aids, and traffic monitoring. Therefore the second metric used for system evaluation is the percentage of points in time when the system reported the correct street segment the vehicle was traveling on, as reported by GPS. Since a lot of the street segment in the map data are quite short, some degree of error in time is allowed when computing the second error. Instead of computing percentage of points where the iNav-reported street segment is the same as the one reported by GPS, the relaxed metric computes the percentage of points where the iNav-reported street segment was in the GPS trace within a fixed time window. For the evaluation a time window of 30 seconds was used.

Table 5.3: iNav average distance error with and without accelerometer input

Drive	**Median Error(w. Accel.)**	**Median Error(No Accel.)**
1	60.1m	98.9m
2	328.6m	334.2m
3	73.1m	73.3m
4	413.1m	261.3m
5	127.2m	148.8m

Figure 5-6 shows the distance error CDF for iNav over all of the test drives, and Table 5.3 lists distance error averages for individual drives. For evaluation, the particle filter was run with 10,000 particles. Note that these numbers are not directly comparable to the numbers from section 5.3, since iNav outputs location estimates every second, whether there is a WiFi observation or not, extrapolating the location when no WiFi data is available. The table shows that iNav performs considerably better in drives without large gaps in WiFi coverage, and indicates improvement in

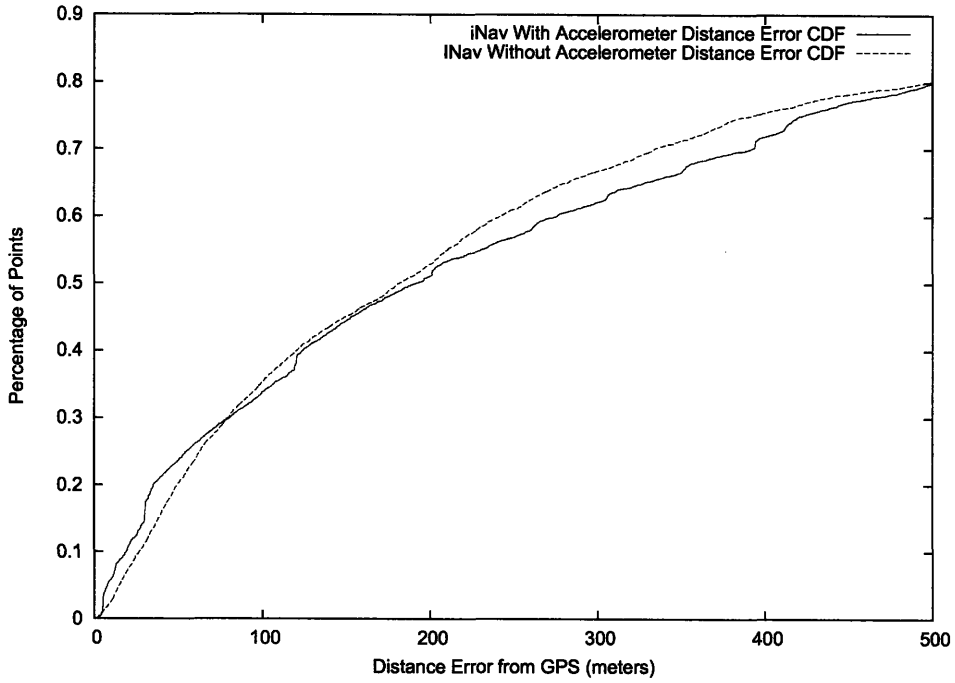


Figure 5-6: Relative performance of Particle Filter(WiFi Only), and Particle Filter(WiFi+Accelerometer)

performance for estimates calculated with accelerometer data for almost all of the drives. The only drive which shows a decrease in performance is drive #4, which has very large WiFi observation gaps, some lasting over 2 minutes.

The CDF plot of errors for all drives shows similar numbers for the system running with and without the accelerometer, although the accelerometer-assisted system makes smaller distance errors (<50 meters) more often. This fits well with the numbers seen in Table 5.3: the accelerometer input boosts system performance in areas with dense WiFi coverage, but degrades it when there is a large amount of WiFi noise or complete lack of WiFi signal.

A possible explanation for the degradation in system performance on drives with large WiFi gaps is that route features identified by the accelerometer become useless without the absolute location context provided by WiFi observations. Knowing that

a right turn was made is only useful when one can estimate which intersection the turn was made at. With a large WiFi gap it is possible that none of the particles have reached the true intersection yet, so the turn information reported by the accelerometer will be applied at a wrong intersection, leading the particle distribution away from the true vehicle trajectory.

Table 5.4 shows the results of the road segment identification accuracy metric for individual drives. Again, accelerometer input improves performance considerably for drives without large gaps in WiFi coverage.

