Chapter 1 – Trajectory Preprocessing

Wang-Chien Lee
John Krumm
(following is John Krumm's part)

Outline for reference:

Trajectory Filtering

Location trajectories are never perfectly accurate, due to sensor noise and other factors. Sometimes the error is acceptable, such as when using GPS to identify which city a person is in. In other situations, we can apply various filtering techniques to the trajectory to smooth the noise and potentially decrease the error in the measurements. This section explains and demonstrates some conventional filtering techniques using sample data.

It is important to note that filtering is not always necessary. In fact, we rarely use it for GPS data. Filtering is important in those situations where the trajectory data is particularly noisy, or when one wants to derive other quantities from it, like speed or direction.

Sample Data

To demonstrate some of the filtering techniques in this chapter, we recorded a trajectory with a GPS logger, shown in Figure 1. The GPS logger recorded 1075

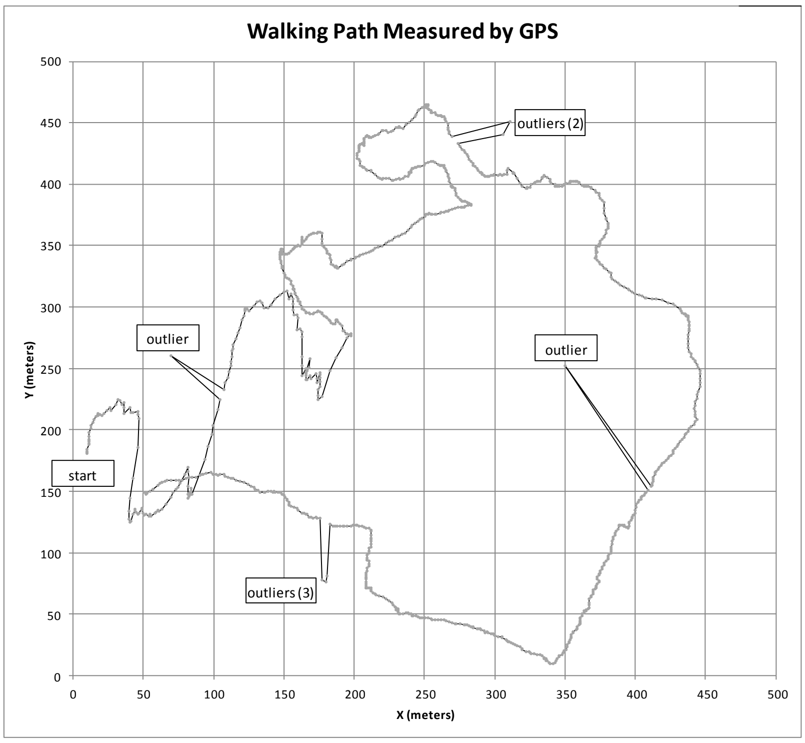

Figure 1: This is a trajectory recorded by a GPS logger. The outliers were inserted later for demonstration.

points at a rate of one per second during a short walk around the Microsoft campus in Redmond, Washington USA. For plotting, we converted the latitude/longitude points to (x,y) in meters. While the walk itself followed a casual, smooth path, the recorded trajectory shows many small spikes due to measurement noise. In addition, we manually added some outliers to simulate large deviations that sometimes appear in recorded trajectories. These outliers are marked in Figure 1. We will use this data to demonstrate the effects of the filtering techniques we describe below.

Trajectory Model

The actual, unknown trajectory is denoted as a sequence of coordinates $\boldsymbol{x}_i = (x_i, y_i)^T$. The index i represents time increments, with $i = 1 \dots N$. The boldface $\boldsymbol{x}_i$ is a two-element vector representing the x and y coordinates of the trajectory coordinate at time i.

