CHAPTER 9

Processing Sequential Sensor Data

John Krumm

CONTENTS

9.1 INTRODUCTION

Ubiquitous computing (ubicomp) applications are normally envisioned to be sensitive to context, where context can include a person's location, activity, goals, resources, state of mind, and nearby people and things. Context is often inferred with sensors that periodically measure some aspect of the user's state. For instance, a global positioning system (GPS) sensor can repeatedly measure a person's location at some interval in time. This is an example of sequential sensor data in that it is a sequence of sensor reading of the same entity spread out over time. Unfortunately, sensors are never perfect in terms of noise or accuracy. For example, a GPS sensor gives noisy latitude/longitude measurements, and sometimes the measurements are wildly inaccurate (outliers). In addition, sensors often do not measure the necessary state variables directly. Although a GPS sensor can be used to infer a person's velocity and even mode of transportation (Patterson et al., 2003), it cannot directly measure these states. This chapter is aimed at introducing fundamental techniques for processing sequential sensor data to reduce noise and infer context beyond what the sensor actually measures. The techniques discussed are not necessarily on the cutting edge of signal processing, but they are well-accepted approaches that have proven to be fundamentally useful in ubicomp research. Because of their wide acceptance and usefulness, you should feel comfortable using them in your own ubicomp work. Specifically, this chapter discusses mean and median filters, the Kalman filter, the particle filter, and the hidden Markov model (HMM). Each of these techniques processes sequential sensor data, but they all have different assumptions and representations, which are highlighted to help the reader make an intelligent choice.

This chapter concentrates on processing sequential sensor measurements, because it is usually necessary to make repeated measurements to keep up with possibly changing context in ubicomp applications. For instance, a person's location usually changes with time, so location must be measured repeatedly. Sequential measurements mean that processing techniques can take advantage of both a sense of the past and a sense of the future, which will be explained next.

A sense of the past is useful because context does not change completely randomly, but instead shows some coherence over time. Although an isolated sensor measurement might lead to an uncertain conclusion about a person's context, repeated measurements give more certainty in spite of noisy measurements, partly because context normally does not change

very quickly compared to how often measurements can be taken. Thus, it often makes sense to average together the last few measurements, which is a simple means of exploiting continuity, and one which this chapter examines in Section 9.3. In general, a new measurement triggers a reestimation of the context state variables that is sensitive not only to the new measurement, but also to previous measurements and/or estimates. This historical perspective means the techniques can exploit expectations about the continuity of the state variables. The mean filter fulfills expectations that the output varies relatively slowly and that it should be sensitive to current and past measurements. All the techniques in this chapter pay attention to past measurements or past state estimates in some way.

A sense of the future is usually realized with a dynamic model of the measured process, something that the mean and median filters do not have. For instance, if a person is walking in a certain direction, it is most likely he or she will continue in that direction, because turns are somewhat rare. Like a sense of the past, a sense of the future can help smooth out noise in the measurements. A dynamic model can also keep track of more than just what is measured. For example, the Kalman filter, as explained in Section 9.4, tracks both location and speed based only on location measurements. Paying attention to higher level, albeit unmeasured, state variables means that the processing technique can exploit reasonable assumptions about the behavior of the whole system rather than just the directly measured parts of it. In the Kalman filter, a speed estimate can be used to mitigate the effect of a wildly distant location measurement, because the user might not have been able to move to the distant point given the current speed. As a bonus, these higher level variables become available for use in a context-sensitive system. For instance, even if a system is measuring location only, speed estimates help the processing and can be used as a context output. The activity inference project takes this to the extreme: Lester et al. (2006) were able to distinguish among various activities (e.g., sitting, standing, walking) by measuring acceleration, sound, and barometric pressure. Some of the processing techniques in this chapter explicitly estimate these extra state variables to improve their performance, and the estimates are a useful by-product for inferring context.

A running example in this chapter helps highlight the assumptions, processing, and output of each technique. The next section introduces the example, followed by discussions of each technique in subsequent sections.

9.2 TRACKING EXAMPLE

The problem presented in this chapter focuses on tracking a moving person in the (x,y) plane based on noisy (x,y) measurements taken at an interval of 1 second. The simulated actual path and simulated noisy measurements are shown in Figure 9.1, where the unit of measurement is 1 meter. The path starts at the center of the spiral. Pretend that the person is carrying a location sensor, possibly a GPS, if he is outside. There are 1000 measurements spread evenly in time from beginning to end. The measurements are noisy versions of the actual path points. As is common in modeling noisy measurements, the measurements are taken as the actual values plus added Gaussian noise with zero mean. The Gaussian probability distribution is the familiar bell-shaped curve. "Noise" is not noise in the audible sense, but represents random errors in the measurements. Engineers and researchers use Gaussian noise, as opposed to other probability distributions, partly because it is often an adequate model and partly because it is theoretically convenient. The zero mean assumption implies that the sensor is unbiased (no constant offset error), and this is easily realizable

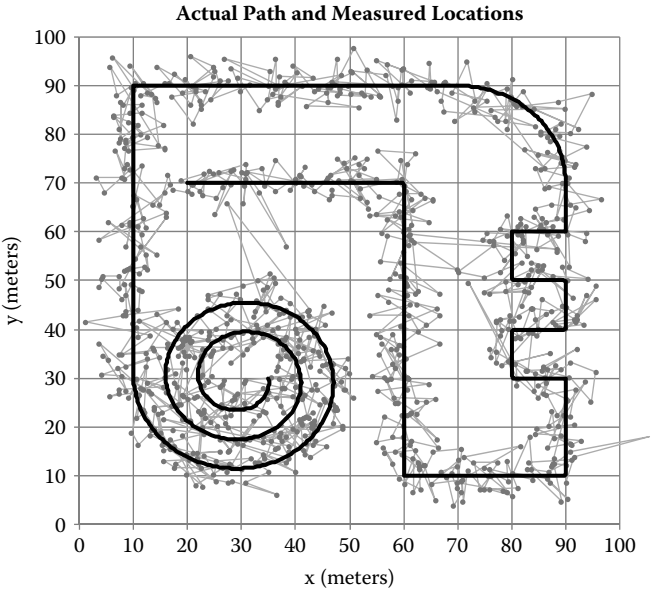

FIGURE 9.1 The actual path, in black, starts at the center of the spiral. The noisy, measured points are gray. One goal of processing the measured points is to estimate the actual path. These data form the basis of the running example in this chapter.

by subtracting any constant offset before processing the measurements. In this tracking example, the standard deviation of the noise is $\sigma = 3$ meters. There are also ten outliers placed randomly along the path, which were taken as the regular noisy measurements plus Gaussian noise with a standard deviation of 15 meters.

This example is helpful because it is easy to visualize and understand. Even though it is clearly in the domain of location measurement, it demonstrates some more general characteristics of the type of inference problems that appear in ubicomp. It consists of sequential measurements that are corrupted by noise, and there are dynamic models of expected behavior that can be used to combat the noise. In this particular example, there are assumptions about the smoothness of the path that are described in subsequent sections. These assumptions can be expressed in terms of other state variables that may be of interest, such as the speed of the person in this case. This example does not illustrate the processing of discrete state variables such as the person's mode of transportation or the number of people traveling together. However, Section 9.6 explains how to convert the problem into discrete variables and how to process them.