Table 5.4: iNav road segment identification accuracy with and without accelerometer input

	% segments identified	
Drive	**With Accel.**	**No Accel.**
1	77.4%	39.0%
2	33.8%	25.0%
3	78.2%	50.4%
4	8.9%	15.9%
5	57.1%	39.0%

5.5 Resource Utilization on Low-Power Devices

This section examines runtime resource utilization of the system. Because iNav is targeted at low-power mobile devices like cell phones, the ability to perform in resource-limited environments is critical. Table 5.5 shows the main performance figures for the system running on an iPhone. The RAM requirements are very modest even for cell phones much less powerful than the iPhone, and running with 10,000 particles the system is still capable of stepping the particle filter over 3 times per second. The data put in phone's permanent storage includes training database and street map for the entire Greater Boston area. At 100MB, the storage space required is fairly large but acceptable for most of today's smartphones.

Table 5.5: iNav system resource utilization and runtime performance with 10,000 particles

Storage used	100MB
RAM used	1.5MB
Particle Filter performance	3.7 cycles / second

Chapter 6

Discussion

This chapter presents a discussion of the evaluation results from the previous chapter with respect to possible applications for iNav. Section 6.2 provides some practical considerations for large-scale deployment of a WiFi localization system such as iNav, followed by an analysis of directions in which future work on iNav can be done. The chapter concludes with a summary of contributions of this thesis.

6.1 Suitability for Location-Based Services/Applications

Possible applications that can be built on top of iNav's localization platform have been discussed in section 1.2.1. This section examines these applications given the system performance evaluation data from chapter 5.

While the average distance error for iNav location estimates is still too high for producing realtime turn-by-turn directions, the high percentage of correctly identified street segments still allows for significant navigational aids. For example, an iNav-based application would be capable of alerting the driver when a turn is coming up few blocks ahead.

Because iNav can with high probability identify the current street the vehicle is on, user interaction with the navigational software can be significantly simplified. In current interface of iPhone's Maps application the workflow for obtaining driving directions is following:

1. User hits the "Find My Location" button to get a WiFi-triangulated location estimate

2. User picks the starting point of the route one the map that is now centered around the location estimate

3. User picks the end point of the route and directions are computed

If the initial point of the route can be estimated with high accuracy, the first two steps can be eliminated, resulting in a much simpler workflow. iNav's performance in identifying the current street segment should allow just that.

The ability to report a street location rather than a simple $(latitude, longitude)$ pair is helpful in other applications as well. For vehicle tracking applications such as a system that reports the time estimates for when a shuttle/bus will arrive at its next stop can benefit from the street location estimate. Since shuttles typically travel on a pre-defined route, the location estimation algorithm's job is further simplified by having the search space restricted to just the vehicle's assigned route. A very cheap system for shuttle tracking can be built by simply putting a GPRS-capable phone with a WiFi radio on every shuttle that needs to be tracked.

The last application that was introduced in section 1.2.1 is urban traffic monitoring. In its current form iNav is not a good match for this application. While the system does a good job at identifying street segments that the vehicle is traveling on, these identifications are made within a fairly large time window. In other words, iNav is good at reporting information like "the vehicle has been on this street segment within 30 seconds from now", which makes it difficult to estimate vehicle speeds on each street segment. iNav's focus is doing the best job possible at estimating the most recent location of the vehicle, which is considerably different from the requirements of a traffic monitoring system. For a traffic monitoring system, an estimate of vehicle's recent path is more important. Such application would make a tradeoff between estimation latency and estimation accuracy, choosing accuracy over latency.

6.2 Practical Considerations for System Deployment

While the current implementation of iNav is functional within the Boston metropolitan area, creating a large-scale deployment of iNav that can provide a WiFi localization platform for multiple cities simultaneously would require considerable changes to the system architecture. This section discusses some of the practical considerations for creating a large-scale deployment of iNav.

The first issue with a large-scale iNav deployment is that with multiple metropolitan areas it is no longer feasible to fit all of the AP centroids/streets data on the mobile device itself. To address this problem, a scheme with over-the-air updates for on-device data can be designed. In such scheme, a central server would host the AP database for all of the APs in all of the markets covered by the system, and individual devices would make requests to the server to send AP data for the area in which the device currently is in. Since a very rough location estimate of the device location can be obtained even with just one WiFi observation, sending a single WIFi fingerprint to the server should be sufficient for the server to determine the general area in which the client device is located, and respond with AP database subset for that area.

With the central server in charge of carrying the AP database, the maintenance of the AP data becomes another issue to consider. The AP database must be updated as old access points get moved and new ones are added. Since GPS-labeled war-driving data is only available for the initial dataset, these updates to the AP database must be made based on the client WiFi data alone. To maintain the central access point database, the devices using the system would periodically send their WiFi traces to the central server. A self-organizing WiFi mapping scheme bootstrapped on a sparse GPS-labeled training set is possible, as described in [11]. A simplification of such scheme would be having the new WiFi fingerprints labeled with WiFi-estimated location instead of GPS location, and then added to the central AP database.

