Learning Motion Patterns from Sensor Data

Lin Liao
University of Washington,
Department of Computer Science
Seattle, WA USA
liaolin@cs.washington.edu

Abstract

Discovering people's motion patterns is useful in many applications of ubiquitous computing, robotics and artificial intelligence. In this project, we present a novel approach for parameterizing motion models based on the structure of the Generalized Voronoi Graph of the map. We discuss in detail the design of the motion model and the sensor model in a Voronoi-graph-based Bayes filter. We implement the Voronoi tracker using the techniques of particle filtering. We then demonstrate how to estimate the parameters of the motion model using an EM algorithm. Applications of this approach on both indoor and outdoor environments give encouraging results for Voronoi tracking and learning.

1 Introduction

It is often important to know the physical locations of people. In recent years, great progress has been made in location estimation and people tracking in the fields of ubiquitous computing, robotics and artificial intelligence [9; 16; 11]. However in many applications, it is not enough to just know a person's current position; we also want to predict a person's trajectory or destination in the near future and furthermore anticipate his needs [18; 5; 3]. This is possible if we are not only able to estimate the state of a person, but are also able to learn the person's motion patterns from his past experiences. Besides making predictions possible, understanding people's motion models is useful in several other ways. For example, mobile applications aware of users' common motion patterns may detect users deviating from their usual motion patterns and possibly getting lost [10]. In addition, as shown in this paper, the learned motion model may be used to improve tracking when the sensor measurements are sparse or noisy.

To discover motion patterns, we first need to figure out what a motion model looks like. This is not easy because even describing human motion behaviors in natural language is very complex. In fact, there seems to be no way to describe them precisely. Thus, the key here is how to simplify motion models reasonably. Our observation is that people's motion patterns usually rely heavily on the environment and therefore the map can give us great deal of valuable information. In this project, we present a novel approach to model humans' motion patterns utilizing the Generalized Voronoi Graph of the map [7] (also called simply Voronoi graph). This approach could be used in both indoor and outdoor environments. In an indoor environment, roughly speaking, the Voronoi graph is the skeleton of the free spaces, see Figure 1 **(a)**. In the outdoor environment, we can think the network of all the streets as the graph, as shown in Figure 3 **(a)**.

Using the Voronoi graph to parameterize the motion models has three key advantages. First, the Voronoi graph bridges the gap between continuous spaces and discrete abstract human motion behaviors. In our description of the motion model, the smallest unit is an edge in the Voronoi graph and we don't distinguish the behavior differences within the area covered by a single edge. Second, such a framework is expressive enough to describe many interesting motion behaviors. For example, the motion model can delineate the most likely direction taken when a person is at an intersection, or the place where a person usually stays, and even the bus stops where a person usually gets on and gets off, etc. Third, confining the state space within the Voronoi graph makes the implementation much more efficient, as we'll show in section 5.

After we determine the general form of the motion model, we need to figure out estimating the parameters quantitatively. An obvious difficulty here is that we cannot estimate the parameters of the motion model directly from the sensor data, since the motion model has effects on the sensor data only through humans' locations, while a person's real locations are not observable. One general approach of solving learning problems with missing features is the well known EM (Expectation-Maximization) algorithm [6]. Using an EM algorithm in our system, we first estimate the trajectory distribution from the sensor readings, then we update the parameters to maximize the likelihood of the trajectory distribution. Notice that the first step is in fact a tracking or localization problem and has been well studied. Particle filtering is one of the most popular approaches. We present a variant of particle filtering that restricts all particles onto the Voronoi graph. We call it Voronoi tracking. Thus, by combining EM learning with Voronoi tracking, we are able to completely accomplish unsupervised learning of the parameters.

In the next section, we give an overview of related work. Then we discuss Voronoi tracking and parameter estimation in sections 3 and 4 respectively. In section 5, we describe the applications of our approach in indoor environment and outdoor environment. At the end, we state our conclusions and discuss future work.

2 Related Work

2.1 Bayesian Filtering and Particle Filtering

Bayesian filtering provides a powerful framework for estimating the state of a dynamical system from sensor measurements and motion model (or control data) [1]. Within this framework, if we suppose the Markov assumption, i.e. the future state is conditionally independent from the past state given the current state, the posterior belief of state x_t at time t can be computed from the previous sate x_{t-1} recursively using the following update rule:

$$p(x_t \,|\, z_{1:t}) \propto p(z_t \,|\, x_t) \int p(x_t \,|\, x_{t-1})\; p(x_{t-1} \,|\, z_{1:t-1}) dx_{t-1} \tag{1}$$

Here $z_{1:t}$ is the history of all sensor measurements obtained up to time t. $p(x_{t-1} \,|\, z_{1:t-1})$ is the posterior belief of the state at time $t-1$. When applying Bayesian filtering to a concrete domain, three key issues have to be addressed:

1. The description of the state x_t. For example, in the context of location estimation, the state x_t typically describes the position and velocity of the object.
2. The probabilistic model of the object dynamics, i.e. motion model. Motion model corresponds to the distribution of $p(x_t \,|\, x_{t-1})$ in equation (1).
3. The sensor model, i.e. the likelihood of a sensor measurement z_t given the state x_t, which corresponds to $p(z_t \,|\, x_t)$ in equation (1). Bayes filtering has a number of variants with each has a different representation of the belief, e.g. Kalman filtering, multi-hypothesis tracking, grid-based representation, etc.