In mathematical terms, the actual locations are given by vectors $\boldsymbol{x}_i = (x_i, y_i)^{\mathrm{T}}$, where $\boldsymbol{x}_i$ is a column vector giving the ith location in the sequence, and i goes from 1 to the number of measurements. Note that the boldface $\boldsymbol{x}_i$ is a vector, and the non-boldface x_i is one of its scalar components. The $\boldsymbol{x}_i$ are unknown and the goal of the processing is to estimate them from the noisy measurements and to possibly infer other state variables such as velocity. The noisy measurements are represented by $\boldsymbol{z}_i$, which are the same as the measurement vectors, but with independent, zero-mean Gaussian noise added to each component:

$$\boldsymbol{z}_i = \boldsymbol{x}_i + \boldsymbol{v}_i \qquad \boldsymbol{v}_i \sim N\left(0, \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix}\right) \tag{9.1}$$

Here, $\boldsymbol{v}_i$ is a random Gaussian noise vector with zero mean and a diagonal covariance matrix. Since the covariance matrix is diagonal, this is the same as adding independent Gaussian noise to each element of $\boldsymbol{x}_i$. Note that Equation (9.1) is not part of any algorithm for inferring $\boldsymbol{x}_i$, but just an assumption on the relationship between the measurements $\boldsymbol{z}_i$ and actual $\boldsymbol{x}_i$ values. It also describes the simulated measurement data used in this chapter's running example.

This example is intentionally simple so it is easy to visualize and thus some of the simpler techniques will work on it. This chapter will show how

each technique performs on this example, and simple techniques, such as the mean and median filter (discussed next), appear to work better than some of the more sophisticated methods. However, each of the techniques presented has tunable parameters that affect their performance, and the presented results do not necessarily represent the optimal state of tune. This means that each of the techniques has the potential to give more accuracy. Moreover, some of the more sophisticated techniques have the ability to give additional information, such as speed estimates, that simpler techniques do not give. Thus, judging the techniques merely by their accuracies presented in this chapter would be naïve.

One danger of a specific, running example is that it appears to limit the applicability of the methods to just that example. However, these techniques have been used for a wide range of sequential sensor data and a wide range of inferred state variables.

9.3 MEAN AND MEDIAN FILTERS

One of the simplest and most effective means of filtering out noise is to average together multiple samples. For the example problem, the mean filter estimates $\boldsymbol{x}_i$ by the average of $\boldsymbol{z}_i$ and the most recent $n - 1$ measurements, that is,

$$\hat{\boldsymbol{x}}_i = \frac{1}{n} \sum_{j=i-n+1}^{i} \boldsymbol{z}_j \tag{9.2}$$

Here $\hat{\boldsymbol{x}}_i$ represents the estimated value of $\boldsymbol{x}_i$. Equation (9.2) represents a sliding window of width n, where n is the number of values to use to compute the mean. This filter is simple to implement and works well to reduce noise, as shown in Figure 9.2, where $n = 10$. The primary disadvantage is that it introduces lag in the estimate, because the average is taken over mostly measurements that come before $\boldsymbol{z}_i$. One way to reduce the lag is to use a weighted average whose weights decrease in value for increasingly older measurements. Note that the mean filter in Equation (9.2) is a "causal filter," because it does not look ahead in time to future measurements to estimate $\boldsymbol{x}_i$. This is true of all techniques discussed in this chapter.

In addition to lag, another potential problem with the mean filter is its sensitivity to outliers. In fact, just a single measured point, placed far enough away, can move the mean to any location. The median is a more robust version of the mean, and it still works if up to half the data is outliers.

$$\hat{\boldsymbol{x}}_i = \text{median}\{\boldsymbol{z}_{i-n+1}, \boldsymbol{z}_{i-n+2}, \ldots, \boldsymbol{z}_{i-1}, \boldsymbol{z}_i\} \tag{9.3}$$

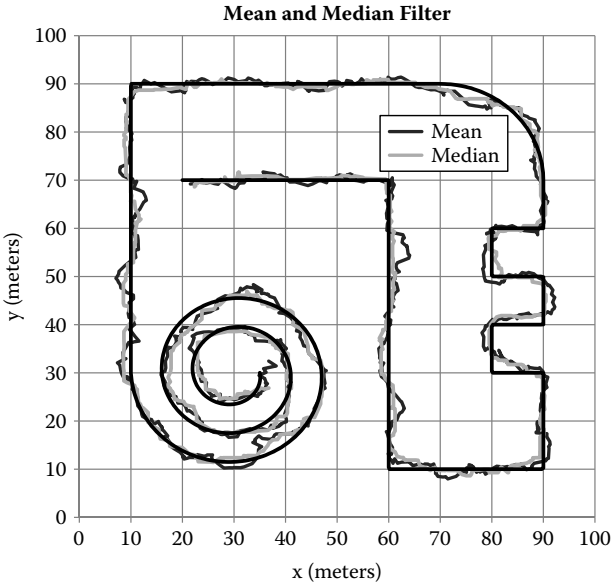

FIGURE 9.2 The actual path is in black. The paths estimated by the mean and median filters are in dark gray and light gray, respectively. The median filter produces fewer large excursions from the true path, because it is more robust to outliers.

Figure 9.2 shows the estimated path based on the median. There are places where the mean drifts relatively far from the actual path due to an outlier, whereas the median stays closer.

The mean and median filters are simple, yet effective techniques for processing sequential sensor data. However, they do suffer from lag, and they do not intrinsically estimate any higher level variables such as speed. One could estimate speed with a numerical derivative, but this is very sensitive to the noise in the original measurements. The remaining techniques discussed in the chapter work to reduce lag with a dynamic model, and the Kalman and particle filters (Sections 9.4 and 9.5) can estimate higher level state variables in a principled way.

9.4 KALMAN FILTER

The Kalman filter is a big step up in sophistication from the mean and median filters discussed above. It explicitly accounts for sensor noise (as long as the noise is additive Gaussian), and it explicitly models the system's dynamics. The Kalman filter introduces probability to the problem

of processing sequential measurements. Instead of producing just an estimated result $\hat{\boldsymbol{x}}_i$ such as the mean and median filters, it also produces an uncertainty estimate that indicates the confidence of the estimate.

The Kalman filter has a number of descriptive adjectives to distinguish it from other filters for processing sequential data (some of the less obvious terminology below is explained subsequently):

- Bayesian—It uses Bayes rule to estimate the probability distribution of the state variables from noisy measurements $\boldsymbol{z}_i$.
- Gaussian—All probability distributions in the Kalman filter are Gaussians, including measurement noise, process noise, and the state estimates.
- Linear—The dynamics of the system being measured are linear.
- Online—The Kalman filter can update the state estimate as soon as the latest measurement is available. It does not have to wait until all the data are available, like a batch filter would.

One of the most enduring references on Kalman filtering is the book by Gelb (1974). The mathematical notation in this section is the same as in Gelb's book.

9.4.1 Linear, Noisy Measurements

The mean and median filters can only estimate states that are directly measured. For instance, if the measurements are (x,y) coordinates as in this chapter's example, the mean and median filters cannot be used to directly, in a principled way, estimate velocity. The Kalman filter makes a distinction between what quantities are in the measurement vector $\boldsymbol{z}_i$ and the unknown state vector $\boldsymbol{x}_i$. In the example for the Kalman filter, these two vectors are

$$\boldsymbol{z}_i = \begin{pmatrix} z_i^{(x)} \\ z_i^{(y)} \end{pmatrix} \qquad \boldsymbol{x}_i = \begin{pmatrix} x_i \\ y_i \\ s_i^{(x)} \\ s_i^{(y)} \end{pmatrix} \tag{9.4}$$

where $z_i^{(x)}$ and $z_i^{(y)}$ are the ith measurements of the x and y coordinates, x_i and y_i are the unknown actual coordinates, and $s_i^{(x)}$ and $s_i^{(y)}$ represent

the unknown velocity vector's x and y components. The state vector contains variables that are not directly measured—velocity, in this case.