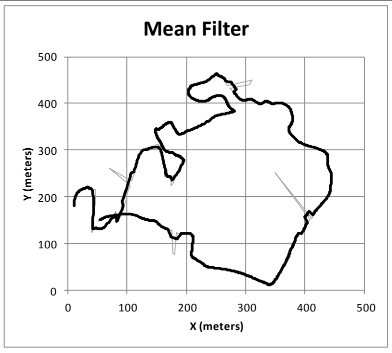	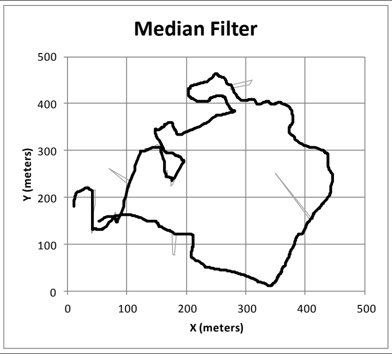
(a) Results of mean filter	(b) Results of median filter

Figure 2: The dark curve show the result of the mean filter in (a) and the median filter in (b). One advantage of the media filter is that it is less affected by outliers. The gray curve shows the original measured trajectory.

Due to sensor noise, measurements are not exact. This error is usually modeled by adding unknown, random Gaussian noise to the actual trajectory points to give the known, measured trajectory, whose coordinates are given as vectors $\boldsymbol{z}_i$ as

$$\boldsymbol{z}_i = \boldsymbol{x}_i + \boldsymbol{v}_i \tag{1}$$

The noise vector $\boldsymbol{v}_i$ is assumed to be drawn from a two-dimensional Gaussian probability density with zero mean and diagonal covariance matrix R, i.e.

$$\boldsymbol{v}_i \sim N(\boldsymbol{0}, R) \qquad \boldsymbol{R} = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix} \tag{2}$$

With the diagonal covariance matrix, this is the same as adding random noise from two different, one-dimensional Gaussian densities to x_i and y_i separately, each with zero mean and standard deviation σ. It is important to note that Equation (1) is just a model for noise from a location sensor. It is not an algorithm, but an approximation of how the measured sensor values differ from the true ones. For GPS, the Gaussian noise model above is a reasonable one [1]. In our experiments, we have observed a standard deviation σ of about four meters.

Mean and Median Filters

One simple way to smooth noise is to apply a mean filter. For a measured point $\boldsymbol{z}_i$, the estimate of the (unknown) true value is the mean of $\boldsymbol{z}_i$ and its $n-1$ predecessors in time. The mean filter can be thought of as a sliding window covering n temporally adjacent values of $\boldsymbol{z}_i$. In equation form, the mean filter is

$$\hat{\boldsymbol{x}}_i = \frac{1}{n}\sum_{j=i-n+1}^{i} \boldsymbol{z}_j \qquad (3)$$

This equation introduces another notational convention: $\hat{\boldsymbol{x}}_i$ is the estimate of $\boldsymbol{x}_i$.

Figure 2(a) shows the result of the mean filter with $n = 10$. The resulting curve is smoother.

The mean filter as given in Equation (3) is a so-called "causal" filter, because it only depends on values in the past to compute the estimate $\hat{\boldsymbol{x}}_i$. In fact, all the filters discussed in this chapter are causal, meaning they can be sensibly applied to real time data as it arrives. For post-processing, one could use a non-causal mean filter whose sliding window takes into account both past and future values to compute $\hat{\boldsymbol{x}}_i$.

One disadvantage of the mean filter is that it introduces lag. If the true underlying value $\boldsymbol{x}_i$ changes suddenly, the estimate from the mean filter will respond only gradually. So while a larger sliding window (larger value of n) makes the estimates smoother, the estimates will also tend to lag changes in $\boldsymbol{x}_i$. One way to mitigate this problem is to use a weighted mean, where more recent values of $\boldsymbol{z}_i$ are given more weight.

Another disadvantage of the mean filter is its sensitivity to outliers. From Figure 2(a), it is clear that the artificially introduced outliers noticeable pull away the estimated curve from the data. In fact, it is possible to find an outlier value to pull the mean to any value we like.