Particle filtering [8] is another variant of Bayesian filtering, which approximates the posterior belief $p(x_t \,|\, z_{1:t})$ using a set of weighted samples:

$$\{\langle x_t^{(i)}, w_t^{(i)} \rangle \mid i = 1, \ldots, n\}$$

Here each $x_t^{(i)}$ is a sample (or state), and the $w_t^{(i)}$ are non-negative numerical factors called *importance weights*, which sum up to one. Particle filters apply the recursive Bayes filter update to estimate posteriors over the state space. The basic form of the particle filter updates the posterior according to the following sampling procedure, often referred to as sequential importance sampling with re-sampling (SISR, see also [8]):

Re-sampling: Draw with replacement a random sample $x_{t-1}^{(i)}$ from the sample set S_{t-1} according to the (discrete) distribution defined through the importance weights $w_{t-1}^{(i)}$.

Sampling: Use $x_{t-1}^{(i)}$ to sample $x_t^{(j)}$ from the distribution $p(x_t \mid x_{t-1})$. $x_t^{(j)}$ now represents the density given by the product $p(x_t \mid x_{t-1})p(x_{t-1} \mid z_{1:t-1})$. This density is the so-called *proposal distribution* used in the next step.

Importance sampling: Weight the sample $x_t^{(j)}$ by the importance weight $p(z_t \mid x_t^{(j)})$, the likelihood of the measurement z_t given the state $x_t^{(j)}$.

Each iteration of these three steps generates a sample drawn from the posterior density. After n iterations, the importance weights of the samples are normalized so that they sum up to 1. It can be shown that this procedure in fact approximates the Bayes filter update (1), using a sample-based representation [8].

2.2 Expectation-Maximization (EM) Algorithm

The EM algorithm [6] is a general method of finding the maximum-likelihood estimate of the parameters of an underlying distribution from a given data set when the data is incomplete or has missing values. Suppose Z is the set of observations and X is the set of data that cannot be observed, then the goal of EM algorithm is to maximize the complete-data likelihood $p(X, Z \mid \Theta)$ by adjusting the parameter Θ. The EM algorithm is an iterative algorithm and each iteration has an E-step and a M-step. In the E-step, we estimate the posterior distribution of the missing data and compute the expected value of the complete data log-likelihood $\log p(X, Z \,|\, \Theta)$ given the observed data and the current parameter estimates Θ^{i-1}. In the second step, the M-step, the algorithm finds the optimized Θ that maximizes the expectation computed in the E-step. Each iteration is guaranteed to increase the log-likelihood and the algorithm is guaranteed to converge to the local maximum of the log-likelihood function.

When applying the general EM algorithm discussed above to Hidden Markov Models, we get the widely-used Baum-Welch algorithm [6; 14]. Suppose we have a hidden Markov process with N states and T time slots. Baum-Welch algorithm updates the state transition probabilities iteratively using the following formula:

$$\begin{aligned} \overline{a}_{ij} &= \frac{\text{expected transitions from state } S_i \text{ to } S_j}{\text{expected transitions from state } S_i} \\ &= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \xi_t(i,j)} \end{aligned} \tag{2}$$

where $\xi_t(i,j)$ is the probability of being in state S_i at time t and in state S_j at time $t+1$.

Given the model and the observation sequence, we have:

$$\begin{aligned}\xi_t(i,j) &= P(x_t = S_i, x_{t+1} = S_j \,|\, z_{1:t}, \Theta) \\ &\propto \alpha_t(i) a_{ij} b_j(z_{t+1}) \beta_{t+1}(j) \end{aligned} \quad (3)$$

where $\alpha_t(i)$ and $\beta_{t+1}(j)$ are the forward variable and backward variable respectively. Refer to [14] for more details. Intuitively, $\alpha_t(i)$ is the amount of evidence from observations up to time t that supports $s_t = S_i$; $\beta_{t+1}(j)$ is the amount of evidence from observations after time $t+1$ that supports $s_{t+1} = S_j$; and $b_j(z_{t+1})$ is the support from z_{t+1}. Thus the Baum-Welch algorithm includes a forward pass and a backward pass where the observations are integrated from forward or from backward.

2.3 Motion Pattern Learning

Our goal of learning motion models is very similar to the goals in [5; 3]; [5] learns the motion patterns using the data collected by the laser range sensors, and [3] learns them using GPS sensor data. Our approach of solving the problem is very different from theirs in a number of ways. Essentially, in both of their work, the problem of learning motion models is reduced to the problem of trajectory clustering. That is, they first collect training data by tracking the moving objects. The tracking is totally separated from learning, and the estimation errors during tracking are ignored. Then they cluster similar trajectories as motion patterns. Their approach does not need a map, but requires a lot of training samples. Instead, we start from the map of the environment and we regard the learning problem as a parameter estimation problem. In our approach, the tracking is one part of learning and we can use the learned model to reduce tracking errors. Moreover, in our approach the learning could be performed even without enough training data and the learned model can be further improved as new data arises.

3 Voronoi Tracking

In this section, we describe our approach of applying Bayesian filtering on a Voronoi graph. We first specify the state representation, motion model and sensor model. Then we discuss the implementation based on particle filtering. The concrete specification of Voronoi tracking may be a little different from application to application. Our discussion here is based on the scenario of indoor tracking (see section 5.1).

