Chapter 3

Kalman Filter and its variants

Reading

1. Barfoot, Chapter 3, 4 for Kalman filter
2. Thrun, Chapter 3 for Kalman filter, Chapter 4 for particle filters
3. Russell Chapter 15.4 for Kalman filter

Hidden Markov Models (HMMs) which we discussed in the previous chapter were a very general class of models. As a consequence algorithms for filtering, smoothing and decoding that we prescribed for the HMM are also very general. In this chapter we will consider the situation when we have a little more information about our system. Instead of writing the state transition and observation matrices as arbitrary matrices, we will use the framework of linear dynamical systems to model them better. Since we know the system a bit better, algorithms that we prescribe for these models for solving filtering, smoothing and decoding will also be more efficient. We will almost exclusively focus on the filtering problem in this chapter. The other two, namely smoothing and decoding, can also be solved easily using these ideas but are less commonly used for these systems.

3.1 Background

Multi-variate random variables and linear algebra For d-dimensional random variables $X, Y \in \mathbb{R}^d$ we have

$$\mathrm{E}[X + Y] = \mathrm{E}[X] + \mathrm{E}[Y];$$

this is actually more surprising than it looks, it is true regardless of whether X, Y are correlated. The covariance matrix of a random variable is defined as

$$\text{cov}(X) = \text{E}[(X - \text{E}[X])\,(X - \text{E}[X])^\top];$$

we will usually denote this by $\Sigma \in \mathbb{R}^{d\times d}$. Note that the covariance matrix is, by construction, symmetric and positive semi-definite. This means it can be factorized as

$$\Sigma = U\Lambda U^\top$$

where $U \in \mathbb{R}^{d\times d}$ is an orthonormal matrix (i.e., $UU^\top = I$) and Λ is a diagonal matrix with non-negative entries. The trace of a matrix is the sum of its diagonal entries. It is also equal to the sum of its eigenvalues, i.e.,

$$\text{tr}(\Sigma) = \sum_{i=1}^{d} \Sigma_{ii} = \sum_{i=1}^{d} \lambda_i(\Sigma)$$

where $\lambda_i(S) \geq 0$ is the i^{th} eigenvalue of the covariance matrix S. The trace is a measure of the uncertainty in the multi-variate random variable X, if X is a scalar and takes values in the reals then the covariance matrix is also, of course, a scalar $\Sigma = \sigma^2$.

A few more identities about the matrix trace that we will often use in this chapter are as follows.

- For matrices A, B we have

  $$\text{tr}(AB) = \text{tr}(BA);$$

  the two matrices need not be square themselves, only their product does.
- For $A, B \in \mathbb{R}^{m\times n}$

  $$\text{tr}(A^\top B) = \text{tr}(B^\top A) = \sum_{i=1}^{m}\sum_{j=1}^{n} B_{ij}A_{ij}.$$

  This operation can be thought of as taking the inner product between two matrices.

Gaussian/Normal distribution We will spend a lot of time working with the Gaussian/Normal distribution. The multi-variate d-dimensional Normal distribution has the probability density

❓ Why is it so ubiquitous?

$$f(x) = \frac{1}{\sqrt{\det(2\pi\Sigma)}}\ \exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\}$$

where $\mu \in \mathbb{R}^d, \Sigma \in \mathbb{R}^{d\times d}$ denote the mean and covariance respectively. You should commit this formula to memory. In particular remember that

$$\int_{x\in\mathbb{R}^d} \exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\}\ \text{d}x = \sqrt{\det(2\pi\Sigma)}$$

which is simply expressing the fact that the probability density function integrates to 1.

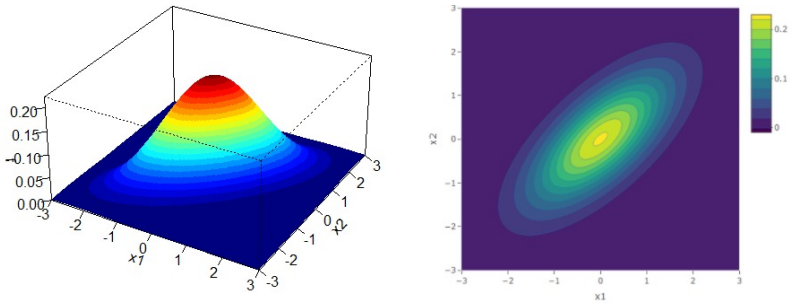


Figure 3.1: Probability density (left) and iso-probability contours (right) of a bi-variate Normal distribution. Warm colors denote regions of high probability.

Given two Gaussian rvs. $X, Y \in \mathbb{R}^d$ and $Z = X + Y$ we have

$$\mathrm{E}[Z] = \mathrm{E}[X + Y] = \mathrm{E}[X] + \mathrm{E}[Y]$$

with covariance

$$\mathrm{cov}(Z) = \Sigma_Z = \Sigma_X + \Sigma_Y + \Sigma_{XY} + \Sigma_{YX}$$

where

$$\mathbb{R}^{d\times d} \ni \Sigma_{XY} = \mathrm{E}\left[(X - \mathrm{E}[X])\,(Y - \mathrm{E}[Y])^\top\right];$$

the matrix S_{YX} is defined similarly. If X, Y are independent (or uncorrelated) the covariance simplifies to

$$\Sigma_Z = \Sigma_X + \Sigma_Y.$$

If we have a linear function of a Gaussian random variable X given by $Y = AX$ for some *deterministic* matrix A then Y is also Gaussian with mean

$$\mathrm{E}[Y] = \mathrm{E}[AX] = A\,\mathrm{E}[X] = A\mu_X$$

and covariance

$$\begin{aligned}\mathrm{cov}(Y) &= \mathrm{E}[(AX - A\mu_X)(AX - A\mu_X)^\top] \\ &= \mathrm{E}[A(X - \mu_X)(X - \mu_X)^\top A^\top] \\ &= A\,\mathrm{E}[(X - \mu_X)(X - \mu_X)^\top]A^\top \\ &= A\Sigma_X A^\top.\end{aligned} \tag{3.1}$$

This is an important result that you should remember.

3.2 Linear state estimation

With that background, let us now look at the basic estimation problem. Let $X \in \mathbb{R}^d$ denote the true state of a system. We would like to build an estimator for this state, this is denote by

$$\hat{X}.$$

An estimator is any quantity that indicates our belief of what X is. The estimator is created on the basis of observations and we will therefore model it as a random variable. We would like the estimator to be unbiased, i.e.,

$$\mathrm{E}[\hat{X}] = X;$$

this expresses the concept that if we were to measure the state of the system many times, say using many sensors or multiple observations from the same sensor, the resultant estimator $\hat{X}$ is correct on average. The error in our belief is

$$\tilde{X} = \hat{X} - X.$$

The error is zero-mean $\mathrm{E}[\tilde{X}] = 0$ and its covariance $\Sigma_{\tilde{X}}$ is called the covariance of the estimator.

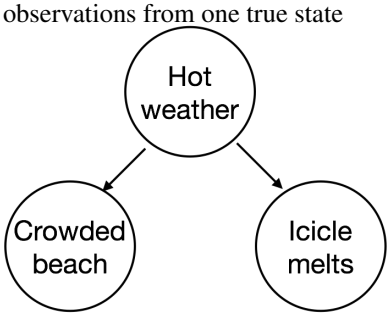
Optimally combining two estimators Let us now imagine that we have two estimators $\hat{X}_1$ and $\hat{X}_2$ for the same true state X. We will assume that the two estimators were created independently (say different sensors) and therefore are conditionally independent random variables given the true state X Say both of them are unbiased but each of them have a certain covariance of the error

$$\Sigma_{\tilde{X}_1} \text{ and } \Sigma_{\tilde{X}_2}.$$

We would like to combine the two to obtain a *better* estimate of what the state could be. *Better* can mean many different quantities depending upon the problem but in general in this course we are interested in improving the error covariance. Our goal is then

> Given two estimators $\hat{X}_1$ and $\hat{X}_2$ of the true state X combine them to obtain a new estimator
>
> $$\hat{X} = \text{some function}(\hat{X}_1, \hat{X}_2)$$
>
> which has the best error covariance $\mathrm{tr}(\Sigma_{\tilde{X}})$.