To express the full relationship between the actual state and measurements, the Kalman filter adds a measurement matrix H and zero-mean Gaussian noise:

$$\boldsymbol{z}_i = H_i \boldsymbol{x}_i + \boldsymbol{v}_i \qquad \boldsymbol{v}_i \sim N(0, R_i) \tag{9.5}$$

Here, the measurement matrix H_i translates between the state vector and the measurement vector. Because the measurements are related to the actual state by a matrix, the measurements are said to be linear. In the example, H_i simply deletes the unmeasured velocity and passes through the (x,y) coordinates:

$$H_i = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{9.6}$$

The noise vector $\boldsymbol{v}_i$ has the same dimensions as the measurement vector $\boldsymbol{z}_i$, and is distributed as zero-mean Gaussian noise with a covariance vector R_i. In the example, the noise covariance is independent of i. Because this example is a simulation, the Kalman filter has the advantage of knowing the exact noise covariance, from Equation (9.1):

$$R_i = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix} \tag{9.7}$$

9.4.2 Linear, Noisy Dynamics

The mean and median filters have no model of how the state variables change over time. The Kalman filter does have such a model. It says that the state at time i is a linear function of the state at time $i - 1$ plus zero-mean, Gaussian noise:

$$\boldsymbol{x}_i = \phi_{i-1} \boldsymbol{x}_{i-1} + \boldsymbol{w}_{i-1} \qquad \boldsymbol{w}_i \sim N(0, Q_i) \tag{9.8}$$

The system matrix φ_{i-1} gives the linear relationship between the state at time $i - 1$ and i. For the example, the system matrix is

$$\phi_{i-1} = \begin{bmatrix} 1 & 0 & \Delta t_i & 0 \\ 0 & 1 & 0 & \Delta t_i \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9.9}$$

Here, Δt_i is the time elapsed between measurements $i - 1$ and i. With this matrix and the state vector of the example, the equation $\boldsymbol{x}_i = \boldsymbol{\varphi}_{i-1}\boldsymbol{x}_{i-1}$ says that $x_i = x_{i-1} + \Delta t_i s_i^{(x)}$ and similarly for y_i. This is just standard physics for a particle moving in a straight line at constant velocity. The system equation also says that the velocity stays constant over time. Of course, this is not true for the example, and is usually not true for any system that controls its own trajectory. In general, the system noise $\boldsymbol{w}_i$ helps account for the fact that $\boldsymbol{\varphi}_{i-1}$ is not an exact model of the system. $\boldsymbol{w}_i$ is zero-mean, Gaussian noise with covariance Q_i. For the example, Q_i is

$$Q_i = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_s^2 & 0 \\ 0 & 0 & 0 & \sigma_s^2 \end{bmatrix} \tag{9.10}$$

This implies that the physical model connecting position and velocity is correct (which it is). But it also implies that velocity is subject to some noise between updates, which helps compensate for the fact that otherwise velocity is assumed to be constant. Setting the actual values of Q_i is not straightforward. For the example, σ_s^2 was chosen to represent the variance in the actual velocity from measurement to measurement. Q_i is modeling the fact that velocity is not constant in the example. If it were, the trajectory would have to be straight. With Q_i given as Equation (9.10), Equation (9.8) states that the velocity changes randomly between time steps, with the changes distributed as a zero mean Gaussian.

9.4.3 All Parameters

Implementing the Kalman filter requires the creation of the measurement model, system model, and initial conditions. Specifically, it requires

H_i = measurement matrix giving measurement $\boldsymbol{z}_i$ from state $\boldsymbol{x}_i$ (Equation 9.5)

R_i = measurement noise covariance matrix (Equation 9.5)

$\boldsymbol{\varphi}_{i-1}$ = system matrix giving state $\boldsymbol{x}_i$ from $\boldsymbol{x}_{i-1}$ (Equation 9.8)

Q_i = system noise covariance matrix (Equation 9.8)

$\hat{\boldsymbol{x}}_0$ = initial state estimate

P_0 = initial estimate of state error covariance

The initial state estimate can usually be estimated from the initial few measurements. For this chapter's example, the initial position came from $\boldsymbol{z}_0$, and the initial velocity was taken as zero. A reasonable estimate of P_0 for this example is

$$P_0 = \begin{bmatrix} \sigma^2 & 0 & 0 & 0 \\ 0 & \sigma^2 & 0 & 0 \\ 0 & 0 & \sigma_s^2 & 0 \\ 0 & 0 & 0 & \sigma_s^2 \end{bmatrix} \tag{9.11}$$

9.4.4 Kalman Filter

The Kalman filter proceeds in two steps for each new measurement $\boldsymbol{z}_i$. The result of the two steps is the mean and covariance of the estimated state, $\hat{\boldsymbol{x}}_i^{(+)}$. The first step is to extrapolate the state and the state error covariance from the previous estimates. These are pure extrapolations with no regard for the measurement, and they depend only on the system model. The (–) superscript indicates an extrapolated value.

$$\hat{\boldsymbol{x}}_i^{(-)} = \phi_{i-1}\hat{\boldsymbol{x}}_{i-1}^{(+)} \tag{9.12}$$

$$P_i^{(-)} = \phi_{i-1}P_{i-1}^{(+)}\phi_{i-1}^{\mathrm{T}} + Q_{i-1} \tag{9.13}$$

The second step is to update the extrapolations with the new measurement, giving the mean and covariance of the new state estimate, $\hat{\boldsymbol{x}}_i^{(+)}$ and $P_i^{(+)}$:

$$K_i = P_i^{(-)}H_i^{\mathrm{T}}\left(H_iP_i^{(-)}H_i^{\mathrm{T}} + R_i\right)^{-1} \tag{9.14}$$

$$\hat{\boldsymbol{x}}_i^{(+)} = \hat{\boldsymbol{x}}_i^{(-)} + K_i\left(\boldsymbol{z}_i - H_i\hat{\boldsymbol{x}}_i^{(-)}\right) \tag{9.15}$$

$$P_i^{(+)} = (I - K_iH_i)P_i^{(-)} \tag{9.16}$$

where I is the identity matrix and K_i is the Kalman gain matrix.

Applying these equations to this chapter's example gives the result shown in Figure 9.3, where the plotted value is $\hat{\boldsymbol{x}}_i^{(+)}$. The estimated path in dark gray does not follow turns very well, partly because the system model assumes the path is a straight line, with only the noise process to account for turns.