One way to mitigate the outlier problem is to use a median filter rather than a mean filter. The median filter simply replaces the mean filter's mean with a median. The equation for the median filter that corresponds to the mean filter in Equation (3) is

$$\hat{\boldsymbol{x}}_i = \text{median}\{\boldsymbol{z}_{i-n+1}, \boldsymbol{z}_{i-n+2}, \dots, \boldsymbol{z}_{i-1}, \boldsymbol{z}_i\} \qquad (4)$$

Figure 2(b) shows the result of the median filter, where it is clear that it is less sensitive to outliers and still gives a smooth result.

The mean and median filters are both simple and effective at smoothing a trajectory. They both suffer from lag. More importantly, they are not designed to help estimate higher order variables like speed. In the next two sections, we discuss the Kalman filter and the particle filter, two more advanced techniques that reduce lag and can be designed to estimate more than just location.

Kalman Filter

The mean and median filters use no model of the trajectory. More sophisticated filters, like the Kalman and particle filters, model both the measurement noise (as given by Equation (1)) and the dynamics of the trajectory.

For the Kalman filter, a simple example is smoothing trajectory measurements from something arcing through the air affected only by gravity, such as a soccer ball. While measurements of the ball's location, perhaps from a camera, are noisy, we can also impose constraints on the ball's trajectory from simple laws of physics. The trajectory estimate from the Kalman filter is a tradeoff between the measurements and the motion model. Besides giving estimates that obey the laws of physics, the Kalman filter gives principled estimates of higher order motion states like speed.

The subsections below develop the model for the Kalman filter for the example trajectory from above. We use notation from the book by Gelb, which is one of the standard references for Kalman filtering [2].

Measurement Model

While the mean and median filters can only estimate what is directly measured, the Kalman filter can estimate other variables like speed and acceleration. In order to do this, the Kalman formulation makes a distinction between what is measured and what is estimated, as well as formulating a linear relationship between the two.

As above, we assume that the measurements of the trajectory are taken as noisy values of x and y:

$$\boldsymbol{z}_i = \begin{pmatrix} z_i^{(x)} \\ z_i^{(y)} \end{pmatrix} \tag{5}$$

Here $z_i^{(x)}$ and $z_i^{(y)}$ are noisy measurements of the x and y coordinates.

The Kalman filter gives estimates for the state vector, which describes the full state of the object being tracked. In our case, the state vector will include both the object's location and velocity:

$$\boldsymbol{x}_i = \begin{pmatrix} x_i \\ y_i \\ s_i^{(x)} \\ s_i^{(y)} \end{pmatrix} \tag{6}$$

The elements x_i and y_i are the true, unknown coordinates at time i, and $s_i^{(x)}$ and $s_i^{(y)}$ are the x and y components of the true, unknown velocity at time i. The Kalman filter will produce an estimate of $\boldsymbol{x}_i$, which includes velocity, even though this is not directly measured.

The relationship between the measurement vector $\boldsymbol{z}_i$ and the state vector $\boldsymbol{x}_i$ is

$$\boldsymbol{z}_i = H_i \boldsymbol{x}_i + \boldsymbol{v}_i \tag{7}$$

where H_i, the measurement matrix, translates between $\boldsymbol{x}_i$ and $\boldsymbol{z}_i$. For our example, H_i expresses the fact that we are measuring x_i and y_i to get $z_i^{(x)}$ and $z_i^{(y)}$, but we are not measuring velocity. Thus,

$$H_i = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{8}$$

H_i also neatly accounts for the dimensionality difference between $\boldsymbol{x}_i$ and $\boldsymbol{z}_i$. While the subscript on H_i means it could change with time, it does not in our example.

The noise vector $\boldsymbol{v}_i$ in Equation (7) is the same as the zero-mean, Gaussian noise vector in Equation (2). Thus Equation (7) is how the Kalman filter models measurement noise. In fact, Gaussian noise has been proposed as a simple model of GPS noise [1], and for our example it would be reasonable to set the measurement noise σ to a few meters.