ℹ Conditionally independent observations from one true state



3.2.1 One-dimensional Gaussian random variables

Consider the case when $\hat{X}_1, \hat{X}_2 \in \mathbb{R}$ are Gaussian random variables with means μ_1, μ_2 and variances σ_1^2, σ_2^2 respectively. Assume that both are unbiased estimators of $X \in \mathbb{R}$. Let us combine them linearly to obtain a new estimator

$$\hat{X} = k_1\hat{X}_1 + k_2\hat{X}_2.$$

How should we pick the coefficients k_1, k_2? We would of course like the new estimator to be unbiased, so

$$\begin{aligned} \mathrm{E}[\hat{X}] = \mathrm{E}[k_1\hat{X}_1 + k_2\hat{X}_2] = (k_1 + k_2)X = X \\ \Rightarrow k_1 + k_2 = 1. \end{aligned}$$

The variance of the $\hat{X}$ is

$$\mathrm{var}(\hat{X}) = k_1^2\sigma_1^2 + k_2^2\sigma_2^2 = k_1^2\sigma_1^2 + (1 - k_1)^2\sigma_2^2.$$

The optimal k_1 that leads to the smallest variance is thus given by

$$k_1 = \frac{\sigma_2^2}{\sigma_1^2 + \sigma_2^2}.$$

We set the derivative of $\mathrm{var}(\hat{X})$ with respect to k_1 to zero to get this. The final estimator is

$$\hat{X} = \frac{\sigma_2^2}{\sigma_1^2 + \sigma_2^2}\hat{X}_1 + \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2}\hat{X}_2. \tag{3.2}$$

It is unbiased of course and has variance

$$\sigma_{\hat{X}}^2 = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2}.$$

Notice that since $\sigma_2^2/(\sigma_1^2 + \sigma_2^2) < 1$, the variance of the new estimator is smaller than either of the original estimators. This is an important fact to remember, combining two estimators *always* results in a better estimator.

Some comments about the optimal combination.

- It is easy to see that if $\sigma_2 \gg \sigma_1$ then the corresponding estimator, namely $\hat{X}_2$ gets less weight in the combination. This is easy to understand, if one of our estimates is very noisy, we should rely less upon it to obtain the new estimate. In the limit that $\sigma_2 \to \infty$, the second estimator is not considered at all in the combination.
- If $\sigma_1 = \sigma_2$, the two estimators are weighted equally and since $\sigma_{\hat{X}}^2 = \sigma_1^2/2$ the variance reduces by half after combination.
- The minimal variance of the combined estimator is not zero. This is easy to see because if we have two noisy estimates of the state, combining them need not lead to us knowing the true state with certainty.

3.2.2 General case

Let us now perform the same exercise for multi-variate Gaussian random variables. We will again combine the two estimators linearly to get

$$\hat{X} = K_1\hat{X}_1 + K_2\hat{X}_2$$

where $K_1, K_2 \in \mathbb{R}^{d\times d}$ are matrices that we would like to choose. In order for the estimator to be unbiased we again have the condition

$$\begin{aligned}\mathrm{E}[\hat{X}] = \mathrm{E}[K_1\hat{X}_1 + K_2\hat{X}_2] &= (K_1 + K_2)X = X \\ \Rightarrow K_1 + K_2 &= I_{d\times d}.\end{aligned}$$

The covariance of $\hat{X}$ is

$$\begin{aligned}\Sigma_{\tilde{X}} &= K_1\Sigma_1 K_1^\top + K_2\Sigma_2 K_2^\top \\ &= K_1\Sigma_1 K_1^\top + (I - K_1)\Sigma_2(I - K_1)^\top.\end{aligned}$$

Just like the minimized the variance in the scalar case, we will minimize the trace of this covariance matrix. We know that the original covariances Σ_1 and Σ_2 are symmetric. We will use the following identity for the partial derivative of a matrix product

$$\frac{\partial}{\partial A}\,\mathrm{tr}(ABA^\top) = 2AB \tag{3.3}$$

for a symmetric matrix B. Minimizing $\mathrm{tr}(\Sigma_{\tilde{X}})$ with respect to K_1 amounts to setting

$$\frac{\partial}{\partial K_1}\,\mathrm{tr}(\Sigma_{\tilde{X}}) = 0$$

which yields

$$\begin{aligned}0 &= K_1\Sigma_1 - (I - K_1)\Sigma_2 \\ \Rightarrow K_1 = \Sigma_2(\Sigma_1 + \Sigma_2)^{-1} &\text{ and } K_2 \quad = \Sigma_1(\Sigma_1 + \Sigma_2)^{-1}.\end{aligned}$$

The optimal way to combine the two estimators is thus

$$\hat{X} = \Sigma_2(\Sigma_1 + \Sigma_2)^{-1}\hat{X}_1 + \Sigma_1(\Sigma_1 + \Sigma_2)^{-1}\hat{X}_2. \tag{3.4}$$

You should consider the similarities of this expression with the one for the scalar case in (3.2). The same broad comments hold, i.e., if one of the estimators has a very large variance, that estimator is weighted less in the combination.

3.2.3 Incorporating Gaussian observations of a state

Let us now imagine that we have a sensor that can give us observations of the state. The development in this section is analogous to our calculations in Chapter 2 with the recursive application of Bayes rule or the observation matrix of the HMM. We will consider a special type of sensor that gives observations

$$\mathbb{R}^p \ni Y = CX + \nu \tag{3.5}$$

which is a linear function of the true state $X \in \mathbb{R}^d$ with the matrix $C \in \mathbb{R}^{p\times d}$ being something that is unique to the particular sensor. This observation is not precise and we will model the sensor as having zero-mean Gaussian noise

$$\nu \sim N(0, Q)$$

of covariance $Q \in \mathbb{R}^{p \times p}$. Notice something important here, the dimensionality of the observations need not be the same as the dimensionality of the state. This should not be surprising, after all the the number of observations in the HMM need not be the same as the number of the states in the Markov chain.

We will solve the following problem. Given an existing estimator $\hat{X}'$ we want to combine it with the observation Y to update the estimator to $\hat{X}$, in the best way, i.e., in a way that gives the minimal variance. We will again use a linear combination

$$\hat{X} = K'\hat{X}' + KY.$$

Again we want the estimator to be unbiased, so we set

$$\begin{aligned} \mathrm{E}[\hat{X}] &= \mathrm{E}[K'\hat{X}' + KY] \\ &= K'X + K\,\mathrm{E}[Y] \\ &= K'X + K\,\mathrm{E}[CX + \nu] \\ &= K'X + KCX \\ &= X. \end{aligned}$$

to get that

$$\begin{aligned} I &= K' + KC. \\ \Rightarrow \hat{X} &= (I - KC)\hat{X}' + KY \\ &= \hat{X}' + K(Y - C\hat{X}'). \end{aligned} \tag{3.6}$$

This is special form which you will do well to remember. The old estimator $\hat{X}'$ gets an additive term $K(Y - C\hat{X}')$. For reasons that will soon become clear, we call this term

$$\text{innovation} = Y - C\hat{X}'.$$

Let us now optimize K as before to compute the estimator with minimal variance. We will make the following important assumption in this case.

> We will assume that the observation Y is independent of the estimator $\hat{X}'$ given X. This is a natural assumption because presumably our original estimator $\hat{X}'$ was created using past observations and the present observation Y is therefore independent of it given the state X.

The covariance of $\hat{X}$ is

$$\Sigma_{\tilde{X}} = (I - KC)\Sigma_{\tilde{X}'}(I - KC)^\top + KQK^\top.$$

We optimize the trace of $\Sigma_{\tilde{X}}$ with respect to K to get

$$\begin{aligned} 0 &= \frac{\partial}{\partial K}\,\mathrm{tr}(\Sigma_{\tilde{X}}) \\ 0 &= -2(I - KC)\Sigma_{\tilde{X}'}C^\top + 2KQ \\ \Rightarrow \Sigma_{\tilde{X}'}C^\top &= K(C\Sigma_{\tilde{X}'}C^\top + Q) \\ \Rightarrow K &= \Sigma_{\tilde{X}'}C^\top(C\Sigma_{\tilde{X}'}C^\top + Q)^{-1}. \end{aligned}$$

The matrix $K \in \mathbb{R}^{d \times p}$ is called the "Kalman gain" after Rudoph Kalman who developed this method in the 1960s.