3.1 State Description

The state x_t of an object is represented as $\langle e, d, v, m \rangle$, where e denotes the current edge on the graph, d indicates the distance of the object from the start vertex of edge e, v is a non-negative number and indicates the velocity of the object, and $m \in \{\text{stopped}, \text{moving}\}$ indicates the current motion mode of the object. Note that we treat the Voronoi graph as a directed graph where each undirected edge is regarded as two directed edges. Thus an edge e also includes the motion directions of the object.

3.2 Motion Model

The motion model $p(x_t \mid x_{t-1})$ has to take into account that the objects are constrained to moving on the graph. As we have mentioned, the smallest unit for the motion model is an edge in the graph. Let G=(V,E) be the Voronoi graph where V is the set of vertexes and E is the set of directed edges. We divide the motion model into three components:

1. For each edge $e_i \in E$, we have:

$$\begin{cases} p(m_t = \text{moving} \,|\, m_{t-1} = \text{moving}, e_i) \\ p(m_t = \text{stopped} \,|\, m_{t-1} = \text{moving}, e_i) \\ p(m_t = \text{moving} \,|\, m_{t-1} = \text{stopped}, e_i) \\ p(m_t = \text{stopped} \,|\, m_{t-1} = \text{stopped}, e_i) \end{cases} \quad (4)$$

These four parameters (in fact, only two out of the four are independent) determine the mode transition distribution.

2. For each edge $e_i \in E$, we have:

$$\begin{cases} p(e_j \,|\, e_i) > 0 & \text{if } e_j \text{ is } e_i\text{'s neighboring edge} \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

Given the object is leaving edge e_i, this group of parameters determines the distribution of the following edge.

3. We suppose the velocity distribution in the whole environment is a Gaussian distribution $\mathcal{N}(v_{mean}, \sigma_v)$ where v_{mean} is the average velocity and σ_v is the standard deviation of velocity.

The distribution of $p(x_t \,|\, x_{t-1})$ can be factorized as:

$$\begin{aligned} p(x_t \,|\, x_{t-1}) &= p(e_t, d_t, v_t, m_t \,|\, e_{t-1}, d_{t-1}, v_{t-1}, m_{t-1}) \\ &= p(m_t \,|\, m_{t-1}) p(v_t \,|\, v_{t-1}, m_t) \\ &\quad \cdot p(e_t, d_t \,|\, e_{t-1}, d_{t-1}, v_t, m_t) \end{aligned} \quad (6)$$

$p(m_t \mid m_{t-1})$ is available directly from the motion model.
$p(v_t \,|\, v_{t-1}, m_t)$ is computed as:

$$p(v_t \,|\, v_{t-1}, m_t) = \begin{cases} \delta(v_t, 0) & \text{if } m_t = \text{stopped} \\ \mathcal{N}(v; v_{mean}, \sigma_v) & \text{if } m_t = \text{moving and} \\ & v_t = w_1 v_{t-1} + w_2 v \end{cases} \quad (7)$$

where $\delta(v_t, 0)$ is the Dirac delta function which is 1, if $v_t = 0$ and 0 otherwise. w_1,w_2 are both non-negative numbers and sum to 1. That is, $v_t = 0$ if the mode is "stopped", otherwise it is a linear combination of the v_{t-1} and the random velocity obtained from the motion model. The weights w_1 and w_2 are set manually.

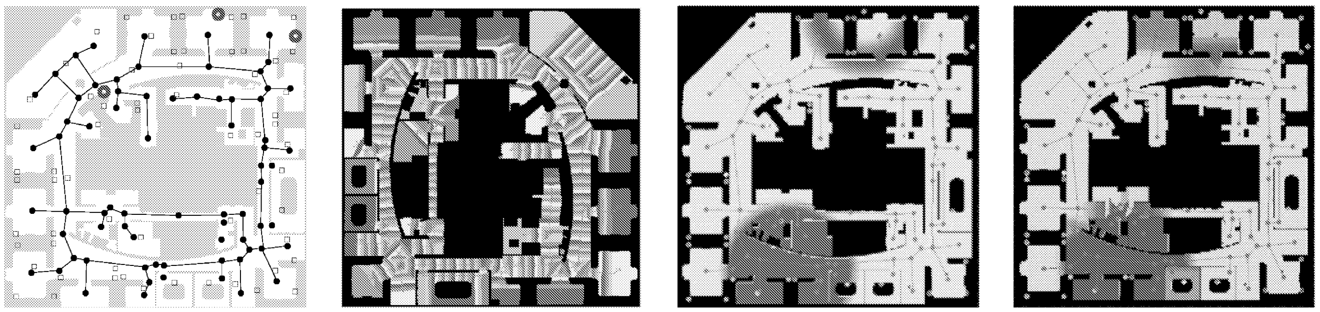

Figure 1: *Voronoi graphs for location estimation: (a) Indoor environment along with manually pruned Voronoi graph. Shown are also the positions of ultrasound Crickets (circle) and infrared sensors (square). (b) Patches used to compute likelihoods of sensor measurements. (c) Likelihoods of an ultra-sound cricket reading (upper) and an infrared reading (lower) in the full state space and (d) The likelihoods in (c) projected onto the Voronoi graph patches.*

$p(e_t, d_t \mid e_{t-1}, d_{t-1}, v_t, m_t)$ is more complex, and is computed as follows:

$$p(e_t, d_t \,|\, e_{t-1}, d_{t-1}, v_t, m_t) = \begin{cases} \delta(e_t, e_{t-1})\delta(d_t, d_{t-1}) & \text{if } m_t = \text{stopped} \\ \delta(e_t, e_{t-1})\delta(d_t, d_{t-1} + v_t\Delta t) & \\ & \text{if } d_{t-1} + v_t\Delta t \le |e_{t-1}| \\ p(e_t \,|\, e_{t-1})\delta(d_t, d_{t-1} + v_t\Delta t - |e_{t-1}|) & \\ & \text{if } d_{t-1} + v_t\Delta t > |e_{t-1}| \end{cases} \quad (8)$$

where Δt is duration of the time slot and $|e_{t-1}|$ is the length of edge e_{t-1}. That is, if m_t =stopped, the object will stay at the same position. Otherwise, if the distance it moves, i.e. $v_t\Delta t$, is less than its distance to the end of the edge, the object will stay on the edge and we only need to update d_t. Otherwise, the object will travel to another edge with transition probability $p(e_t \,|\, e_{t-1})$.