Dynamic Model

If the first half of the Kalman filter model is measurement, the second half is dynamics. The dynamic model approximates how the state vector $\boldsymbol{x}_i$ changes with time. Like the measurement model, it uses a matrix and added noise:

$$\boldsymbol{x}_i = \Phi_{i-1} \boldsymbol{x}_{i-1} + \boldsymbol{w}_{i-1} \tag{9}$$

This gives $\boldsymbol{x}_i$ as a function of its previous value $\boldsymbol{x}_{i-1}$. The system matrix Φ_{i-1} gives the linear relationship between the two. For the example problem, we have

$$\Phi_{i-1} = \begin{bmatrix} 1 & 0 & \Delta t_i & 0 \\ 0 & 1 & 0 & \Delta t_i \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10}$$

Here Δt_i is the elapsed time between the state at time i and time $i - 1$. Recalling the state vector from Equation (6), the top two rows of the system matrix say that $x_i = x_{i-1} + \Delta t_i s_i^{(x)}$ and similarly for y_i. This is standard physics for a particle with constant velocity.

The bottom two rows of system matrix say $s_i^{(x)} = s_{i-1}^{(x)}$ and $s_i^{(y)} = s_{i-1}^{(y)}$, which means the velocity does not change. Of course, we know this is not true, or else the trajectory would be straight with no turns. The dynamic model accounts for its own inaccuracy with the noise term $\boldsymbol{w}_{i-1}$. This is another zero-mean Gaussian noise term. For our example, we have

$$\boldsymbol{w}_i \sim N(\mathbf{0}, Q_i) \qquad Q_i = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_s^2 & 0 \\ 0 & 0 & 0 & \sigma_s^2 \end{bmatrix} \tag{11}$$

With the first two rows of zeros, this says that the relationship between location and velocity (e.g. $x_i = x_{i-1} + \Delta t_i s_i^{(x)}$) is exact. However, the last two rows say that the assumption in the system matrix about constant velocity is not quite true, but that the velocity is noisy, i.e. $s_i^{(x)} = s_i^{(x)} + N(0, \sigma_s^2)$. This is how the Kalman filter maintains its assumption about the linear relationship between the state vectors over time, yet manages to account for the fact that the dynamic model does not account for everything.

Entire Kalman Filter Model

The Kalman filter requires a measurement model and dynamic model, both discussed above. It also requires assumptions about the initial state and uncertainty of the initial state. Here are the all the required elements:

H_i – measurement matrix giving measurement $\boldsymbol{z}_i$ from state $\boldsymbol{x}_i$, Equation (8).

R_i – measurement noise covariance matrix, Equation (2).

Φ_{i-1} -- system matrix giving state $\boldsymbol{x}_i$ from $\boldsymbol{x}_{i-1}$, Equation (10).

Q_i -- system noise covariance matrix, Equation (11).

$\hat{\boldsymbol{x}}_0$ – initial state estimate.

P_0 – initial estimate of state error covariance.

The initial state estimate can usually be estimated from the first measurement. For our example, the initial position came from $\boldsymbol{z}_0$, and the initial velocity was taken as zero. For P_0, a reasonable estimate for this example is

$$P_0 = \begin{bmatrix} \sigma^2 & 0 & 0 & 0 \\ 0 & \sigma^2 & 0 & 0 \\ 0 & 0 & \sigma_s^2 & 0 \\ 0 & 0 & 0 & \sigma_s^2 \end{bmatrix} \qquad (12)$$

The value of σ is an estimate of the sensor noise for GPS. For our example, we set $\sigma = 4\ \text{meters}$ based on earlier experiments with our particular GPS logger. We set $\sigma_s = 6.62\ \text{meters/second}$, which we computed by looking at the changes in velocity estimated naively from the measurement data.