Kalman gain This is an important formula and it helps to have a mnemonic and a slightly simpler notation to remember it by. If Σ' is the covariance of the previous estimator, Q is the covariance of the zero-mean observation and C is the matrix that gives the observation from the state, then the Kalman gain is

$$K = \Sigma_{\tilde{X}'} C^\top (C \Sigma_{\tilde{X}'} C^\top + Q)^{-1}. \tag{3.7}$$

and the new estimator for the state is

$$\hat{X} = \hat{X}' + K(Y - C\hat{X}').$$

The covariance of the updated estimator $\hat{X}$ is given by

$$\begin{aligned} \Sigma_{\tilde{X}} &= (I - KC)\Sigma_{\tilde{X}'}(I - KC)^\top + KQK^\top \\ &= \left(\Sigma_{\tilde{X}'}^{-1} + C^\top Q^{-1} C\right)^{-1}. \end{aligned} \tag{3.8}$$

If $C = I$, the Kalman gain is the same expression as the optimal coefficient in (3.4). This should not be surprising because the observation is an estimator for the state.

The second expression for $\Sigma_{\tilde{X}}$ follows by substituting the value of the Kalman gain K. Yet another way of remembering this equation equation is to notice that

$$\begin{aligned} \Sigma_{\tilde{X}}^{-1} &= \Sigma_{\tilde{X}'}^{-1} + C^\top Q^{-1} C \\ K &= \Sigma_{\tilde{X}}^{-1} C^\top Q^{-1} \\ \hat{X} &= \hat{X}' + \Sigma_{\tilde{X}}^{-1} C^\top Q^{-1} \left(Y - C\hat{X}'\right). \end{aligned} \tag{3.9}$$

ℹ Derive these expressions for the Kalman gain and the covariance yourself.

3.2.4 An example

Consider the scalar case when we have multiple measurements of some scalar quantity $x \in \mathbb{R}$ corrupted by noise.

$$y_i = x + \nu_i$$

where $y_i \in \mathbb{R}$ and the scalar noise $\nu_i \sim N(0, 1)$ is zero-mean and standard Gaussian. Find the updated estimate of the state x after k such measurements; this means both the mean and the covariance of the state.

You can solve this in two ways, you can either use the measurement matrix C to be $\mathbb{1}_k = [1, \ldots, 1]$ to be a vector of all ones and apply the formula in (3.7) and (3.8) Show that the estimate $\hat{x}_k$ after k measurements has mean

and covariance

$$\mathrm{E}[\hat{x}_k] = \frac{1}{k}\sum_{i=1}^{k} y_i.$$

$$\mathrm{cov}(\hat{x}_k) = C^\top C^{-1} = \frac{1}{k}.$$

If we take one more measurement $y_{k+1} = x + \nu_{k+1}$ with noise $\nu_{k+1} \sim N(0, \sigma^2)$, show using (3.9) that

$$\mathrm{cov}(\hat{x}_{k+1})^{-1} = \mathrm{cov}(\hat{x}_k)^{-1} + \frac{1}{\sigma^2}$$

$$\Rightarrow \mathrm{cov}(\hat{x}_{k+1}) = \frac{\sigma^2}{\sigma^2 k + 1}.$$

The updated mean using (3.9) again

$$\begin{aligned}\mathrm{E}[\hat{x}_{k+1}] &= \hat{x}_k + \mathrm{cov}(\hat{x}_{k+1})\frac{1}{\sigma^2}\left(y_{k+1} - \hat{x}_k\right)\\ &= \hat{x}_k + \frac{y_{k+1} - \hat{x}_k}{\sigma^2 k + 1}.\end{aligned}$$

You will notice that if the noise on the $k+1^{\text{th}}$ observation is very small, even after k observations, the new estimate fixates on the latest observation

$$\sigma \to 0 \Rightarrow \hat{x}_{k+1} \to y_{k+1}.$$

Similarly, if the latest observation is very noisy, the estimate does not change much

$$\sigma \to \infty \Rightarrow \hat{x}_{k+1} \to \hat{x}_k.$$

3.3 Background on linear and nonlinear dynamical systems

> The true state X need not static. We will next talk about models for how the state of the world evolves using ideas in dynamical systems.

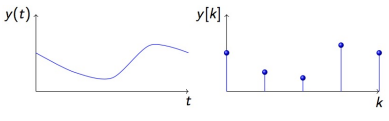
A continuous-time signal is a function that associates to each time $t \in \mathbb{R}$ a real number $y(t)$. We denote signals by

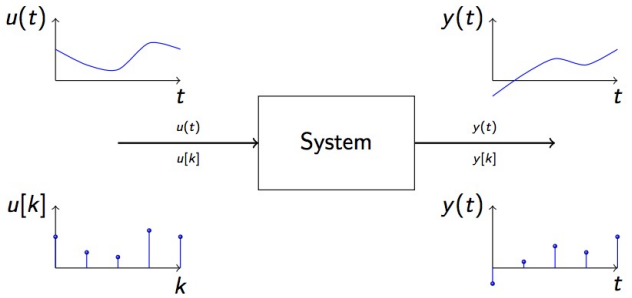
$$y : t \mapsto y(t).$$

Similarly a discrete-time signal is a function that associates to each integer k a real number $y(k)$, we have been denoting quantities like this by y_k.

A dynamical system is an operator (a box) that transforms an input signal $u(t)$ or u_k to an output $y(t)$ or y_k respectively. We call the former a continuous-time system and the latter a discrete-time system.

ℹ A continuous-time signal $y(t)$ and discrete-time signal y_k.





Almost always in robotics, we will be interested in systems that are temporally *causal*, i.e., the output at time t_0 is only a function the input *up to* time t_0. Analogously, the output at time k_0 for a discrete-time system is dependent only on the input up to time k_0. Most systems in the physical world at temporally causal.

❓ Can you give an example of a dynamical system that is non-causal? Think of how a DVD Ripper, or a pre-programmed acrobatic maneuver on a plane works.

State of a system We know that if the system is causal, in order to compute its output at a time t_0, we only need to know all the input from time $t = (-\infty, t_0]$. This is a lot of information. The concept of a state, about which we have been cavalier until now helps with this. The state $x(t_1)$ of a causal system at time t_1 is the information needed, together with the input u between times t_1 and t_2 to *uniquely compute* the output $y(t_2)$ at time t_2, for all times $t_2 \geq t_1$. In other words, the state of a system summarizes the whole history of what happened between $(-\infty, t_1)$.

Typically the state of a system is a d-dimensional vector in $\mathbb{R}^d$. The dimension of a system is the minimum d required to define a state.

❓ Discuss some examples of the state.

Is the state of a system uniquely defined?

3.3.1 Linear systems

A system is called a linear system if for any two input *signals* u_1 and u_2 and any two real numbers a, b

$$u_1 \to y_1$$
$$u_2 \to y_2$$
$$au_1 + bu_2 \to ay_1 + by_2.$$

Linearity is a very powerful property. For instance, it suggests that if we can decompose a complicated input into the sum of simple signals, then the output of the system is also a sum of the outputs of these simple signals. For example, if we can write the input as a Fourier series $u(t) = \sum_{i=0}^{\infty} a_i \cos(it) + b_i \sin(it)$ we can pass each of the terms in this summation to system and get the output of $u(t)$ by summing up the individual outputs.