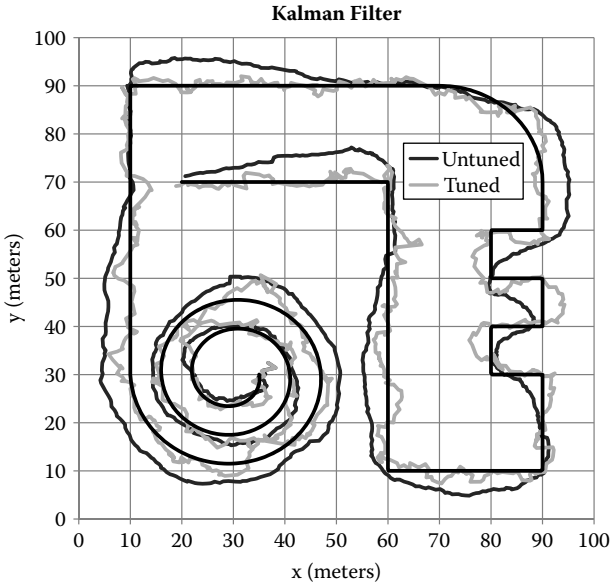

FIGURE 9.3 The Kalman filter result, in dark gray, tends to overshoot turns because its dynamic model assumes a single, straight line path. The lighter gray line is a version of the Kalman filter tuned to be more sensitive to the data. It follows turns better, but is also more sensitive to noise.

9.4.5 Discussion

The major difference between the Kalman filter and the mean and median filters is that the Kalman filter has a dynamic model of the system to keep up with changes over time. Besides fixing the lag problem, the dynamic model counterbalances the measurement model to give a tunable tradeoff between a designer's belief in the predictive dynamics versus sensor readings. This tuning is reflected in the choice of the elements of the measurement and system covariance matrices. Although the measurement covariance is relatively simple to estimate based on sensor characteristics, the system covariance is more difficult, and it represents an opportunity for tuning the tradeoff between dynamics and measurement. Figure 9.3 shows two results of the Kalman filter: the less accurate one in dark gray reflects a principled choice of σ_s, which specifies the noise in the predictive velocity model. The inferred path tends to have trouble tracking curves and corners, because the model is biased too much toward straight line paths. The more accurate inferred path in light gray comes from an optimal choice of σ_s based on an exhaustive

sequence of test values of σ_s. The price of this more responsive model is a wigglier path, because this filter is more sensitive to the measurements. Of course, using a textbook example makes such a search easy. The problem of tuning is more difficult for real applications.

The dynamic model can include parameters beyond the ones that are measured. In the example, the state vector for the Kalman filter added velocity as a state parameter, even though the measurements had location only. An obvious extension to the example would be to include acceleration. The ability to track nonmeasured parameters is a benefit, because the nonmeasured parameters may be useful for context inference.

The Kalman filter can also make sensor fusion straightforward. For the example in this section, an accelerometer could add valuable data about location and velocity. This would involve augmenting the state vector, system model, and measurement model with acceleration.

Another advantage of the Kalman filter is that it gives an estimate of its own uncertainty in the form of the covariance matrix $P_i^{(+)}$. A knowledge of uncertainty is useful for ubicomp systems. For instance, a Kalman filter tracker might indicate that a person is equally likely in the kitchen or the living room, which could affect whether certain automatic actions are triggered.

One of the main limitations of the Kalman filter is the linearity of the dynamic model. Some processes are inherently linear, such as the radar traces of ballistic missiles to which Kalman filters were applied a long time ago. System dynamics are often not linear. For instance, a ball bouncing off a wall could not be represented by the state system matrix φ_{i-1} in the Kalman filter. Likewise, the combinations of distances and angles in pose estimation are not linear. The extended Kalman filter can sometimes solve nonlinear problems by linearizing the system around the estimated state. Until ubicomp advances to the ballistic missile stage, applications of the basic Kalman filter in the field will be relatively rare.

The Kalman filter is also unsuitable for representing discrete state variables such as a person's mode of transportation, knowledge of the world, or goals. Fortunately, for nonlinear problems with a mix of continuous and discrete state variables, the particle filter is a practical, although computationally more expensive, solution. This is the topic of the next section.

9.5 PARTICLE FILTER

The particle filter is a more general version of the Kalman filter, with less restrictive assumptions and, because of that, more computational demand. Unlike the Kalman filter, the particle filter does not require a linear model

for the process in question, and it does not assume Gaussian noise. One of the main advantages of the particle filter for ubicomp applications is that it can easily represent an arbitrary mix of continuous and discrete variables along with a rich model of how these variables interact and affect sensors. In terms of the list of properties given for the Kalman filter, the particle filter is Bayesian, non-Gaussian, nonlinear, and online.

One good example of a particle filter for a ubicomp application is the work of Patterson et al. (2003). They created a rich model of a person's location, velocity, transportation mode, GPS error, and the presence of a parking lot or bus stop. The only sensor used was GPS for location and velocity. Using a particle filter, they could infer all the other variables. Note that their state space contained both continuous variables (location, velocity, and GPS error) and discrete variables (transportation mode, presence of parking lot or bus stop).

The particle filter has its name because it represents a multitude of possible state vectors, which can be thought of as particles. Each particle can be considered a hypothesis about the true state, and its plausibility is a function of the current measurement. After each measurement, the most believable particles survive, and they are subject to random change in state according to a probabilistic dynamic model. An easy-to-understand introduction to particle filtering is the chapter by Doucet et al. (2001), and this chapter uses their notation. Hightower and Borriello (2005) give a useful case study of particle filters for location tracking in ubicomp.

As in the Kalman filter section (Section 9.4.4), the subsections below explain how to implement a particle filter using this chapter's tracking example.

9.5.1 Problem Formulation

The particle filter is based on a sequence of unknown state vectors, $\boldsymbol{x}_i$, and measurement vectors, $\boldsymbol{z}_i$, which is the same as the Kalman filter, both in general and in this chapter's example. As a reminder, for the example, the state vector represents location and velocity, and the measurement vector represents a noisy version of location. The particle filter parallels the Kalman filter with a probabilistic model for measurements and dynamics, although both are more general than the Kalman filter.

The probability distribution $p(\boldsymbol{z}_i|\boldsymbol{x}_i)$, which you must provide, models the noisy measurements. This can be any probability distribution, which is more general than the Kalman filter formulation, which was $\boldsymbol{z}_i = H_i\boldsymbol{x}_i + \boldsymbol{v}_i$. The measurement probability distribution gives a probabilistic relationship between the actual state $\boldsymbol{x}_i$ given the measurement $\boldsymbol{z}_i$. This models the

realistic scenario where the measurement can be some complicated function of the state, with some uncertainty. To stay consistent, the particle filter example will use the same measurement model as the Kalman filter, which states

$$p(\boldsymbol{z}_i|\boldsymbol{x}_i) = N\left((x_i, y_i)^{\mathrm{T}}, R_i\right) \tag{9.17}$$

This says that the measurement is normally distributed around the actual location (x_i,y_i) with the same covariance matrix used for the Kalman measurement model in Equation (9.7). However, $p(\boldsymbol{z}_i|\boldsymbol{x}_i)$ could be much more interesting and expressive. For instance, with knowledge of a map, $p(\boldsymbol{z}_i|\boldsymbol{x}_i)$ could express the fact that measurements in some regions are less accurate than measurements in other regions (e.g., the problem of GPS losing satellite signals in buildings, tunnels, and urban canyons).