6.3 Future Work

The current implementation of iNav had demonstrated that accelerometer data can be used in conjunction with WiFi observations to improve location estimation accuracy. This section examines possible ways of further improving iNav performance, and discusses directions for future work.

6.3.1 Algorithm Improvements

Several improvements can be made in the algorithms used by iNav. Since particle filter performance is dictated by the accuracy of the sensor and transition models, improvements to either of those would be beneficial to the overall system performance.

The accuracy of the WiFi Model is difficult to improve without increasing the computational complexity of the algorithm. As discussed in section 2.2.2, the Fingerprinting algorithm could improve estimation accuracy at the expense of added computational cost. Alternatively, the estimation accuracy of the centroid algorithm can be raised by training it on a larger GPS-labeled dataset.

There are a number of ways in which the Accelerometer Model could be enhanced. Based on the evaluation numbers, the current implementation is very good at detecting when the car is on a straight trajectory, but its accuracy could be improved for identifying when turns are made. The current thresholding algorithm is a very simple approach to accelerometer data processing and it is possible that better results could be obtained with more sophisticated algorithms.

Additional information, such as vehicle speeding up or slowing down, should be possible to extract from accelerometer data, even if at very coarse granularity. Speed information can be used to further bias particle mobility model, improving estimation accuracy for estimates based on the accelerometer readings only (for example when passing through an area with sparse WiFi coverage).

6.3.2 Integrating Additional Data Sources

Integrating additional location information sources could be used to improve system estimation accuracy as well. Similar to WiFi, cell towers and Bluetooth devices act as radio beacons, and therefore can be used as location sensors. Integrating these additional sources with the existing system would require creating a sensor model for each additional location data source, and changing the particle scoring function in the particle filter to take into account the output of the new sensor model.

It is also possible to use a GPS module as an additional location sensor for the system. One of the possible motivations for doing so would be increasing location estimation accuracy and system availability compared to WiFi localization and GPS alone.

Finally, multiple devices with WiFi radios can be used for collaborative localization. With this setup, each device can use the WiFi observations made by other devices surrounding it as a separate location sensor. Same as with other additional data sources discussed, combining observations from multiple devices for particle scoring would require a separate sensor model for the additional observations.

Chapter 7

Conclusion

In summary, the major contributions of this thesis include designing and implementing a WiFi localization system capable of utilizing street map data and accelerometer input to improve estimation accuracy.

The created implementation of the system is specifically optimized for running on low-power devices such as cell phones. The embedded performance of the system is achieved in part by the proposed and implemented optimization of the resampling step in particle filter algorithm, which reduces the time complexity of the step from $O(N^2)$ to $O(NlogN)$.

Finally, this thesis evaluates the performance of accelerometer-assisted WiFi localization system on test drives with varying levels of WiFi coverage, and examines the suitability of accelerometer-assisted WiFi localization system for various location-based services and applications.

Bibliography

[1] Stumbler project http://code.google.com/p/iphone-wireless/.

[2] M. Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking.

[3] Paramvir Bahl and Venkata N. Padmanabhan. Radar: An in-building rf-based user location and tracking system.

[4] Yu-Chung Cheng, Yatin ChawatheAnthony, and LaMarcaJohn Krumm. Accuracy characterization for metropolitan-scale wi-fi localization.

[5] A. Doucet, N. de Freitas, and N. Gordon. Sequential monte carlo in practice.

[6] Jakob Eriksson, Hari Balakrishnan, Samuel Madden, and Lev Popov. Estimating road travel times using wifi.

[7] William Griswold, Robert Boyer, Steven Brown, Tan Minh Truong, Ezekiel Bhasker, Gregory Jay, and R. Benjamin Shapiro. Activecampus - sustaining educational communities through mobile technology.

[8] John Krumm and Eric Horvitz. Locadio: Inferring motion and location from wi-fi signal strengths.

[9] Andrew M. Ladd, Kostas E. Bekris, Algis Rudys, Guillaume Marceau, Lydia E. Kavraki, and Dan S. Wallach. Robotics-based location sensing using wireless ethernet.

[10] Anthony LaMarca, Yatin Chawathe, Sunny Consolvo, Jeffrey Hightower, Ian Smith, James Scott, Tim Sohn, James Howard, Jeff Hughes, Fred Potter, Jason Tabert, Pauline Powledge, Gaetano Borriello, and Bill Schilit. Place lab: Device positioning using radio beacons in the wild.

[11] Anthony LaMarca, Jeff Hightower, Ian Smith, and Sunny Consolvo. Self-mapping in 802.11 location systems.

[12] Julia Letchner, Dieter Fox, and Anthony LaMarca. Large-scale localization from wireless signal strength.

[13] Chen M., Sohn T., Chmelev D., Hahnel D., and Hightower J. Practical metropolitan-scale positioning for gsm phones. 2005.

[14] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system.

[15] Z. Shah and R.A. Malaney. Particle filters and position tracking in wi-fi networks.