Finite-dimensional systems can be written using a set of differential equations as follows. Consider the spring-mass system. If $z(t)$ denotes the position of the mass at time t and $u(t)$ is the force that we are applying upon it at time

t, the position of the mass satisfies the differential equation

$$m\frac{\mathrm{d}^2 z(t)}{\mathrm{d}t^2} + c\frac{\mathrm{d}z(t)}{\mathrm{d}t} + kx(t) = u(t)$$
$$\text{or } m\ddot{x} + c\dot{x} + kx = u$$

in short. Here m is the mass of the block, c is the damping coefficient of the spring and k is the spring force constant. Let us define

$$z_1(t) := z(t)$$
$$z_2(t) := \frac{\mathrm{d}z(t)}{\mathrm{d}t}.$$

We can now rewrite the dynamics as

$$\begin{bmatrix}\dot{z_1}\\ \dot{z_2}\end{bmatrix} = \begin{bmatrix}0 & 1\\ -k/m & -c/m\end{bmatrix}\begin{bmatrix}z_1\\ z_2\end{bmatrix} + \begin{bmatrix}0\\ 1/m\end{bmatrix}u$$

3.3.2 Linear Time-Invariant (LTI) systems

If we define the state $x(t) = \begin{bmatrix}z_1(t)\\ z_2(t)\end{bmatrix}$, then the above equation can be written as

$$\dot{x}(t) = Ax(t) + Bu(t). \tag{3.10}$$

This is a linear system that takes in the input $u(t)$ and has a state $x(t)$. You can check the conditions for linearity to be sure. It is also a linear time-invariant (LTI) system because the matrices A, B do not change with time t. **The input $u(t)$ is also typically called the control (or action, or the control input) and essentially the second half of the course is about computing good controls.**

Since the state at time t encapsulates everything that happened to the system due to the inputs $\{u(-\infty), u(t)\}$, we can say that the system computes its output $y(t)$ as a function of the state $x(t)$ and the latest input $u(t)$

$$y(t) = \text{function}(x(t), u(t))$$

If this function is linear we have

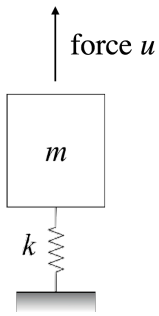
$$y(t) = Cx(t) + Du(t). \tag{3.11}$$

The pair of equations (3.10) and (3.11) together are the so-called state-space model of an LTI system. The development for discrete-time systems is completely analogous, we will have

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k\\ y_k &= Cx_k + Du_k. \end{aligned} \tag{3.12}$$

If the matrices A, B, C, D change with time, we have a time-varying system.

A second-order spring mass system



3.3.3 Nonlinear systems

Nonlinear systems are defined entirely analogously as linear systems. Imagine if we had a non-linear spring in the spring-mass system whereby the dynamics of the block was given by

$$m\ddot{z} + c\dot{z} + (k_1 z + k_2 z^2) = u.$$

The state of the system is still $x = [z_1, z_2]^\top$. But we cannot write this second-order differential equation as two first-order linear differential equations. We are forced to write

$$\dot{x} = \begin{bmatrix} \dot{z_1} \\ \dot{z_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -k_1/m - k_2 z_1/m & -c/m \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1/m \end{bmatrix} u.$$

Such systems are called nonlinear systems. We will write them succinctly as

$$\begin{aligned} \dot{x} &= f(x, u) \\ y &= g(x, u). \end{aligned} \tag{3.13}$$

The function $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ that maps the state-space and the input space to the state-space is called the dynamics of the system. Analogously, for discrete-time nonlinear systems we will have

$$\begin{aligned} x_{k+1} &= f(x_k, u_k) \\ y_k &= g(x_k, u_k). \end{aligned}$$

❓ Is the nonlinear spring-mass system time-invariant?

3.4 Markov Decision Processes (MDPs)

Let us now introduce a concept called MDPs which is very close to Markov chains that we saw in the previous chapter. In fact, you are already implementing an MDP in your HW 1 problem on the Bayes filter.

> MDPs are a model for the scenario when we do not completely know the dynamics $f(x_k, u_k)$.

This may happen for a number of reasons and it is important to appreciate them in order to understand the widespread usage of MDPs.

1. We did not do a good job of identifying the function $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$. This may happen when you are driving a car on an icy road, if you undertake the same control as you do on a clean road, you might reach a different future state x_{k+1}.

2. We did not use the correct state-space $\mathcal{X}$. You could write down the state of the car as given by $(x, y, \theta, \dot{x}, \dot{y}, \dot{\theta})$ where x, y are the Euclidean co-ordinates of the car and θ is its orientation. This is not a good model

for studying high-speed turns, which are affect by other quantities like wheel slip, the quality of the suspension etc.

We may not even know the full state sometimes. This occurs when you are modeling how users interact with an online website like Amazon.com, you'd like to model the change in state of the user from "perusing stuff" to "looking stuff to buy it" to "buying it" but there are certainly many other variables that affect the user's behavior. As another example, consider the path that an airplane takes to go from Philadelphia to Los Angeles. This path is affected by the weather at all places along the route, it'd be cumbersome to incorporate the weather to find the shortest-time path for the airplane.

3. We did not use the correct control-space U for the controller. This is akin to the second point above. The gas pedal which one may think of as the control input to a car is only one out of the large number of variables that affect the running of the car's engine.

MDPs are a drastic abstraction of all the above situations. We write

$$x_{k+1} = f(x_k, u_k) + \epsilon_k \tag{3.14}$$

where the "noise" ϵ_k is not under our control. The quantity ϵ_k is not arbitrary however, we will assume that

1. noise ϵ_k is a random variable and we know its distribution. For example, you ran your car lots of times on icy road and measured how the state x_{k+1} deviates from similar runs on a clean road. The difference between the two is modeled as ϵ_k. Note that the distribution of ϵ_k may be a function of time k.

2. noise at different timesteps $\epsilon_1, \epsilon_2, \ldots,$ is independent.

Instead of a deterministic transition for our system from x_k to x_{k+1}, we now have

$$x_{k+1} \sim \mathrm{P}(x_{k+1} \mid x_k, u_k).$$

which is just another way of writing (3.14).

The latter is a probability table of size $|\mathcal{X}| \times |\mathcal{U}| \times |\mathcal{X}|$ akin to the transition matrix of a Markov chain except that there is a different transition matrix for every control $u \in \mathcal{U}$. The former version (3.14) is more amenable to analysis. MDPs can be alternatively called stochastic dynamical systems, we will use either names for them in this course. For completeness, let us note down that linear stochastic systems will will written as

$$x_{k+1} = Ax_k + Bu_k + \epsilon_k.$$

The moral of this section is to remember that as pervasive as noise

❓ You should think about the state-space, control-space and the noise in the MDP for the Bayes filter problem in HW 1. Where do we find MDPs in real-life? There are lots of expensive robots in GRASP, e.g., a Kuka manipulator such as this https://www.youtube.com/watch?v=ym64NFC costs upwards of $100,000. Would you model it as a stochastic dynamical system?

seems in all problem formulations in this course, it models different situations depending upon the specific problem. Understanding where noise comes from is important for real-world applications.

Noise in continuous-time systems You will notice that we only talked about discrete-time systems with noise in (3.14). We can also certainly talk about continuous-time systems whose dynamics f we do not know precisely

$$\dot{x}(t) = f(x(t), u(t)) + \epsilon(t) \tag{3.15}$$

and model the gap in our knowledge as noise $\epsilon(t)$. While this may seem quite natural, it is mathematically very problematic. The hurdle stems from the fact that if we want $\epsilon(t)$ to be a random variable *at each time instant*, then the signal $\epsilon(t)$ may not actually exist, e.g., it may not even be continuous. Signals like $\epsilon(t)$ exist only in very special cases, one of them is called "Brownian motion" where the increment of the signal after infinitesimal time Δt is a Gaussian random variable

$$\epsilon(t + \Delta t) - \epsilon(t) = N(0, \Delta t).$$

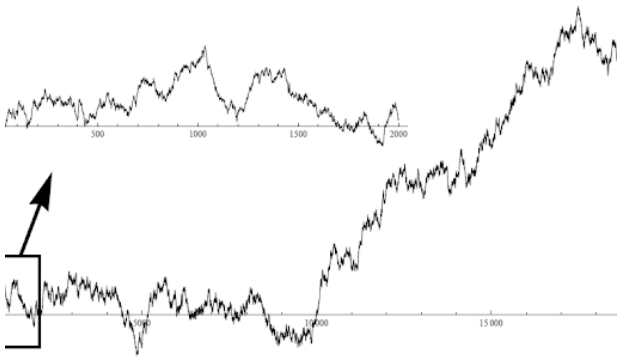


Figure 3.2: A typical Brownian motion signal $\epsilon(t)$. You can also see an animation at https://en.wikipedia.org/wiki/File:Brownian_Motion.ogv

❓ Do continuous-time systems, stochastic or non-stochastic, exist in the real world? Consider the Kuka manipulator again, do you think the dynamics of this robot is a continuous-time system? Would you model it so?