Another probability distribution you must provide is $p(\boldsymbol{x}_i|\boldsymbol{x}_{i-1})$, which models the dynamics. It gives the distribution of the current state $\boldsymbol{x}_i$ given the previous state $\boldsymbol{x}_{i-1}$. This is also more general than the Kalman filter, whose dynamics model is $\boldsymbol{x}_i = \varphi_{i-1}\boldsymbol{x}_{i-1} + w_{i-1}$. It is not necessary to write down $p(\boldsymbol{x}_i|\boldsymbol{x}_{i-1})$, but the particle filter requires sampling from it. That is, given an $\boldsymbol{x}_{i-1}$, the particle filter algorithm requires the generation of a random $\boldsymbol{x}_i$, subject to $p(\boldsymbol{x}_i|\boldsymbol{x}_{i-1})$. For this chapter's example, the assumption is that the location changes deterministically as a function of the velocity, and that the velocity is randomly perturbed with Gaussian noise:

$$\begin{aligned}
x_{i+1} &= x_i + s_i^{(x)}\Delta t_i \\
y_{i+1} &= y_i + s_i^{(y)}\Delta t_i \\
s_{i+1}^{(x)} &= s_i^{(x)} + w_i^{(x)} \qquad w_i^{(x)} \sim N\left(0,\sigma_s^2\right) \\
s_{i+1}^{(y)} &= s_i^{(y)} + w_i^{(y)} \qquad w_i^{(y)} \sim N\left(0,\sigma_s^2\right)
\end{aligned} \tag{9.18}$$

This is the same as the assumed dynamic model for the Kalman filter in Equation (9.8), but expanded for each component of the state vector to emphasize that the update does not need to be linear. It also does not need to be Gaussian. For consistency with the example, however, this dynamic model is both linear and Gaussian. The generality of $p(\boldsymbol{x}_i|\boldsymbol{x}_{i-1})$ would allow for more domain knowledge in the model. For instance, with knowledge of a map, $p(\boldsymbol{x}_i|\boldsymbol{x}_{i-1})$ could express the fact that it is more likely to accelerate going down hills and decelerate going up hills.

The other component of the particle filter formulation is a prior distribution $p(\boldsymbol{x}_0)$ on the state vector before any dynamics can be applied. For the example, let the prior be a normal distribution around the first measurement:

$$p(\boldsymbol{x}_0) = N(\boldsymbol{z}_0, R_i) \tag{9.19}$$

Although the variables in the example are all continuous, note that these distributions could involve a mix of continuous and discrete variables, and they could interact in interesting ways. For instance, consider appending a discrete home activity variable to the state vector $\boldsymbol{x}_i$, where the activities can be {sleeping, eating, studying}. Assuming that the subject engages in only one of these activities at a time, the measurement model of the particle filter $p(\boldsymbol{z}_i|\boldsymbol{x}_i)$ could express the fact that certain activities are more likely to occur in or near certain rooms of the house.

9.5.2 Particle Filter

The particle filter works with a population of N particles representing the state vector: $\boldsymbol{x}_i^{(j)}$, $j = 1, \ldots, N$. Unlike the other methods in this chapter, the particle filter actually involves generating random numbers, which are necessary to instantiate the values of the particles. This sampling means writing a routine that generates sample random numbers adhering to the relevant probability distributions.

Although there are several variations of the particle filter, one of the most popular is called the bootstrap filter, introduced by Gordon (1994). It starts by initializing N samples of $\boldsymbol{x}_0^{(j)}$ generated from the prior distribution $p(\boldsymbol{x}_0)$. In the example, a plot of these particles would cluster around the first measurement $\boldsymbol{z}_0$, because the prior term in Equation (9.19) is a normal distribution with mean $\boldsymbol{z}_0$.

With the initialization complete, for $i > 1$, the first step is "importance sampling," which uses the dynamic model to generate random $\tilde{\boldsymbol{x}}_i^{(j)}$ from $p(\boldsymbol{x}_i|\boldsymbol{x}_{i-1})$. This propagates the particles forward according to the assumed dynamic model, but without any guidance from the new measurement $\boldsymbol{z}_i$.

The next step computes "importance weights" for each particle according to the measurement model:

$$\tilde{w}_i^{(j)} = p\left(\boldsymbol{z}_i \middle| \tilde{\boldsymbol{x}}_i^{(j)}\right) \tag{9.20}$$

A larger importance weight indicates a particle that is better supported by the measurement. The importance weights are then normalized so they sum to one.

Finally, the last step in the loop is the "selection step," which samples a new set of particles $\boldsymbol{x}_i^{(j)}$ from the $\tilde{\boldsymbol{x}}_i^{(j)}$ based on the normalized importance weights. This involves picking N of $\tilde{\boldsymbol{x}}_i^{(j)}$ at random, where the probability of picking $\tilde{\boldsymbol{x}}_i^{(j)}$ is proportional to its weight $\tilde{w}_i^{(j)}$. This step tends to eliminate unlikely particles, and it is not unusual to pick the same $\tilde{\boldsymbol{x}}_i^{(j)}$ more than once if its weight is relatively large. The selection step is the last step in the loop. The algorithm returns to the importance sampling step for the next value of i to process the next measurement.

To compute an estimate of $\boldsymbol{x}_i$ at any point, which was the original goal, compute the weighted mean of the particles:

$$\hat{\boldsymbol{x}}_i = \sum_{j=1}^{N} \tilde{w}_i^{(j)} \tilde{\boldsymbol{x}}_i^{(j)} \tag{9.21}$$

This works for continuous variables. For discrete variables, a viable approach is weighted voting. Similar equations work for computing other expected values, such as variance. This is useful for estimating the uncertainty of the estimate.

Applied to this chapter's example problem, the particle filter gives the result shown in Figure 9.4.

9.5.3 Discussion

The main problem with the particle filter is computation time. In general, using more particles (larger N) helps the algorithm work better, because it can more completely represent and explore the space of possible state vectors. But often, adding enough particles to make a noticeable improvement in the results also causes a significant increase in computation time. Fox (2003) gives a method for choosing N based on bounding the approximation error.

Because the particle filter allows such a rich state representation, it is tempting to add state variables. In this chapter's example, it would be interesting to add state variables governing the mode of transportation (e.g., walking vs. driving), intended route, and intended destination. However, increasing the dimensionality of the state vector usually

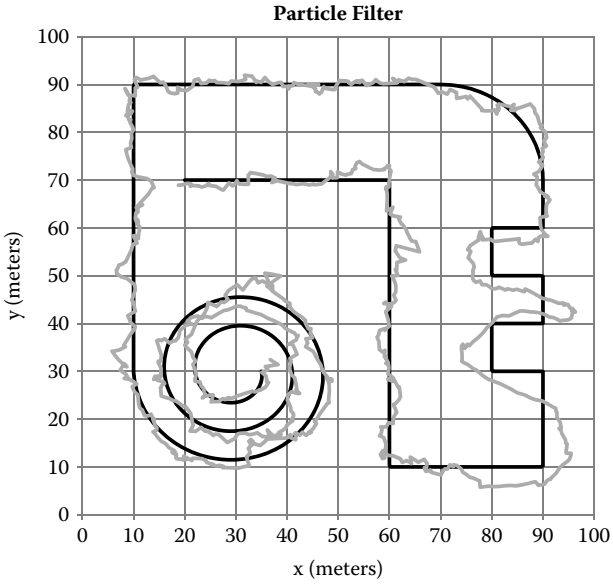

FIGURE 9.4 The particle filter result.

requires adding more particles to account for the larger state space, leading to increased computation time. One interesting remedy for this is the Rao-Blackwellized particle filter (Murphy and Russell, 2001). It uses a more conventional filter, such as Kalman, for the state variables where it is appropriate, and it uses a particle filter for the other state variables. For instance, Kalman could cover location and velocity, and a particle filter could track the higher level states.