Kalman Filter

For the derivation of the Kalman filter, see [2]. The result is a two-step algorithm that first extrapolates the current state to the next state using the dynamic model. In equations, this is

$$\hat{\boldsymbol{x}}_i^{(-)} = \Phi_{i-1}\hat{\boldsymbol{x}}_{i-1}^{(+)} \qquad (13)$$

$$P_i^{(-)} = \Phi_{i-1}P_{i-1}^{(+)}\Phi_{i-1}^T + Q_{i-1} \qquad (14)$$

The terms in Equation (13) should be familiar. The $^{(-)}$ superscript refers to the extrapolated estimate of the state vector, and the $^{(+)}$ superscript refers to the extrapolated value of the state vector. Equation (14) is interesting in that it concerns an estimate P_i of the covariance of the state vector estimate, giving some idea of the error associated with the estimate.

The first step of the Kalman filter is pure extrapolation, with no use of measurements. The second step incorporates the current measurement to make new estimates. The equations are

$$K_i = P_i^{(-)}H_i^T\left(H_iP_i^{(-)}H_i^T + R_i\right)^{-1} \qquad (15)$$

$$\hat{\boldsymbol{x}}_i^{(+)} = \hat{\boldsymbol{x}}_i^{(-)} + K_i\left(\boldsymbol{z}_i - H_i\hat{\boldsymbol{x}}_i^{(-)}\right) \qquad (16)$$

$$P_i^{(+)} = (I - K_iH_i)P_i^{(-)} \qquad (17)$$

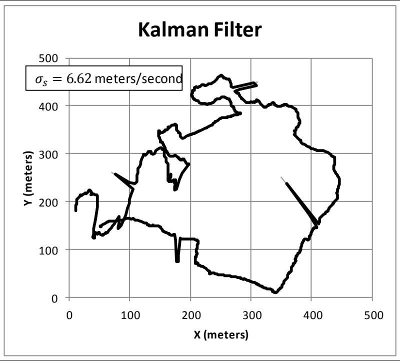	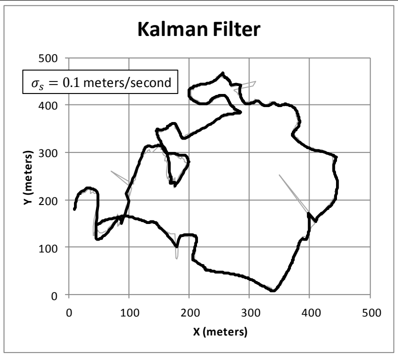
(c) Kalman filter with $\sigma_s = 6.62\ \text{meters/second}$	(d) Kalman filter with $\sigma_s = 0.1\ \text{meters/second}$

Figure 3: The dark curve show the result of the Kalman filter. In (a), the process noise σ_s comes from an estimate on the original noisy data. The process noise in (b) is much smaller, leading to a smoother filtered trajectory.

Equation (15) gives the Kalman gain matrix K_i. It is used in Equation (16) to give the state estimate $\hat{\boldsymbol{x}}_i^{(+)}$ and in Equation (17) to give the state covariance estimate $P_i^{(+)}$.

Applying these equations to the example trajectory gives the plot in Figure 3(a).

Kalman Filter Discussion

One of the advantages of the Kalman filter over the mean and median filters is its lack of lag. There is still some intrinsic lag, because the Kalman filter depends on previous measurements for its current estimate, but also includes a dynamic model to keep it more current. Another advantage is the richer state vector. In our example, it includes velocity as well as location, making the Kalman filter a principled way to estimate velocity based on a sequence of location measurements. It is easy to add acceleration as another part of the state vector. Yet another advantage is the Kalman filter's estimate of uncertainty in the form of the covariance matrix $P_i^{(+)}$ given in Equation (17). Knowledge of the uncertainty of the estimate can be used by a higher-level algorithm to react intelligently to ambiguity, possibly by invoking another sensor or asking a user for help.

One of the mysteries of the Kalman filter is the process noise, which is embodied as σ_s in our example. In our example, this represents how much the tracked object's velocity changes between time increments. In reality, this is difficult to estimate in many cases, including our example trajectory of a pedestrian. A larger value of σ_s represents less faith in the dynamic model relative to the measurements. A smaller value puts more weight on the dynamic model, often leading to a smoother trajectory. This is illustrated in Figure 3(b) where we have reduced the value of σ_s from our original value of 6.62 meters/second to 0.1 meters/second. The resulting trajectory is indeed smoother and less distracted by outliers.