In summary, for a given motion model and current state, we can compute the next state based on equations (4)- (8).

3.3 Sensor Model

Given that a sensor has a measurement z at time t, the likelihood of $p(z \,|\, x)$ is a function of x. For different kinds of sensors, this function takes quite different formats. In Figure 1 **(c)**, the probabilistic models of ultrasound Crickets and infrared badges are shown.

In the scenario of Voronoi tracking, the state has the format $x =< e, d, v, m >$. One simple way to computing the likelihood is to compute the $2D$ position of x and apply the sensor model only to that point. But this method may be problematic because a single point on the Voronoi graph may not be a good representation of the space it covers. Hence, to compute the likelihood of $p(z \,|\, x)$, we compute the average likelihood of all the positions projected onto (e, d). That is:

$$\begin{aligned} p(z \,|\, x) &= p(z \,|\, e, d) && (9) \\ &= \int_{\nu \in \mathcal{S}(e,d)} p(z \mid \nu)\, p(\nu \mid e, d)\, d\nu && (10) \end{aligned}$$

Here, (9) follows from the fact that in our scenario the velocity and mode of moving objects does not affect sensor measurements. $\mathcal{S}(e, d)$ denotes the set of all positions ν projected onto (e, d). In our implementation of the sensor model, we discretized positions on the graph, which results in location *patches* $\mathcal{S}(e, d)$, as illustrated in Figure 1 **(b)-(d)**.

3.4 Particle Filtering-based Voronoi Tracking

The application of particle filters to location estimation on a Voronoi graph is rather straightforward. The resampling step does not have to be changed at all. Sampling the motion update has to be done according to equations (4)-(8). To be more specific, we first sample motion mode m_t based on $p(m_t \,|\, m_{t-1})$ in (4). If m_t ="stopped", we set $v_t = 0$. Otherwise, we sample new velocity v_t based on (7). After v_t is determined, we compute the distance s it moves as $s = v_t\Delta t$. For this distance s, we have to determine whether the motion along the edge results in a transition to another edge. If not, then $d_t = d_{t-1} + s$ and $e_t = e_{t-1}$. Otherwise, $d_t = s - |e_{t-1}| + d_{t-1}$ and the next edge e_t is drawn with probability $p(e_t \mid e_{t-1})$. After these sampling steps, the resulting states are distributed according to $p(x_t \mid x_{t-1})$. The importance sampling step of the particle filter is implemented by weighting each sample proportional to the projected observation likelihood as given in (10).

4 Parameter Estimation

As discussed in the introduction, we use the EM algorithm to handle the situation where we don't know the real trajectories. In this section, we demonstrate how to apply the EM algorithm in our domain. We first derive the formula of estimating the parameters of the motion model, and then we describe an efficient implementation.

4.1 E-Step:

In the E-step, we update the posterior distribution over the trajectory of the person and compute the expectation of the log-likelihood. We define:

$$\begin{aligned} & Q(\Theta, \Theta^{(i-1)}) && (11) \\ = \; & E[\log p(z_{1:t}, x_{1:t} \mid \Theta) \mid z_{1:t}, \Theta^{(i-1)}] \\ = \; & \int_{x_{1:t}} \log p(z_{1:t}, x_{1:t} \mid \Theta) p(x_{1:t} \mid z_{1:t}, \Theta^{(i-1)}) dx_{1:t} && (12) \end{aligned}$$

Here $x_{1:t}$ and $z_{1:t}$ are the states and observations, respectively. Θ are the parameters of the Voronoi graph-based model we want to estimate and $\Theta^{(i-1)}$ are the estimation thereof at the $i-1$-th iteration of the EM algorithm. The difficulty here is to estimate the posterior distribution over state trajectories $x_{1:t}$ given $z_{1:t}$ and $\Theta^{(i-1)}$. In our scenario, it is not possible to find a closed form solution to this estimation problem. Therefore we resort to approximate approaches. Observe that when we do particle filtering using the motion model with parameter $\Theta^{(i-1)}$, the particle distribution at each time t along with the history of particles is an approximation for $p(x_{1:t} \mid z_{1:t}, \Theta^{(i-1)})$. Therefore, (12) can be rewritten as

$$Q(\Theta, \Theta^{(i-1)}) \approx \frac{1}{m} \sum_{j=1}^{m} \log p(z_{1:t}, x_{1:t}^{(j)} | \Theta), \qquad (13)$$

where m is the number of particles and $x_{1:t}^{(j)}$ is the state history of the j-th particle. For simplicity, we assume that all the particles have equal weight, *i.e.* after they are resampled. It is straightforward to extend our derivation to the case of different weights.