9.6 HIDDEN MARKOV MODEL

The HMM works only for discrete valued state space variables. This is in contrast to the Kalman filter, which works only for continuous valued variables, and the particle filter, which works for a mix of discrete and continuous variables. The HMM has been primarily the tool of speech understanding researchers, where the problem is to find the set of words that best accounts for a speech signal. In fact, the classic tutorial paper on HMMs by Rabiner (1989) was aimed at speech understanding, although the paper is useful to learn about the HMM for any application.

As in the previous sections, this section uses the chapter's tracking example to illustrate how to use an HMM.

9.6.1 Problem Formulation

As with the previous filtering methods, the measurements are represented by $\boldsymbol{z}_i$, where the subscript represents the time the measurement was taken. Instead of a continuous state $\boldsymbol{x}_i$, the HMM works with discrete states $X_i^{(j)}$. Here, the subscript again refers to time. The superscript indexes through the M possible states. In an activity recognition system, $X_i^{(j)}$ could represent different modes of transportation at time i, that is, $X_i^{(j)} \in$ {bus, foot, car}, as in Patterson et al. (2003). Here, $j = 1$ means "bus," $j = 2$ means "foot," and $j = 3$ means "car," and there are $M = 3$ possible states.

In this chapter's example, the natural way to represent the state is with a continuous coordinate $(x_i,y_i)^{\mathrm{T}}$. For the HMM, however, with its requirement for discrete state, the example splits the coordinate plane into small cells, as shown in Figure 9.8. Each cell in the example is a square whose sides are 1 meter long. Since the space extends to 100 meters along both axes, there are $M = 10{,}000$ cells. The goal of HMM is to estimate which of these cells contains the tracked object after each continuous measurement $\boldsymbol{z}_i$.

Both the Kalman filter and particle filter have a measurement model. In the Kalman filter, the measurement model, given by Equation (9.5), says that the measurement is a linear function of the state plus additive Gaussian noise. In the particle filter, the measurement model is a general, conditional probability $p(\boldsymbol{z}_i|\boldsymbol{x}_i)$. The measurement model for the HMM is similar to the one for the particle filter, except that it gives the discrete probability of each state, given the measurement: $P(X_i^{(j)}|\boldsymbol{z}_i)$. P (uppercase) indicates a discrete probability distribution. In the HMM terminology, the measurement model is used to compute the "observation probabilities." Given a measurement $\boldsymbol{z}_i$, there is one observation probability value for each possible state $X_i^{(j)}$. These observation probabilities must sum to 1; that is,

$$\sum_{j=1}^{M} P\left(X_i^{(j)}|\boldsymbol{z}_i\right) = 1 \tag{9.22}$$

In the example, the measurements have Gaussian noise, as in Equation (9.1). This means that the observation probabilities at time i are concentrated in the cells around the one containing $\boldsymbol{z}_i$. In fact, the amount of probability in each cell is properly computed by integrating the 2-D Gaussian over the boundaries of each cell. An adequate approximation,

used in the example here, is to compute the value of the Gaussian at the center of each cell and then normalize the probabilities so they sum to 1. Note that since the Gaussian goes on forever (infinite support), all cells have some nonzero probability.

Both the Kalman filter and particle filter have a dynamic model. For the Kalman filter, the dynamic model says that the new state is a linear function of the old state, plus additive Gaussian noise, as given by Equation (9.8). The particle filter is more general, specifying the dynamic model as a conditional probability distribution $p(\boldsymbol{x}_i|\boldsymbol{x}_{i-1})$. The dynamic model for the HMM is expressed in terms of transition probabilities, $P(X_{i+1}^{(k)}|X_i^{(j)}) = a_{jk}$. This gives the probability of transitioning from state j to state k between successive measurements. In the {bus, foot, car} example, the transition probabilities could reflect the fact that it is improbable to transition directly from a bus to a car, because there is usually some foot travel in between. The transition probabilities going from one state to all the others must sum to 1:

$$\sum_{k=1}^{M} a_{jk} = 1 \tag{9.23}$$

In another example, the Locadio WiFi location system, Krumm and Horvitz (2004) attempted to infer whether a person was moving based on WiFi signal strengths measured from the laptop they were carrying. High variance in the measured signal strengths was found to indicate movement. However, the raw inferences from signal strength variance indicated that people would transition between moving and still much more often than in reality, as shown in Figure 9.9. An HMM was used to reduce the number of transitions. Here, the transition probabilities are given by the diagram in Figure 9.5, where the self-transition probabilities are quite large, which help prevent spurious transitions between the two states.

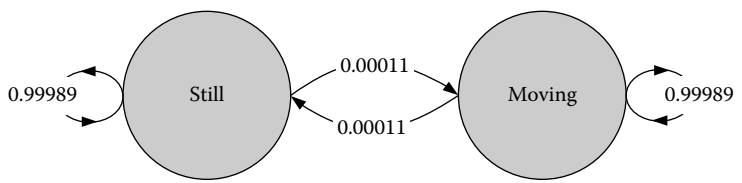

FIGURE 9.5 These transition probabilities were used in an HMM to smooth the transitions in Figure 9.9.

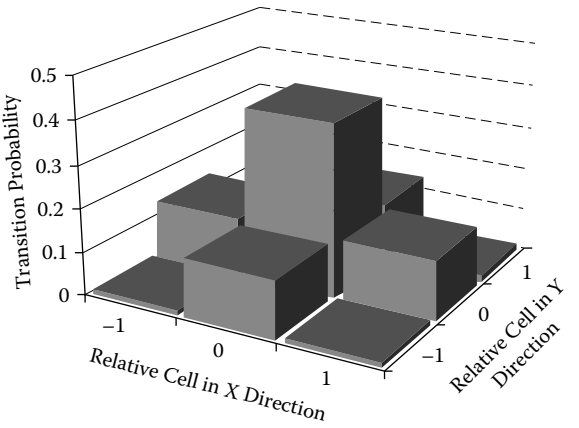

FIGURE 9.6 The HMM transition probabilities for each cell look like this, with the most probable transition being back to the same cell. The only other nonzero transition probabilities are to the immediate neighbors.

For this chapter's tracking example, there are $M = 10{,}000$ states, one for each cell. The transition probabilities reflect the fact that it is more likely to transition to a nearby cell than a distant cell. In fact, an examination of the actual path shows that the probability of staying in the same cell from time i to $i+1$ is about 0.40. The only other nonzero transition probabilities are into the cells immediately surrounding the current cell, as shown in Figure 9.6. The example uses the values in Figure 9.6 to fill in the transition probabilities a_{jk}.

The last element of the HMM is a set of initial state probabilities, $P(X_0^{(j)})$. With no knowledge of the initial state, it is reasonable to specify a uniform distribution over all states, that is, $P(X_0^{(j)}) = 1/M$. For the tracking example, the initial state probabilities were spread in a Gaussian pattern around the first measurement.