We will not worry about this technicality in this course. We will talk about continuous-time systems with noise but with the implicit understanding that there is some underlying real-world discrete-time system and the continuous-time system is only an abstraction of it.

3.4.1 Back to Hidden Markov Models

Since our sensors measure the state x of the world, it will be useful to think of the output y of a dynamical system as the observations from Chapter 2. This idea neatly ties back our development of dynamical systems to observations.

Just like we considered an HMM with observation probability

$$\mathrm{P}(Y_k = y|\ X_k = x)$$

we will consider dynamical systems for which we do not precisely know how the output computation. We will model the gap in our knowledge of the exact observation mechanism as the output being a noisy function of the state. This is denoted as

$$y_k = g(x_k) + \nu_k. \tag{3.16}$$

The noise ν_k is similar to the noise in the dynamics ϵ_k in (3.14). Analogously, we can also have noise in the observations of a linear system

$$y_k = Cx_k + Du_k + \nu_k.$$

Hidden Markov Models with underlying MDPs/Markov chains and stochastic dynamical systems with noisy observations are two different ways to think of the same concept, namely getting observations across time about the true state of a dynamic world.

In the former we have

$$\begin{aligned} \text{(state transition matrix)} \quad & \mathrm{P}(X_{k+1} = x' \mid X_k = x, u_k = u) \\ \text{(observation matrix)} \quad & \mathrm{P}(Y_k = y \mid X_k = x), \end{aligned}$$

while in the latter we have

$$\begin{aligned} \text{(nonlinear dynamics)} \quad & x_{k+1} = f(x_k, u_k) + \epsilon_k \\ \text{(nonlinear observation model)} \quad & y_k = g(x_k) + \nu_k, \end{aligned}$$

or

$$\begin{aligned} \text{(linear dynamics)} \quad & x_{k+1} = Ax_k + Bu_k + \epsilon_k \\ \text{(linear observation model)} \quad & y_k = Cx_k + Du_k + \nu_k. \end{aligned}$$

HMMs are easy to use for certain kinds of problems, e.g., speech-to-text, or a robot wandering in a grid world (like the Bayes filter problem in HW 1). Dynamical systems are more useful for certain other kinds of problems, e.g., a Kuka manipulator where you can use Newton's laws to simply write down the functions f, g.

3.5 Kalman Filter (KF)

We will now introduce the Kalman Filter. It is the analog of the Bayes filter from the previous chapter. This is by far the most important algorithm in robotics and it is hard to imagine running any robot without the Kalman fiter or some variant of it.

Consider a linear dynamical system with linear observations

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k + \epsilon_k \\ y_k &= Cx_k + \nu_k. \end{aligned} \tag{3.17}$$

ℹ Observation noise and dynamics noise are different in subtle ways. The former may not always be due to our poor modeling. For instance, the process by which a camera acquires its images has some inherent noise. You may have seen a side-by-side comparison of different cameras using their ISOs

An image taken from a camera with low lighting has a lot of "noise". What causes this noise?

ℹ You will agree that creating the state-transition matrix for the Bayes filter problem in HW 1 was really the hardest part of the problem. If the state-space were continuous and not a discrete cell-based world, you could have written the dynamics very easily in one line of code.

where the noise vectors

$$\epsilon_k \sim N(0, R)$$
$$\nu_k \sim N(0, Q)$$

are both zero-mean and Gaussian with covariances R and Q respectively. We have also assumed that $D = 0$ because typically the observations do not depend on the control.

ℹ We will assume that the distribution of noise ϵ_k, ν_k does not change with time k. If it does change in your problem, you will see that following equations are quite easy to modify.

Our goal is to compute the best estimate of the state after multiple observations

$$\mathrm{P}(x_k \mid y_1, \ldots, y_k).$$

Notice that this is the same as the filtering problem that we solved for Hidden Markov Models. Just like we used the forward algorithm to compute the filtering estimate recursively, we are going to use our development of the Kalman gain to incorporate a new observation recursively.

3.5.1 Step 0: Observing that the state estimate at any timestep is a Gaussian

Maintaining the entire probability distribution $\mathrm{P}(x_k \mid y_1, \ldots, y_k)$ is difficult now, as opposed to the HMM with a finite number of states. We will exploit the following important fact. If we assume that the initial distribution of x_0 was a Gaussian, since all operations in (3.17) are linear, our new estimate of the state $\hat{x}_k$ at time k is also a Gaussian

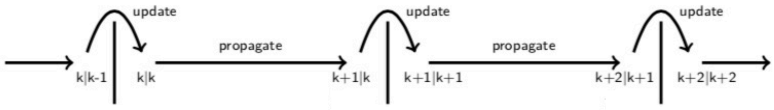
$$\hat{x}_{k|k} \sim \mathrm{P}(x_k \mid y_1, \ldots, y_k) \equiv N(\mu_{k|k}, \Sigma_{k|k}).$$

The subscript

$$\hat{x}_{k+1|k}$$

denotes that the quantity being talked about, i.e., $\hat{x}_{k+1|k}$, or others like $\mu_{k+1|k}$, is of the $(k+1)^{\text{th}}$ timestep and was calculated on the basis of observations up to (and including) the k^{th} timestep. We will therefore devise recursive updates to obtain $\mu_{k+1|k+1}, \Sigma_{k+1|k+1}$ using their old values $\mu_{k|k}, \Sigma_{k|k}$. We will imagine that our initial estimate for the state $\hat{x}_{0|0}$ has a known distribution

$$\hat{x}_{0|0} \sim N(\mu_{0|0}, \Sigma_{0|0}).$$



3.5.2 Step 1: Propagating the dynamics by one timestep

Suppose we had an estimate $\hat{x}_{k|k}$ after k observations/time-steps. Since the dynamics is linear, we can use the prediction problem to compute the estimate

of the state at time $k+1$ before the next observation arrives

$$\mathrm{P}(x_{k+1} \mid y_1, \ldots, y_k).$$

From the first equation of (3.17), this is given by

$$\hat{x}_{k+1|k} = A\hat{x}_{k|k} + Bu_k + \epsilon_{k+1}$$

Notice that the subscript on the left-hand side is $k+1|k$ because we did not take into account the observation at timestep $k+1$ yet. The mean and covariance of this estimate are given by

$$\begin{aligned} \mu_{k+1|k} = \mathrm{E}[\hat{x}_{k+1|k}] &= \mathrm{E}[A\hat{x}_{k|k} + Bu_k + \epsilon_{k+1}] \\ &= A\mu_{k|k} + Bu_k. \end{aligned} \tag{3.18}$$

We can also calculate the covariance of the estimate $\hat{x}_{k+1|k}$ to see that

$$\begin{aligned} \Sigma_{k+1|k} = \mathrm{cov}(\hat{x}_{k+1|k}) &= \mathrm{cov}(A\hat{x}_{k|k} + Bu_k + \epsilon_{k+1}) \\ &= A\Sigma_{k|k}A^\top + R, \end{aligned} \tag{3.19}$$

using our calculation in (3.1).

Observe that even if we knew the state dynamics precisely, i.e., if $R = 0$, we still have a non-trivial propagation equation for $\Sigma_{k+1|k}$.