One of the main limitations of the Kalman filter is the requirement that the dynamic model be linear, i.e. that the relationship between $\boldsymbol{x}_{i-1}$ and $\boldsymbol{x}_i$ be expressed as a matrix multiplication (plus noise). Sometimes this can be solved with an extended Kalman filter, which linearizes the problem around the current value of the state. But this can be difficult for certain processes, like a bouncing ball or an object constrained by predefined paths. In addition, all the variables in the Kalman filter model are continuous, without a convenient way to represent discrete variables like the mode of transportation or goal. Fortunately, the particle filter fixes these problems, and we discuss it next.

Particle Filter

The particle filter is similar to the Kalman filter in that they both use a measurement model and a dynamic model. The Kalman filter gains efficiency by assuming linear models (matrix multiplication) plus Gaussian noise. The particle filter relaxes these assumptions for a more general, albeit generally less efficient, algorithm. But, as shown by Hightower and Borriello, particle filters are practical for tracking even on mobile devices [3].

The particle filter gets its name from the fact that it maintains a set of "particles" that each represent a state estimate. There is a new set of particles generated each time a new measurement becomes available. There are normally hundreds or thousands of particles in the set. Taken together, they represent the probability distribution of possible states. A good introduction to particle filtering is the chapter by Doucet et al. [4], and this section uses their notation.

As in the previous section on Kalman filtering, this section shows how to apply particle filtering to our example tracking problem.

Particle Filter Formulation

As with the Kalman filter discussed above, the particle filter makes estimates $\hat{\boldsymbol{x}}_i$ of a sequence of unknown state vectors $\boldsymbol{x}_i$, based on measurements $\boldsymbol{z}_i$. For our example, these vectors are formulated just as in the Kalman filter. As a reminder, the state vector $\boldsymbol{x}_i$ has four scalar elements representing location and velocity. The measurement vector $\boldsymbol{z}_i$ has two elements representing a location measurement with some degree of inaccuracy.

The particle filter's measurement model is a probability distribution $p(\boldsymbol{z}_i|\boldsymbol{x}_i)$ giving the probability of seeing a measurement given a state vector. This distribution must be provided to the particle filter. This is essentially a model of a noisy sensor which might produce many different possible measurements for a given state. It is much more general than the Kalman filter's measurement model, which is limited to $\boldsymbol{z}_i = H_i\boldsymbol{x}_i + \boldsymbol{v}_i$, i.e. a linear function of the state plus added Gaussian noise.

To stay consistent with the example, however, we will use the same measurement model as in the Kalman filter, writing it as

$$p(\boldsymbol{z}_i|\boldsymbol{x}_i) = N((x_i, y_i)^{\boldsymbol{T}}, R_i) \tag{18}$$

This says that the measurement is a Gaussian distributed around the actual location with a covariance matrix R_i from Equation (2). The measurement ignores the velocity components in $\boldsymbol{x}_i$.

While this is the same measurement model that we used for the Kalman filter, it could be much more expressive. For instance, it might be the case that the location sensor's accuracy varies with location, such as GPS in an urban canyon. The particle filter's model could accommodate this.

In addition to the measurement model, the other part of the particle filter formulation is the dynamic model, again paralleling the Kalman filter. The dynamic model is also a probability distribution, which simply gives the distribution over the current state $\boldsymbol{x}_i$ given the previous state $\boldsymbol{x}_{i-1}$: $p(\boldsymbol{x}_i|\boldsymbol{x}_{i-1})$. The analogous part of the Kalman filter model is $\boldsymbol{x}_i = \Phi_{i-1}\boldsymbol{x}_{i-1} + \boldsymbol{w}_{i-1}$ (Equation (9)). The particle filter version is much more general. For instance, it could model the fact that vehicles often slow down when climbing hills and speed up going down hills. It can also take into account road networks or paths through a building to constrain where a trajectory can go. This feature of the particle filter has proved useful in many tracking applications.