Our approach is in fact a direct extension of the Monte Carlo EM algorithm [19]. The only difference is that we allow particles to evolve with time. It has been shown that when m is large enough, Monte Carlo EM estimation converges to the theoretical EM estimation [12].

4.2 M-step:

The goal of the M-step is to maximize the expectation we computed in the E-step by updating the parameter estimations. From (13), we have:

$$\begin{aligned} \Theta^{(i)} &= \underset{\Theta}{\operatorname{argmax}}\, Q(\Theta, \Theta^{(i-1)}) \\ &= \underset{\Theta}{\operatorname{argmax}} \sum_{j=1}^{m} \log p(z_{1:t}, x_{1:t}^{(j)} \mid \Theta) \\ &= \underset{\Theta}{\operatorname{argmax}} \sum_{j=1}^{m} (\log p(z_{1:t} | x_{1:t}^{(j)}) + \log p(x_{1:t}^{(j)} | \Theta)) && (14) \\ &= \underset{\Theta}{\operatorname{argmax}} \sum_{j=1}^{m} \log p(x_{1:t}^{(j)} \mid \Theta) && (15) \end{aligned}$$

Here, (14) follows from the independence condition $p(z_{1:t} \mid x_{1:t}^{(j)}, \Theta) = p(z_{1:t} \mid x_{1:t}^{(j)})$, *i.e.* observations are independent of model parameters if the state trajectory is known. This result shows that we only need to update the parameters to maximize the sum of the likelihood of each particle history.

One point worth mentioning is that our derivation of combining the EM algorithm with particle filtering does not require a specific parameterization of the motion model. Although we only implement such an algorithm over the Voronoi based motion model, it is also applicable to other parameterizations.

4.3 Implementation

Directly evaluating equation (15) requires a particle smoothing operation which usually requires storing the history of each particle. Therefore it is often too expensive. In practice, we avoid direct smoothing by performing a forward particle filtering and backward particle filtering step. Then we multiply the two distributions at corresponding time slices, which corresponds to the Baum-Welch algorithm as discussed in section 2.2. This is very efficient since at each time we only save the aggregate information for each edge, not for each particle.

We now discuss the implementation in more details. Recall that the parameters Θ of the model consist of the transition probabilities $p(e_i \mid e_j)$ on the Voronoi graph, the mode switching parameters $p(m_t \mid m_{t-1}, e_i)$ of the motion model, and the Gaussian velocity parameters (v_{mean}, σ_v^2). For the first E-step, we initialize $\Theta^{(0)}$ with some reasonable values using background knowledge of typical human motion and a uniform distribution for the outgoing edges at each vertex of the Voronoi graph. Then, we collect data in the environment and estimate the position of the person using the initial model. Time is discretized in intervals of equal size Δt. After each time step, we determine $e_{ij}(t)$, the number of particles that move from one edge e_i to another edge e_j. We also generate counts for $m_{ijk}(t)$, the number of particles on edge e_i that switch from one motion state m_j into another other motion state m_k. The reason for learning different switching models for each edge is that we want to be able to determine where a person typically stops for extended periods of time. To estimate the parameters of the Gaussian motion model, we simply have to store the velocities of all samples that are in the "moving" state. In the backward pass, the same values are computed when localizing the person backward in time, *i.e.* we proceed backwards through the data log. The final values of the E-step are then given by multiplication of the estimates of the forward and backward pass at each time slice. To maximize the parameter set $\Theta^{(1)}$, the M-step essentially converts the frequency counts obtained in the E-step into probabilities. The parameters of the Gaussian motion model are the mean and variance of the velocity values. In the next iteration of EM, these new parameters are used to re-estimate the expectations using forward backward localization. The algorithm stops if $\Theta^{(i)}$ and $\Theta^{(i-1)}$ are close enough.

This idea of forward and backward passes come from

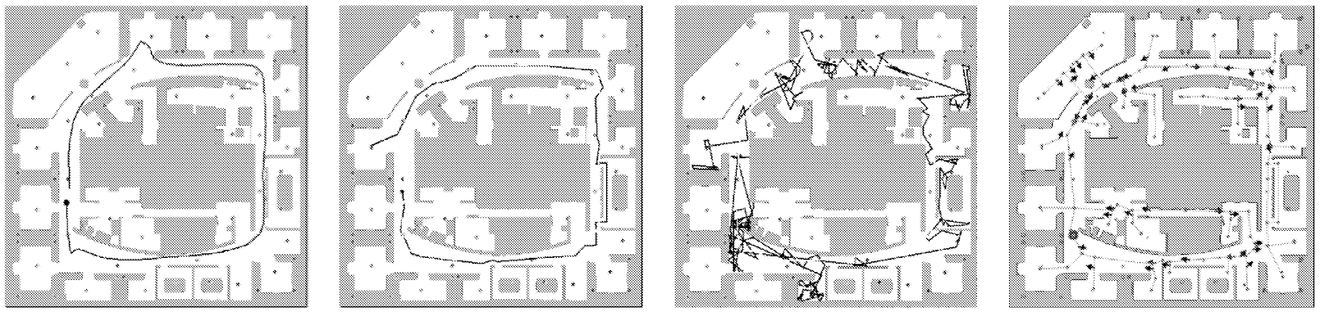
(a) (b) (c) (d)

Figure 2: *(a) True trajectory of the moving robot. Shown only a small part of the data log we collected. (b) The most likely trajectory estimated using Voronoi tracking (c) The most likely trajectory estimated using particle filters. (d) Transition model learned using EM. Shown are only those transitions for which the probability is above 0.7*

the forward-backward variables in the Baum-Welch algorithm (see section 2.2).