3.5.3 Step 2: Incorporating the observation

After the dynamics propagation step, our estimate of the state is $\hat{x}_{k+1|k}$, this is the state of the system that we believe is true after k observations. We should now incorporate the latest observation y_{k+1} to update this estimate to get

$$\mathrm{P}(x_{k+1} \mid y_1, \ldots, y_k, y_{k+1}).$$

This is exactly the same problem that we saw in Section 3.2.3. Given the measurement

$$y_{k+1} = Cx_{k+1} + \nu_{k+1}$$

we first compute the Kalman gain K_{k+1} and the updated mean of the estimate as

$$\begin{aligned} K_{k+1} &= \Sigma_{k+1|k}C^\top \left(C\Sigma_{k+1|k}C^\top + Q\right)^{-1} \\ \mu_{k+1|k+1} &= \mu_{k+1|k} + K_{k+1}\left(y_{k+1} - C\mu_{k+1|k}\right). \end{aligned} \tag{3.20}$$

The covariance is given by our same calculation again

$$\begin{aligned} \Sigma_{k+1|k+1} &= \left(I - K_{k+1}C\right)\Sigma_{k+1|k}, \text{ or} \\ &= \left(I - K_{k+1}C\right)\Sigma_{k+1|k}\left(I - K_{k+1}C\right)^\top + K_{k+1}QK_{k+1}^\top, \text{ or} \\ &= \left(\Sigma_{k+1|k} + C^\top Q^{-1}C\right)^{-1}. \end{aligned} \tag{3.21}$$

The second expression is known as Joeseph's form and is numerically more stable than the other expressions.

The new estimate of the state is

$$\hat{x}_{k+1|k+1} \sim \mathrm{P}(x_{k+1} \mid y_1, \ldots, y_{k+1}) \equiv N(\mu_{k+1|k+1}, \Sigma_{k+1|k+1}).$$

and we can again proceed to Step 1 for the next timestep.

3.5.4 Discussion

There are several important observations to make and remember about the Kalman Filter (KF).

- **Recursive updates to compute the best estimate given all past observations.** The KF is a recursive filter (just like the forward algorithm for HMMs) and incorporates observations one by one. The estimate that it maintains, namely $\hat{x}_{k+1|k+1}$, depends upon all past observations

$$\hat{x}_{k+1|k+1} \sim \mathrm{P}(x_{k+1} \mid y_1, \ldots, y_{k+1}).$$

  We have simply *computed* the estimate recursively.
- **Optimality of the KF for linear systems with Gaussian noise.** The KF is optimal in the following sense. Imagine if we had access to all the observations $y_1, \ldots, y_k$ beforehand and computed some other estimate

$$\hat{x}_{k|k}^{\text{fancy filter}} = \text{some function}(\hat{x}_{0|0}, y_1, \ldots, y_k).$$

  We use some other fancy method to design this estimator, e.g., nonlinear combination of the observations or incorporating observations across multiple timesteps together etc. to obtain something that has the smallest error with respect to the true state x_k

$$\mathrm{tr}\left(\underset{\epsilon_1,\ldots,\epsilon_k,\nu_1,\ldots,\nu_k}{\mathrm{E}} \left[(\hat{x}_{k|k}^{\text{fancy filter}} - x_k)(\hat{x}_{k|k}^{\text{fancy filter}} - x_k)^\top \right] \right). \quad (3.22)$$

  Then this estimate would be exactly the same as that of the KF

$$\hat{x}_{k|k}^{\text{fancy filter}} = \hat{x}_{k|k}^{\text{KF}}.$$

  This is a deep fact. First, the KF estimate was created recursively and yet we can do no better than it with our fancy estimator. This is analogous to the fact that the forward algorithm computes the correct filtering estimate even if it incorporates observations one by one recursively. Second, the KF combines the new observation and the old estimate linearly in (3.20). You could imagine that there is some other way to incorporate new observations, but it turns out that for linear dynamical systems with Gaussian noise, the KF is the best solution, we can do no better.
- **The KF is the best linear filter.** If we had a nonlinear dynamical system or a non-Gaussian noise with a linear dynamics/observations, there are other filters that can give a smaller error (3.22) than the KF.

In the next section, we will take a look at one such example. However, even in these cases, the KF is the *best linear filter*.

- **Assumptions that are implicit in the KF**. We assumed that both the dynamics noise ϵ_{k+1} and the observation noise ν_{k+1} are uncorrelated with the estimate $\hat{x}_{k+1|k}$ computed prior to them (where did we use these assumptions?). This implicitly assumes that dynamics and observation noise are "white", i.e., uncorrelated in time

$$\mathrm{E}[\epsilon_k \ \epsilon_{k'}^\top] = 0 \quad \text{for all } k, k'$$
$$\mathrm{E}[\nu_k \ \nu_{k'}^\top] = 0 \quad \text{for all } k, k'.$$

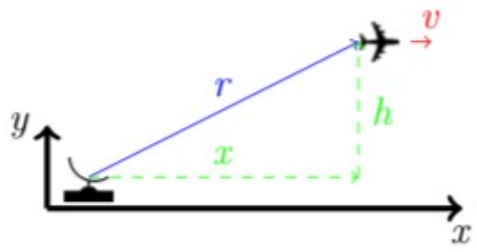
The Wikipedia webpage at https://en.wikipedia.org/wiki/Kalman_filter#Example_application,_technical gives a simple example of a Kalman Filter.

❓ How should one modify the KF equations if we have multiple sensors in a robot, each coming in at different frequencies?

3.6 Extended-Kalman Filter (EKF)

The KF heavily exploits the fact that our dynamics/measurements are linear. In robotics problems, either of them or typically both, will be nonlinear. The Extended-Kalman Filter (EKF) is a modification of the KF to handle such situations.

Example of a nonlinear dynamical system The state of most real problems evolves as a nonlinear function of their current state and control. This is a the same for sensors such as cameras measure a nonlinear function of the state. We will first see how to linearize a given nonlinear system shown below.



We have a radar sensor that measures the distance of the plane r from the radar trans-receiver up to noise ν. We would like to measure its distance x and height h. If the plane travels with a constant velocity, we have

$$\dot{x} = v, \text{ and } \dot{v} = 0,$$

and

$$r = \sqrt{x^2 + h^2} + \nu.$$

Since we do not know how the plane might change its altitude, we will assume that it maintains a constant altitude

$$\dot{h} = 0.$$

Notice that the above systems are our model for how the state of the airplane evolves and could of course be wrong. As we discussed, we will model this as noise.

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \epsilon;$$
$$r = \sqrt{x_1^2 + x_3^2} + \nu;$$

here $x_1 \equiv x$, $x_2 \equiv v$ and $x_3 = h$, and $\epsilon \in \mathbb{R}^3, \nu \in \mathbb{R}$ are zero-mean Gaussian noise. The dynamics in this case is linear but the observations are a nonlinear function of the state.

One way to use the Kalman Filter for this problem is to linearize the observation equation around some state, say $x_1 = x_2 = x_3 = 0$ using the Taylor series

$$\begin{aligned} r_{\text{linearized}} &= r(0,0,0) + \left.\frac{\partial r}{\partial x_1}\right|_{x_1=0,x_3=0} (x_1 - 0) + \left.\frac{\partial r}{\partial x_3}\right|_{x_1=0,x_3=0} (x_3 - 0) \\ &= 0 + \left.\frac{2x_1}{2\sqrt{x_1^2 + x_3^2}}\right|_{x_1=0,x_3=0} x_1 + \left.\frac{2x_3}{2\sqrt{x_1^2 + x_3^2}}\right|_{x_1=0,x_3=0} x_3 \\ &= x_1 + x_3. \end{aligned}$$

In other words, upto first order in x_1, x_3, the observations are linear and we can therefore run the KF for computing the state estimate after k observations.

You can try to perform a similar linearization for a simple model of a car

$$\dot{x} = \cos\theta$$
$$\dot{y} = \sin\theta$$
$$\dot{\theta} = u.$$

where x, y, θ are the XY-coordinates and the angle of the steering wheel respectively. This model is known as a Dubins car.