It is not necessary to write out the dynamic model $p(\boldsymbol{x}_i|\boldsymbol{x}_{i-1})$. Instead, it is sufficient to sample from it. That is, given a value of $\boldsymbol{x}_{i-1}$, we must be able to create samples of $\boldsymbol{x}_i$ that adhere to $p(\boldsymbol{x}_i|\boldsymbol{x}_{i-1})$. For our example, we will use the same dynamic model as the Kalman filter, which says that location changes deterministically as a function of the velocity and that velocity is randomly perturbed with Gaussian noise:

$$\begin{aligned}
x_{i+1} &= x_i + v_i^{(x)}\Delta t_i \\
y_{i+1} &= y_i + v_i^{(y)}\Delta t_i \\
v_{i+1}^{(x)} &= v_i^{(x)} + w_i^{(x)} \qquad & w_i^{(x)} \sim N(0, \sigma_s^2) \\
v_{i+1}^{(y)} &= v_i^{(y)} + w_i^{(y)} \qquad & w_i^{(y)} \sim N(0, \sigma_s^2)
\end{aligned} \tag{19}$$

The above is a recipe for generating random samples of $\boldsymbol{x}_{i+1}$ from $\boldsymbol{x}_i$. Contrary to the Kalman filter, the particle filter requires actually generating random numbers, which in this case serve to change the velocity.

Finally, also paralleling the Kalman filter, we need an initial distribution of the state vector. For our example, we can say the initial velocity is zero and the initial location is a Gaussian around the first measurement with a covariance matrix R_i from Equation (2).

Particle Filter

The particle filter maintains a set of N state vectors, called particles: $\boldsymbol{x}_i^{(j)}$, $j = 1 \dots N$. There are several versions of the particle filter, but we will present the Bootstrap Filter from Gordon [5]. The initialization step is to generate N particles from the initial distribution. For our example, these particles would have zero velocity and be clustered around the initial location measurement with a Gaussian distribution as explained above. This is the first instance of how the particle filter requires actually generating random hypotheses about the state vector. This is different from the Kalman filter which generates state estimates and uncertainties directly. We will call these particles $\boldsymbol{x}_0^{(j)}$.

With a set of particles and $i > 1$, the first step is "importance sampling," which uses the dynamic model $p(\boldsymbol{x}_i|\boldsymbol{x}_{i-1})$ to probabilistically simulate how the particles change over one time step. This is analogous to the extrapolation step in the Kalman filter in that it proceeds without regard to the measurement. For our example, this means invoking Equation (19) to create $\tilde{\boldsymbol{x}}_i$. The tilde (~) indicates extrapolated values. Note that this involves actually generating random numbers for the velocity update.

The next step computes “importance weights” for all the particles using the measurement model. The importance weights are

$$\tilde{w}_i^{(j)} = p\left(\boldsymbol{z}_i \middle| \tilde{\boldsymbol{x}}_i^{(j)}\right) \tag{20}$$

Larger importance weights correspond to particles that are better supported by the measurement. The important weights are then normalized so they sum to one.

The last step in the loop is the “selection step” when a new set of N particles $\boldsymbol{x}_i^{(j)}$ is selected at random from the $\tilde{\boldsymbol{x}}_i^{(j)}$ based on the normalized importance weights. The probability of selecting an $\tilde{\boldsymbol{x}}_i^{(j)}$ for the new set of particles is proportional to its importance weight $\tilde{w}_i^{(j)}$. It is not unusual to select the same $\tilde{\boldsymbol{x}}_i^{(j)}$ more than once if it has a larger importance weight. This is the last step in the loop, and processing then returns to the importance sampling step with the new set of particles.