5 Experiments

We evaluate our approach in two environments with very different scales. The first environment is indoor 30m by 30m office area with a number of ultrasound sensors and infrared sensors installed [1]. The second environment is a urban area where we have a collection of GPS sensor readings of a graduate student's outdoor activity episodes [2]. The experiments demonstrate the encouraging performance of our approach in both environments.

5.1 Indoor Environment Experiment

This experiment is based on data recorded at the Intel Research Lab Seattle. This office is 30 meters by 30 meters. It is equipped with two different kinds of id sensors: seventy-three Versus infrared receivers which solely provide id information, and three Cricket ultrasound receivers, which provide identity and distance estimates [17; 13], as shown in Figure 1 **(a)**.

To generate data for which ground truth is available, we equipped a Pioneer IIDX robot with two Versus badges, a Cricket beacon, and additionally with a Sick laser range-finder. Our experiment is based on a log of sensor measurements received while driving the robot through the environment at an average velocity of 30 cm/s. During the experiment, we only stopped the robot at designated resting places. The laser range-finder allowed us to accurately estimate the path of the robot using the map which is regarded as the ground truth of robot locations. Note that we only use the ground truth as a way to evaluate our approach, the algorithm itself is not aware of that information. The complete data log was split into a training set of 25 minutes and a test set of 10 minutes of robot motion.

The goal of this experiment is two-folded. First we want to compare Voronoi tracking with "classical" particle filters on tracking moving objects before any learning. Second, we want to verify whether our learning algorithm is able to give us reasonable results.

The particle filter that is used for comparison shares the same motion model and sensor model with Voronoi tracker with only two differences. The first is that its motion model has no idea of the structure of Voronoi graph. The particles will move across the whole free space and changes moving directions with some random Gaussian probability. The second difference is that the sensor model computes the probability over the whole free space and will not project it onto patches (see Figure 1 **(c)**).

The metric we use for comparison is the average localization error, which is defined as the average distance from each particle to real locations. The resulting error on the test data was 2.586m for the Voronoi tracker and 3.230m for the particle filter. This result is encouraging since it indicates that the projection onto the Voronoi graph may even yield better tracking performance. Figure 2**(a)**-**(c)** shows a segment of trajectory and the tracking results using the Voronoi tracker and particle filter, respectively. It is obvious from the graph that Voronoi tracker can give a much more stable estimation of the most likely trajectory.

Therefore, even without learning, Voronoi tracking outperforms the particle filter. The main reason is that the Voronoi-graph-based motion model is closer to the real motion model of moving objects in most cases.

Another advantage of Voronoi tracker is that it needs much less particles than the particle filter, since the moving space is much smaller.

Next, we evaluated the learning performance of Voronoi tracker. To do so, we trained the Voronoi motion model using the EM algorithm described in Section 4 on the training set. The results are summarized in the table below.

[1]This application is implemented on top of the infrastructure of Location Stack. http://portolano.cs.washington.edu/projects/location/.

[2]This application is part of the project Activity Compass http://www.cs.washington.edu/homes/djp3/AI/AssistedCognition/ActivityCompass/ and is implemented together with Don Patterson.

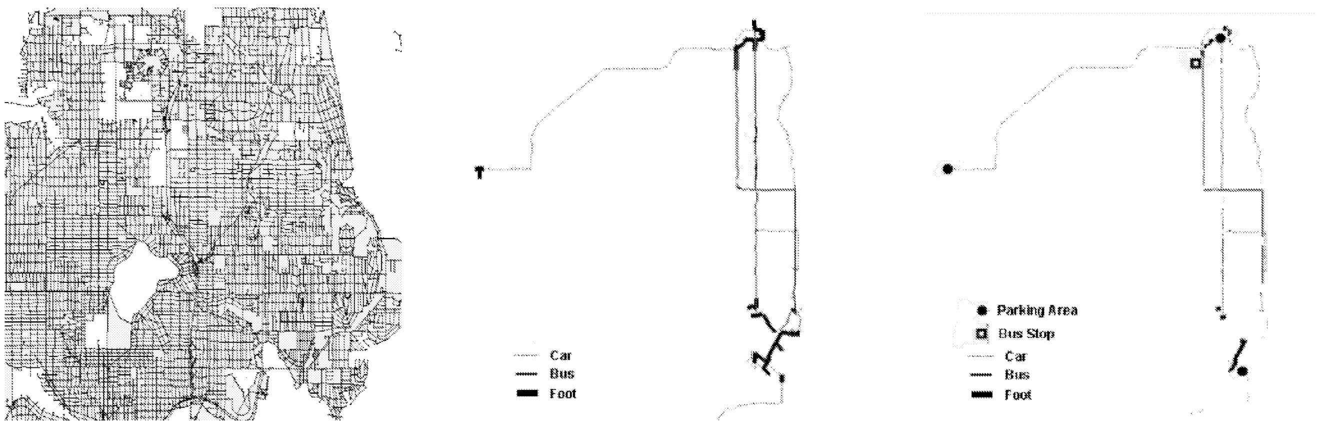

Figure 3: *The three figures are in the same scale. For clarity, the street map is not displayed in (b) and (c). (a) Street map of northern Seattle where most of the episodes in our data log happened. (b) One training data log. Different transportation modes are displayed using different types of lines, as explained in the legend. Note there are overlaps between paths of the car and the bus that are not distinguishable from the figure. (c) Motion model after learning, including the edge transitions and transportation mode transitions. Shown only those transition with probabilities above* 0.7.