3.6.1 Propagation of statistics through a nonlinear transformation

Given a Gaussian random variable $\mathbb{R}^d \ni x \sim N(\mu_x, \Sigma_x)$, we saw how to compute the mean and covariance after an *affine* transformation $y = Ax$

$$\mathrm{E}[y] = A\,\mathrm{E}[x], \text{ and } \Sigma_y = A\Sigma_x A^\top.$$

If we had a nonlinear function of x

$$\mathbb{R}^p \ni y = f(x)$$

we can use the Taylor series by linearizing around the mean of x to approximate the first and second moments of y as follows.

$$\begin{aligned} y = f(x) &\approx f(\mu_x) + \left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x=\mu_x} (x - \mu_x) \\ &= Jx + (f(\mu_x) - J\mu_x). \end{aligned}$$

where we have defined the Jacobian matrix

$$\mathbb{R}^{p\times d} \ni J = \left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x=\mu_x}. \tag{3.23}$$

This gives

$$\begin{aligned} \mathrm{E}[y] &\approx \mathrm{E}[Jx + (f(\mu_x) - J\mu_x)] = f(\mu_x) \\ \Sigma_y &= \mathrm{E}[(y - \mathrm{E}[y])(y - \mathrm{E}[y])^\top] \approx J\Sigma_x J^\top. \end{aligned} \tag{3.24}$$

Observe how, up to first order, the mean μ_x is directly transformed by the nonlinear function f while the covariance Σ_x is transformed as if there were a linear operation $y \approx Jx$.

A simple example

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = f\left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) = \begin{bmatrix} x_1^2 + x_2 x_3 \\ \sin x_2 + \cos x_3 \end{bmatrix}.$$

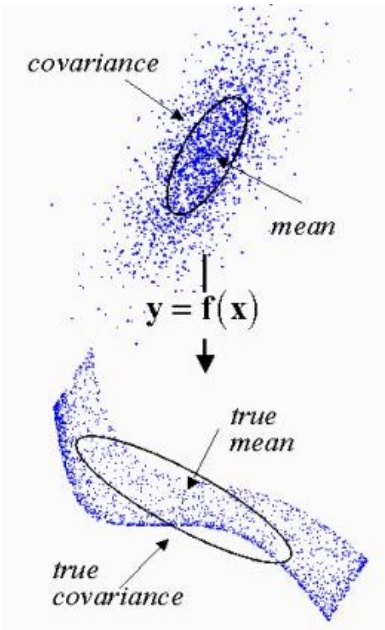
We have

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \nabla f(x) = \begin{bmatrix} 2x_1 & x_3 & x_2 \\ 0 & \cos x_2 & -\sin x_3 \end{bmatrix}.$$

The Jacobian at $\mu_x = [\mu_{x_1}, \mu_{x_2}, \mu_{x_3}]^\top$ is

$$J = \nabla f(x)\Big|_{x=\mu_x} = \begin{bmatrix} 2\mu_{x_1} & \mu_{x_3} & \mu_{x_2} \\ 0 & \cos \mu_{x_2} & -\sin \mu_{x_3} \end{bmatrix}.$$

It is very important to remember that we are approximating the distribution of $\mathrm{P}(f(x))$ as a Gaussian. Even if x is a Gaussian random variable, the distribution of $y = f(x)$ need not be Gaussian. Indeed y is only Gaussian if f is an affine function of x.



3.6.2 Extended Kalman Filter

The above approach of linearizing the observations of the plane around the origin may lead to a lot of errors. This is because the point about which we linearize the system is fixed. We can do better by linearizing the system at each timestep. Let us say that we are given a nonlinear system

Can you say where will our linearized observation equation incur most error?

$$\begin{aligned} x_{k+1} &= f(x_k, u_k) + \epsilon \\ y_k &= g(x_k) + \nu. \end{aligned}$$

> The central idea of the Extended Kalman Filter (EKF) is to linearize a nonlinear system at each timestep k around the latest state estimate given by the Kalman Filter and use the resultant linearized dynamical system in the KF equations for the next timestep.

Step 1: Propagating the dynamics by one timestep

We will linearize the dynamics equation around the mean of the previous state estimate $\mu_{k|k}$

$$\begin{aligned} x_{k+1} &= f(x_k, u_k) + \epsilon \\ &\approx f(\mu_{k|k}, u_k) + \left.\frac{\partial f}{\partial x}\right|_{x=\mu_{k|k}} (x_k - \mu_{k|k}) + \epsilon_k. \end{aligned}$$

Let the Jacobian be

$$A(\mu_{k|k}) = \left.\frac{\partial f}{\partial x}\right|_{x=\mu_{k|k}}. \tag{3.25}$$

The mean and covariance of the EKF after the dynamics propagation step is therefore given by

$$\begin{aligned} \mu_{k+1|k} &= f(\mu_{k|k}, u_k) \\ \Sigma_{k+1|k} &= A\Sigma_{k|k}A^\top + R. \end{aligned} \tag{3.26}$$

It is worthwhile to notice the similarities of the above set of equations with (3.18) and (3.19). The mean $\mu_{k|k}$ is propagated using a nonlinear function f to get $\mu_{k+1|k}$, the covariance is propagated using the Jacobian $A(\mu_{k|k})$ which is recomputed using (3.25) at each timestep.

Step 2: Incorporating the observation

We have access to $\mu_{k+1|k}$ after Step 1, so we can linearize the nonlinear observations at this state.

$$\begin{aligned} y_{k+1} &= g(x_{k+1}) + \nu \\ &\approx g(\mu_{k+1|k}) + \left.\frac{\mathrm{d}g}{\mathrm{d}x}\right|_{x=\mu_{k+1|k}} (x_{k+1} - \mu_{k+1|k}) + \nu \end{aligned}$$

Again define the Jacobian

$$C(\mu_{k+1|k}) = \frac{\partial g}{\partial x}\bigg|_{x=\mu_{k+1|k}}. \tag{3.27}$$

Consider the fake observation which is a transformed version of the actual observation y_{k+1} (think of this as a new sensor or a post-processed version of the original sensor)

$$y'_{k+1} = y_{k+1} - g(\mu_{k+1|k}) + C\mu_{k+1|k} \approx Cx_{k+1}.$$

Our fake observation is a nice linear function of the state x_{k+1} and we can therefore use the Kalman Filter equations to incorporate this fake observation

$$\begin{aligned}\mu_{k+1|k+1} &= \mu_{k+1|k} + K(y'_{k+1} - C\,\mu_{k+1|k}) \\ \text{where } K &= \Sigma_{k+1|k}C^\top \left(C\Sigma_{k+1|k}C^\top + Q\right)^{-1}.\end{aligned}$$

Let us resubstitute our fake observation in terms of the actual observation y_{k+1}.

$$y'_{k+1} - C\,\mu_{k+1|k} = y_{k+1} - g(\mu_{k+1|k}),$$

to get the EKF equations for incorporating one observation

$$\begin{aligned}\mu_{k+1|k+1} &= \mu_{k+1|k} + K(y_{k+1} - g(\mu_{k+1|k})) \\ \Sigma_{k+1|k+1} &= (I - KC)\,\Sigma_{k+1|k}.\end{aligned} \tag{3.28}$$

Summary of the Extended Kalman Filter The EKF can use a nonlinear model of the system but linearizes the dynamics and observation equations at each timestep before plugging them into the Kalman Filter formulae.