9.6.2 Hidden Markov Model

An example HMM is summarized in Figure 9.7. The time index starts at zero on the left and increases to the right, with a total of $N = 3$ measurements. The rows represent $M = 8$ possible states. At the beginning, there are M initial state probabilities. The subsequent columns represent observation probabilities that are sensitive to the measurements $\boldsymbol{z}_i$. Connecting the states in each column are transition probabilities. The goal of the HMM is to find an optimal path starting with the initial state probabilities,

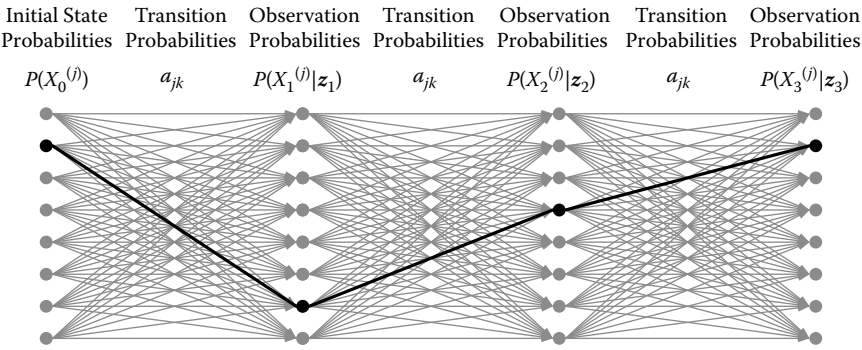

FIGURE 9.7 HMM considers all possible states at each time step. This HMM has $M = 8$ states and $N = 3$ measurements. The Viterbi algorithm is an efficient way to find the most probable path through the observation probabilities and transition probabilities. The dark lines represent one possible path.

through the transition and observation probabilities. The mostly likely path is the one that maximizes the product of these probabilities.

One way to find the optimal path is an exhaustive search through the M^N possible paths. Although this is could be feasible for the HMM in Figure 9.7, the tracking example has $M = 10{,}000$ and $N = 1000$ for a total of 1×10^{4000} different possible paths. Instead, finding the optimal path traditionally makes use of the Viterbi algorithm, explained for the HMM by Rabiner (1989). The Viterbi algorithm uses dynamic programming to efficiently find the most probable path.

Results of the HMM on this chapter's example are shown in Figure 9.8.

9.6.3 Discussion

The HMM is sometimes a very helpful add-on to a system that infers discrete states as a means of reducing the frequency of transitions between states. For example, one part of the Locadio system attempted to infer if a user was walking or still, based on the variance of measured WiFi signal strengths. The raw measurements of variance indicated very frequent transitions between the two states. The system used a simple two-state HMM with transition probabilities that approximated the realistic probabilities of making such a transition, leading to a much smoother output, as shown in Figure 9.9.

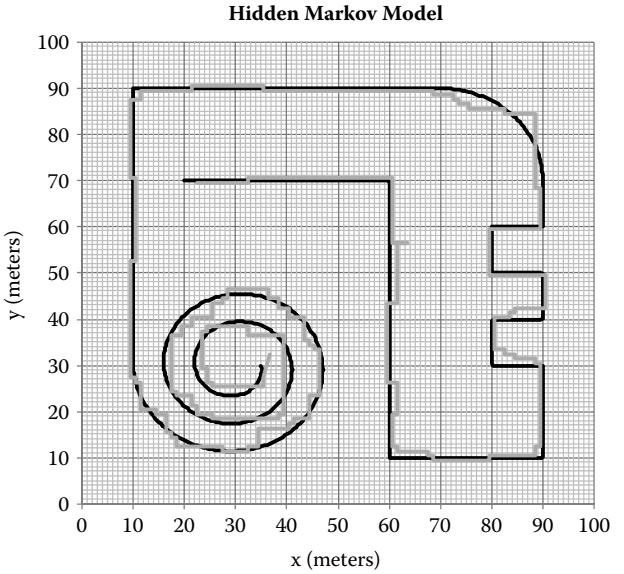

FIGURE 9.8 The HMM works with discrete state variables. For this example, each discrete state is one of 10,000 1 × 1 meter cells. The gray line shows the result of the HMM.

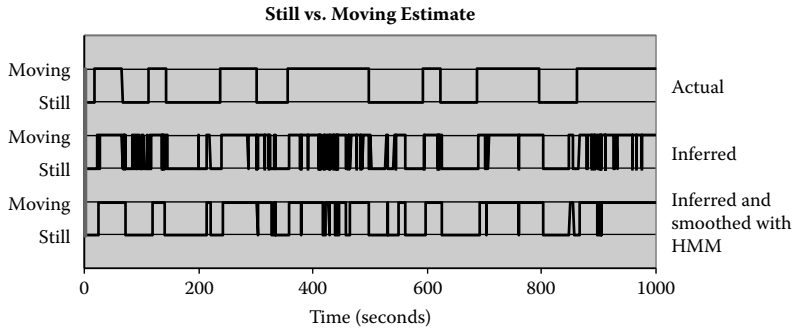

FIGURE 9.9 An HMM can be a useful add-on to a process that infers the discrete state of an object. Here, an HMM was used to smooth the inferred transitions between the states of "still" and "moving" inferred from WiFi signal strengths for a person carrying a laptop inside a building.

One limitation of the HMM is state duration. Although the self-transition probability a_{jj} is adjustable, its value leads to a certain probability density governing how long the subject will stay in state j, as explained by Rabiner (1989). There are techniques to get around this problem, also explained by Rabiner (1989), but they are not as elegant as the original HMM. This was not a problem in the tracking example, because the subject moved at a constant speed along the path (never pausing).

9.7 PRESENTING PERFORMANCE

There are a few standard ways to present the performance of algorithms for processing sequential sensor data. Having de facto standards is important, because it allows a fair comparison of different research results. This section discusses performance measures for both continuous and discrete state measurements.

9.7.1 Presenting Continuous Performance Results

Continuous state variables usually exist in a so-called metric space where there is a defined distance (metric) between the values. For instance, it is easy to define the distance between pairs of temperatures, pressures, light levels, speeds, etc. In the tracking example, the distance is simply Euclidean distance. In general, the error between the actual state vector $\boldsymbol{x}_i$ and the estimated state vector $\hat{\boldsymbol{x}}_i$ is the scalar distance between the vectors: $e_i = \|\hat{\boldsymbol{x}}_i - \boldsymbol{x}_i\|$. For these scalar distances, it is easy to compute the mean and median error over all the estimates, both of which are legitimate means of assessing performance. For one-dimensional state variables such as speed, note that e_i is the absolute value of the difference between the estimated and the actual value. This is desirable, because then negative and positive errors will not cancel each other out, which could lead to an artificially optimistic aggregate performance estimate. Figure 9.10 shows the mean and median errors for the tracking example using the processing algorithms discussed above. The mean error is the expected error in the probabilistic sense, and it is sensitive to outliers. For instance, if one or a few errors are extremely large, these will pull the mean to a noticeably larger value, possibly misrepresenting the fact that the algorithm is usually close to the correct result. The median is much less sensitive to outliers.