EM Iteration	Tracking Errors	Avg. Change
before learning	2.586	
1	1.888	0.235
2	1.787	0.083
3	1.530	0.075
4	1.538	0.054

The middle column is the average localization error over the training set for the different iterations of EM. As can be seen, the algorithm converges after only 3 iterations and the average localization error decreases significantly. This fact is supported by Figure 2 **(d)**, which shows the transition model of the Voronoi graph after learning (shown are only transitions with probability above 70%). The plot demonstrates that the motion model did learn transitions along the main path of the robot. One point worth attention is that at some places, e.g. the upper left room in the map, our result shows some biased transitions although the robot has never been there. The reason is that since the sensor measurements are very noisy and error-prone, there may be some side effects although the forward-backward passes help reduce the effects dramatically.

5.2 Outdoor Environment Experiment

In the experiment, we use a data set which consists of logs of GPS data collected by another graduate student. The data contains position and velocity information at 2-10 second intervals during periods of time in which he was moving outdoors. From the data set, we choose 29 episodes representing a total of 12 hours of logs. This subset consists of all of portions of the data set which were bounded by GPS signal loss, had no intermediate loss of signal for more than 30 seconds, and contained a change in the mode of transportation at some point in the episode. These episodes were divided chronologically into three groups which formed the sets for three-fold cross-validation for our learning. Figure 3 **(a)**,**(b)** show the street map and one training data set.

In this scenario, each state not only includes the position, velocity information, but also include the mode of transportation: foot, car or bus. The transportation information is important for delineating the outdoor motion patterns and is not available from the GPS data. The motion model also must change accordingly to take into account the transition between transportations. Since the GPS data also contains velocities and moving directions, we modify our sensor model to integrate this information as well.

In this experiment, since GPS provides fairly accurate position information most of the time, we are more interested in the tracking of transportations. To do that, we label all the data manually with the real transportation and at each time we compare the most likely mode from the trackers with the real mode. For comparison, we use 3 different approaches. The first is a decision tree classifier which classifies the modes based on the velocities. The second is the Voronoi tracking algorithm which has no clue of bus stops and car locations. The third is the similar to the second exempt we include bus stop and car location information into our model.

The results are summarized in the table below.

Model	Cross-Validation Accuracy
Decision Tree	55%
Voronoi Tracking w/o bus stops and car locations	60%
Voronoi Tracking w/ bus stops and car locations	80%

The result shows that our approach can track well when using the bus stops and car locations information. Note that 80% is in fact quite robust given the fact that often a change of transportations cannot be detected instantaneously and each episode in our data set contains at least one transportation mode change.

Next, we show the learning results and the location prediction capabilities of the learned model.

One training data set and the corresponding model after learning is shown in Figure 3. Comparing **(b)** and **(c)** in Figure 3, it is obvious that our motion model captures most of the edge transition probabilities correctly. What is more interesting, it also correctly figure out most of the places where transportation switches happened. For example, as shown in the figure, the learned model correctly contains the 3 parking areas including the ones at home and campus. It also recognize the bus stop where the student gets on the bus to campus.

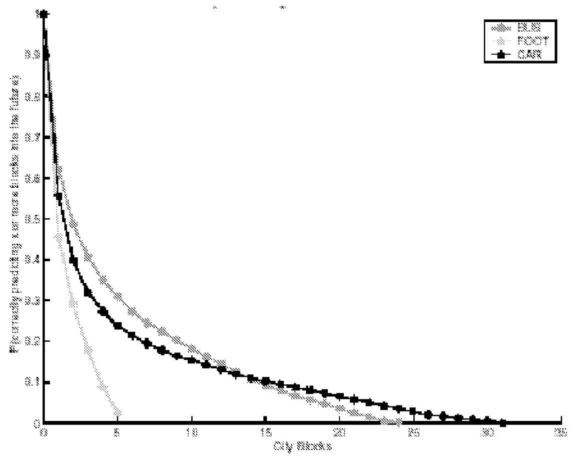

Figure 4: *Prediction capability of the learned model*

The prediction capabilities of our approach are shown in Figure 4. Here, the learned model was used to predict the location of the person into the future. This was done by providing the ground truth location to the algorithm and then predicting the most likely path based on the transition probabilities learned from the training data. The figure shows the percentage of trajectories that were predicted correctly, given different prediction interval. Prediction length was measured in city blocks. For example, in 50% of the cases, the location of the person was predicted correctly for 2.5 blocks when the person was on the bus. In 30% of the cases, the prediction was correct for 5 blocks, and 15 blocks were predicted correctly in 10% of the cases.

6 Conclusions and Future Work

In this paper, we presented a novel approach to parameterize motion models based on the Voronoi graph of the map. Modeling motion model in this way has 3 key advantages: 1) It bridges the gap between continuous space and discrete motion behaviors by naturally approximating real motion patterns. 2) It is expressive enough to represent many useful motion patterns. 3) It reduces the state space and can be implemented efficiently.

We discussed in detail the design of a Voronoi graph-based Bayes filtering and its implementation using particle filter. We also presented a parameter estimation algorithm based on the EM learning framework. Such a learning algorithm also has three advantages: 1) It is an unsupervised algorithm in that we don't need to know the ground truth during the training stage. Thus it can be more useful in practice. 2) It takes into account the tracking errors in a natural way. The tracking process is in fact one step in the learning and the learned model can be used to reduce tracking errors. 3) Our algorithm for learning motion model does not require a specific parameterization. It can be applied to any motion model specifications besides Voronoi graph.