While the particles give a distribution of state estimates, one can compute a single estimate with a weighted sum:

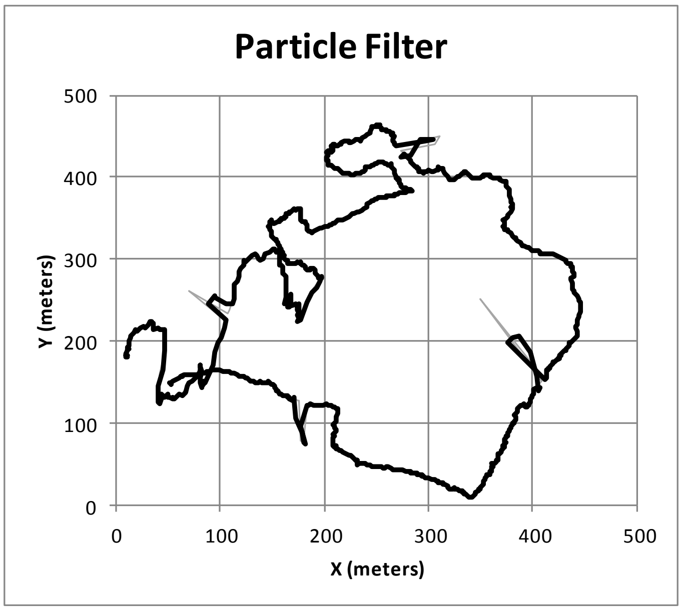

Figure 4: This is the result of the particle filter. It is similar to the result of the Kalman filter in Figure 3(a) since it uses the same measurement model, dynamic model, and noise assumptions.

$$\hat{\boldsymbol{x}}_i = \sum_{j=1}^{N} \tilde{w}_i^{(j)} \tilde{\boldsymbol{x}}_i^{(j)} \tag{21}$$

Applying the particle filter to our example problem gives the result in Figure 4. We used $N = 1000$ particles. This result looks similar to the Kalman filter result, since we used the same measurement model, dynamic model, and noise in both.

Particle Filter Discussion

One potential disadvantage of the particle filter is computation time, which is affected by the number of particles. More particles generally give a better result, but at the expense of computation. Fox gives a method to choose the number of particles based on bounding the approximation error [6].

Even though the particle filter result looks similar to the Kalman filter result in our example, it is important to understand that the particle filter has the potential to be much richer. As mentioned previously, it could be made sensitive to a network of roads or walking paths. It could include a discrete state variable representing the mode of transportation, e.g. walking, bicycling, in a car, or on a bus.

While it is tempting to add many variables to the state vector, the cost is often more particles required to make a good state estimate. One solution to this problem is the Rao-Blackwellized particle filter [7]. It uses a more conventional filter, like Kalman, to track some of the state variables and a particle filter for the others.

1. van_Diggelen, F. (2007) *GNSS Accuracy: Lies, Damn Lies, and Statistics*. GPS World.
2. Gelb, A., et al., *Applied Optimal Estimation*. 1974: The MIT Press.
3. Hightower, J. and G. Borriello, *Particle Filters for Location Estimation in Ubiquitous Computing: A Case Study*, in *6th International Conference on Ubiquitous Computing (UbiComp 2004)*. 2004, Springer: Nottingham, UK. p. 88-106.
4. Doucet, A., N.d. Freitas, and N. Gordon, *An Introduction to Sequential Monte Carlo Methods*, in *Sequential Monte Carlo Methods in Practice*, A. Doucet, N.d. Freitas, and N. Gordon, Editors. 2001, Springer: New York.
5. Gordon, N.J., *Bayesian Methods for Tracking*. 1994, Imperial College, University of London: London.
6. Fox, D., *Adapting the Sample Size in Particle Filters Through KLD-Sampling* The International Journal of Robotics Research, 2003. **22**(12): p. 985-1003.

7. Murphy, K. and S. Russell, *Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks*, in *Sequential Monte Carlo Methods in Practice*, A. Doucet, N.d. Freitas, and N. Gordon, Editors. 2001, Springer-Verlag: New York. p. 499-515.