A more detailed description of the error is based on the cumulative error distribution. This shows, for different values of the error e^*, how

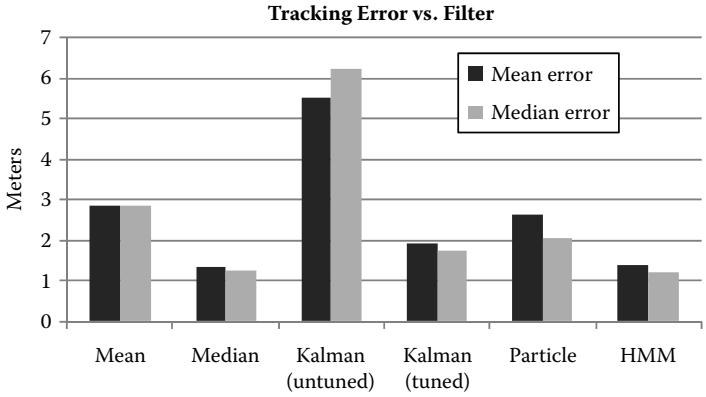

FIGURE 9.10 The mean and median error of six different processing algorithms applied to the noisy data in Figure 9.1.

often the resulting error is less than or equal to e^*. Figure 9.11 shows the cumulative error distributions for the example algorithms in this chapter. A steeply rising curve indicates better performance. The data used for this type of plot are also used to find percentile errors. For example, the 90th percentile error is the error value where the cumulative error curve crosses

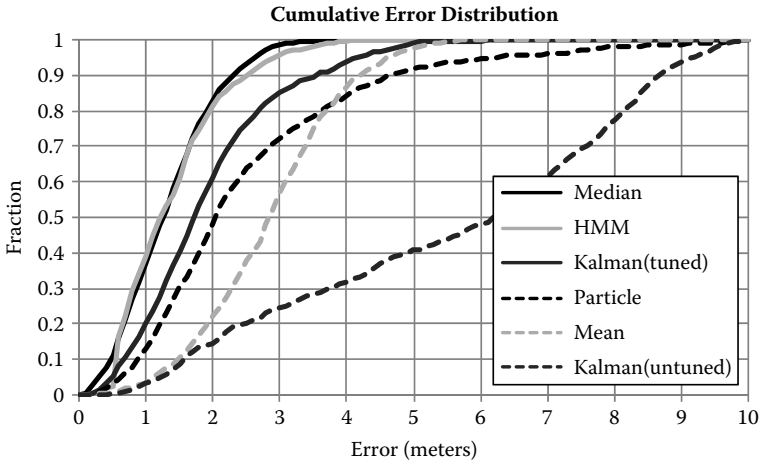

FIGURE 9.11 The cumulative error distribution gives a more detailed view of how the errors are spread. It is also useful for reading the percentile errors. For instance, the 90th percentile error for HMM is about 2.5 meters, meaning that the error is less than or equal to 2.5 meters 90% of the time.

0.9, and it means that the errors will be less than or equal to this value 90% of the time. The 50th percentile error is the same as the median, and the 100th percentile error is the maximum error.

In addition to a quantitative assessment of performance, a qualitative assessment is also useful. For instance, using this chapter's example, a qualitative assessment would include how well the algorithm works on straight segments, smooth curves, and sharp turns. Although the qualitative performance is hard to compare from algorithm to algorithm, it can be important, because sometimes qualitative differences have an effect on the quality of the upper level application. For instance, in the tracking example, sharp turns might be indicative of a subject's erratic behavior, and therefore an algorithm that tracks such turns could be better than one that tracks more accurately overall for certain applications.

Note that the results presented in this section are merely illustrations of how to present performance. They should not be used to declare that one processing algorithm is better than another. Each algorithm can be tuned for better performance, and the results presented here are not necessarily indicative of each algorithm's potential performance.

9.7.2 Presenting Discrete Performance Results

Discrete results normally come from an attempt to classify a situation into one of a plurality of categories. A good example from ubicomp is given by Lester et al. (2006), who use a sensor package to classify a subject's activity into one of sitting, standing, walking, walking up stairs, walking down stairs, riding elevator down, riding elevator up, and brushing teeth. Another common ubicomp application is to detect which "activity of daily living" a subject is engaged in. The Lester et al. (2006) paper gives its results using the time-tested and informative confusion matrix, adapted for this chapter in Table 9.1. The confusion matrix shows, for each discrete category, how often the actual category was classified into each of the whole set of categories, including the correct one. Table 9.1 shows that their classification worked well in most cases, with generally large numbers on the diagonal, meaning most activities were not confused with other activities. The best result is for "going up stairs," where the classification was correct 95% of the time. The weakest result was for standing, which was recognized correctly 55% of the time. Standing was confused with sitting 24% of the time, and with brushing teeth 10% of the time.

TABLE 9.1 Confusion Matrix[a] Showing How Often Each Activity Was Classified into Each Possible Activity

		Inferred Activities							
		Sitting	**Standing**	**Walking**	**Upstairs**	**Downstairs**	**Elevator down**	**Elevator up**	**Brushing teeth**
Actual Activities	Sitting	75%	24%	1%	0%	0%	0%	0%	0%
	Standing	29%	55%	6%	1%	0%	4%	3%	2%
	Walking	4%	7%	79%	3%	4%	1%	1%	1%
	Up stairs	0%	1%	4%	95%	0%	0%	1%	0%
	Down stairs	0%	1%	7%	0%	89%	2%	0%	0%
	Elevator down	0%	2%	1%	0%	8%	87%	1%	0%
	Elevator up	0%	2%	2%	6%	0%	3%	87%	0%
	Brushing teeth	2%	10%	3%	0%	0%	0%	0%	85%

If the classification worked perfectly, all the off-diagonals would be 0%, and the diagonals would be 100%.

[a] Data adapted from Lester et al., 2006. With kind permission of Springer Science + Business Media.

9.8 CONCLUSION

This chapter examined some algorithms for processing sequential sensor measurements. The algorithms were the mean and median filters, the Kalman filter, the particle filter, and the HMM. The process being measured is assumed to show some continuity in time, and the algorithms presented take advantage of this by looking at past measurements to help make a state estimate. For all but the mean and median filters, the algorithms use some type of dynamic model to improve accuracy. Researchers in signal processing and machine learning continue to develop new methods to process sequential sensor data, but the well-established methods in this chapter have proven useful for many ubicomp tasks. They serve as at least a good starting point for new sequential inference tasks in ubicomp. Part of the fun of using them is determining how your problem fits with the assumptions and limitations of each technique.

REFERENCES

Doucet, A., Freitas, N. D., and Gordon, N., An introduction to sequential Monte Carlo methods, in *Sequential Monte Carlo Methods in Practice*, Doucet, A., Freitas, N. D., and Gordon, N., Eds., Springer, New York, 2001.

Fox, D., Adapting the sample size in particle filters through KLD-sampling, *International Journal of Robotics Research* 22(12): 985–1003, 2003.

Gelb, A., *Applied Optimal Estimation*, Analytical Sciences Corporation, MIT Press, Cambridge, MA, 1974.

Gordon, N. J., *Bayesian Methods for Tracking*, Imperial College, University of London, London, 1994.

Hightower, J., and Borriello, G., Particle filters for location estimation in ubiquitous computing: A case study, in *Ubiquitous Computing*, Springer, Nottingham, UK, 2004, pp. 88–106.

Krumm, J., and Horvitz, E., Locadio: Inferring motion and location from Wi-Fi signal strengths, First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (Mobiquitous 2004), Cambridge, MA, USA, 2004, pp. 4–13.

Lester, J., Choudhury, T., and Borriello, G., A practical approach to recognizing physical activities, in *Pervasive Computing*, LNCS 3968, Springer-Verlag, Dublin, Ireland, 2006, pp. 1–16.

Murphy, K., and Russell, S., Rao-Blackwellised particle filtering for dynamic Bayesian networks, in *Sequential Monte Carlo Methods in Practice*, Doucet, A., Freitas, N. D., and Gordon, N., Eds., Springer-Verlag, New York, 2001, pp. 499–515.

Patterson, D., et al., Inferring high-level behavior from low-level sensors, in *Ubiquitous Computing, Ubicomp 2003*, Seattle, WA, USA, 2003, pp. 73–89.

Rabiner, L. R., A tutorial on hidden Markov models and selected applications in speech recognition, *Proceedings of IEEE* 77(2): 257–286, 1989.