We implemented our system in both indoor and outdoor environments. Through the experiments, we showed that:

1. Our approach is by far more efficient than the particle filters which estimate the locations in the complete free spaces. Using the Voronoi-graph-based motion model and sensor model, it needs less particles and gives better tracking accuracy.
2. Our unsupervised learning algorithm is able to discover correctly the motion patterns when fed the real data log. It can capture the edge transition, mode transition and velocity patterns in our experiments.
3. Our approach scales well. It performs well in both a small office area and a large urban area.
4. Our system is relatively easy to implement. Particle filtering is one of the easiest implementations for Bayesian filtering. The tracking is also part of the learning which allows us to reuse the tracking module when implementing learning.

We can extend our work in a number of directions, including the following.

1. Although we have tested our approach using real data, it could be better if we have long term data from multiple people. Then we would be able to analyze the differences of motion behaviors and further verify the robustness of the approach.
2. So far, our Voronoi model only captures first-order transitions. This could be the first step toward high-level pattern learning. For example, most navigation patterns depend on the time of day and the (potentially far away) goal of the person and not only on the previous position. We intend to use the model presented here as a tool to generate long sequences of data logs from which we can

determine high-level behavior patterns. On this point, a hierarchical approach may be appropriate. In a hierarchical structure, a motion model has different expressions on different levels. We believe our general framework of combining EM learning with particle filters will work in hierarchical structures, too.

3. We could use relational models to make predictions about novel events. A significant limitation to our current approach is that useful predictions cannot be made when the user is in a location where he has never been before. However, recent work on relational probabilistic models [2; 15] has developed a promising approach where predictions can be made for novel states by smoothing in statistics from semantically similar states. For example, such a model might predict that the user has a significant chance of entering a nearby restaurant at noon even if there is no history of the user patronizing that particular restaurant.
4. For large areas, e.g. the urban area in our experiment, there are at least thousands of edges. Although most of edges will never be touched, they have to be included in the motion model. We can use non-parametric approach with Dirichlet prior [4] such that we only need to store histograms for the edges visited before and are still able to travel to new edges.

7 Acknowledgments

I would like to thank Dieter Fox and Henry Kautz for their instructions and encouragements throughout the project. I'm also grateful to Don Patterson, Kate Deibel, Jeffrey Hightower, Dirk Schulz and Jonathan Ko for their cooperation, discussions and helps during the project.

References

[1] *Probabilistic Robotics*. 2003. draft.

[2] P. Anderson, C. Domingos and D. Weld. Relational markov models and their application to adaptive web navigation. In *Eighth International Conference on Knowledge Discovery and Data Mining*, 2002.

[3] D. Ashbrook and T. Starner. Learning significant locations and predicting user movement with gps. In *IEEE Sixth International Symposium on Wearable Computers*, Seattle, Washington, 2002.

[4] M.J. Beal, Z. Ghahramani, and C.E. Rasmussen. The infinite hidden Markov model. In *Advances in Neural Information Processing Systems (NIPS)*, 2002.

[5] M. Bennewitz, W. Burgard, and S. Thrun. Using EM to learn motion behaviors of persons with mobile robots. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[6] J.A. Bilmes. A gentle tutorial on the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. Technical Report ICSI-TR-97-021, University of Berkeley, 1998.

[7] H. Choset. *Sensor Based Motion Planning: The Hierarchical Generalized Voronoi Graph.* PhD thesis, Caltech, 1996.

[8] A. Doucet, N. de Freitas, and N. Gordon, editors. *Sequential Monte Carlo in Practice.* Springer-Verlag, New York, 2001.

[9] J. Hightower and G. Borriello. Location systems for ubiquitous computing. *Computer*, 34(8), 2001. IEEE Computer Society Press.

[10] H Kautz, L. Arnstein, G. Borriello, O. Etzioni, and D. Fox. An overview of the assisted cognition project. In *AAAI Workshop on Automation as Caregiver: The Role of Intelligent Technology in Elder Care*, 2002.

[11] O. Fox D. Weld D. Kautz, H. Etzioni. Foundations of assisted cognition systems. Technical report, Dept. of Computer Science & Engineering, University of Washington, 2003.

[12] R. Levine and G. Casella. Implementations of the Monte Carlo EM algorithm. *Journal of Computational and Graphical Statistics*, 10, 2001.

[13] N.B. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. In *Proceedings of MOBICOM 2000*, Boston, MA, 2000. ACM Press.

[14] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings of the IEEE.* IEEE, 1989. IEEE Log Number 8825949.

[15] P. Sanghai, S. Domingos and D. Weld. Dynamic probabilistic relational models, 2003.

[16] D. Schulz, W. Burgard, and D. Fox. People tracking with a mobile robot using joint probabilistic data association filters. *International Journal of Robotics Research (IJRR)*, 2003. Accepted for publication.

[17] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10(1), 1992.

[18] T. Want, R. Pering and D. Tennenhouse. Comparing automatic and proactive computing. *IBM Systems Journal*, 42:129–135, 2003.

[19] G. Wei and M.A. Tanner. A Monte Carlo implementation of the EM algorithm and the poor mans data augmentation algorithms. *Journal of the American Statistical Association*, 85, 1990.