1. Just like the KF, say we have the current state estimate $\mu_{k|k}$ and $\Sigma_{k|k}$.
2. The system applies a control input u_k and we update our state estimate to be
$$\begin{aligned}\mu_{k+1|k} &= f(\mu_{k|k}, u_k) \\ \Sigma_{k+1|k} &= A\Sigma_{k|k}A^\top + R.\end{aligned}$$
where A depends on $\mu_{k|k}$.
3. We next incorporate an observation by linearizing the observation equations around $\mu_{k+1|k}$
$$\begin{aligned}K &= \Sigma_{k+1|k}C^\top \left(C\Sigma_{k+1|k}C^\top + Q\right)^{-1} \\ \mu_{k+1|k+1} &= \mu_{k+1|k} + K(y_{k+1} - g(\mu_{k+1|k})) \\ \Sigma_{k+1|k+1} &= (I - KC)\,\Sigma_{k+1|k}\end{aligned}$$
where again C depends on $\mu_{k+1|k}$

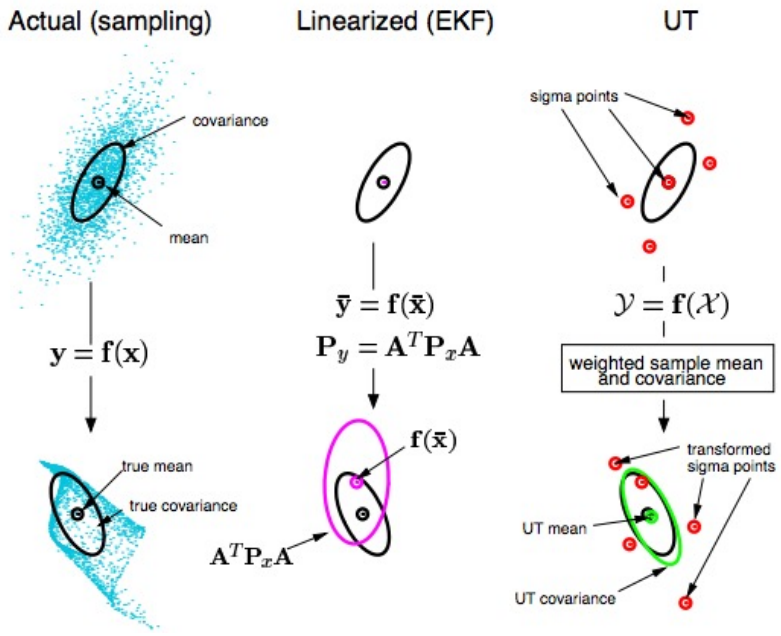
Discussion

1. The EKF dramatically expands the applicability of the Kalman Filter. It can be used for most real systems, even with very complex models f, h. It is very commonly used in robotics and can handle nonlinear observations from complex sensors such as a LiDAR and camera easily. For instance, sophisticated augmented/virtual reality systems like Google ARCore/Snapchat/iPhone etc. (https://www.youtube.com/watch?v=cape_Af9j7w) run EKF to track the motion of the phone or of the objects in the image.
2. The KF was special because it is the optimal linear filter, i.e., KF estimates have the smallest mean squared error with respect to the true state for linear dynamical systems with Gaussian. The EKF is a clever application of KF to nonlinear systems but it no longer has this property. There do exist filters for nonlinear systems that will have a smaller mean-squared error than the EKF. We will look at some of them in the next section.
3. Linearization is the critical step in the implementation of the EKF and EKF state estimate can be easily quite bad, especially if the system goes into states where the linearized matrix A and the nonlinear dynamics $f(x_k, u_k)$ differ significantly. A common trick for handling this is to perform multiple steps of dynamics propagation using a continuous-time model of the system between successive observations. Say we have a system
$$\dot{x} = f(x(t), u(t)) + \epsilon(t)$$
where $\epsilon(t + \delta t) - \epsilon(t)$ is a Gaussian random variable $N(0, R\delta t)$ as $\delta \to 0$; see the section on Brownian motion for how to interpret noise in continuous-time systems. We can construct a discrete-time system from this as
$$\begin{aligned} x_{t+\Delta t} &= x(t) + f(x(t), u(t))\, \Delta t + \epsilon \\ &\equiv f^{\text{discrete-time}}(x(t), u(t)) + \epsilon. \end{aligned}$$
where $\epsilon \sim N(0, R\Delta t)$ is noise. This is now a discrete-time dynamics and we can perform Step 1 of the EKF multiple times to obtain a more accurate estimate of $\mu_{k+1|k}$ and $\Sigma_{k+1|k}$.

3.7 Unscented Kalman Filter (UKF)

Linearization of the dynamics in the EKF is a neat trick to use the KF equations. But as we said, this can cause severe issues in problems where the dynamics is very nonlinear. In this section, we will take a look at a powerful method to handle nonlinear dynamics that is better than linearization.

Let us focus on Step 1 which propagates the dynamics in the EKF.



We know that even if x is Gaussian (faint blue points in top left picture), the transformed variable $y = f(x)$ need not be Gaussian (faint blue points in bottom left). The EKF is really approximating the probability distribution $\mathrm{P}(x_{k+1} \mid y_1, \ldots, y_k)$ as a Gaussian; this distribution could be very different from a Gaussian. This is really the crux of the issue in filtering for nonlinear systems. This approximation, which happens because we are linearizing about the mean $\mu_{k|k}$.

Let us instead do the following:

1. Sample a few points from the Gaussian $N(\mu_{k|k}, \Sigma_{k|k})$ (red points in top right).
2. Transform *each of the points* using the nonlinear dynamics f (red points in bottom right).
3. Compute their mean and covariance to get $\mu_{k+1|k}$ and $\Sigma_{k+1|k}$. Notice how the green ellipse is slightly different than the black ellipse (which is the true mean and covariance). Both of these would be different from the mean and covariance obtained by linearization of f (middle column) but the green one is more accurate.

In general, we would need a large number of sample points (red) to

accurately get the mean and covariance of $y = f(x)$. The Unscented Transform (UT) uses a special set of points known as "sigma points" (these are the ones actually shown in red above) and transforms those points. Sigma points have the special property that the empirical mean of the transformed distribution (UT mean in the above picture) is close to the true mean up to third order; linearization is only accurate up to first order. The covariance (UT covariance) and true covariance also match up to third order.

3.7.1 Unscented Transform

Given a random variable $x \sim N(\mu_x, \Sigma_x)$, the Unscented Transform (UT) uses sigma points to compute an approximation of the probability distribution of the random variable $y = f(x)$.

Preliminaries: matrix square root. Given a symmetric matrix $\Sigma \in \mathbb{R}^{n \times n}$, the matrix square root of Σ is a matrix $S \in \mathbb{R}^{n \times n}$ such that

$$\Sigma = SS.$$

We can compute this via diagonalization as follows.

$$\begin{aligned}
\Sigma &= VDV^{-1} \\
&= V \begin{bmatrix} d_{11} & \cdots & 0 \\ 0 & & \\ \cdots & 0 & \\ 0 & \cdots & d_{nn} \end{bmatrix} V^{-1} \\
&= V \begin{bmatrix} \sqrt{d_{11}} & \cdots & 0 \\ 0 & & \\ \cdots & 0 & \\ 0 & \cdots & \sqrt{d_{nn}} \end{bmatrix}^2 V^{-1}.
\end{aligned}$$

We can therefore define

$$S = V \begin{bmatrix} \sqrt{d_{11}} & \cdots & 0 \\ 0 & & \\ \cdots & 0 & \\ 0 & \cdots & \sqrt{d_{nn}} \end{bmatrix} V^{-1}.$$

Notice that

$$SS = \left(VD^{1/2}V^{-1}\right)\left(VD^{1/2}V^{-1}\right) = VDV^{-1} = \Sigma.$$

We can also define the matrix square root using the Cholesky decomposition $\Sigma = LL^\top$ which is numerically more stable than computing the square root using the above expression. Recall that matrices L and Σ have the same eigenvectors. Typical applications of the Unscented Transform will use this method.

Given a random variable $\mathbb{R}^n \ni x \sim N(\mu, \Sigma)$, we will use the matrix square root to compute the sigma points as

$$\begin{aligned} x^{(i)} &= \mu + \sqrt{n\Sigma}_i^\top \\ x^{(n+i)} &= \mu - \sqrt{n\Sigma}_i^\top \\ &\text{for } i = 1, \dots, n, \end{aligned} \tag{3.29}$$

where $\sqrt{n\Sigma}_i$ is the i^{th} row of the matrix $\sqrt{n\Sigma}$. There are $2n$ sigma points

$$\left\{x^{(1)}, \dots, x^{(2n)}\right\}$$

for an n-dimensional Gaussian. Each sigma point is assigned a weight

$$w^{(i)} = \frac{1}{2n}. \tag{3.30}$$

We then transform each sigma point to get the transformed sigma points

$$y^{(i)} = f(x^{(i)}).$$

The mean and covariance of the transformed random variable y can now be computed as

$$\begin{aligned} \mu_y &= \sum_{i=1}^{2n} w^{(i)} y^{(i)} \\ \Sigma_y &= \sum_{i=1}^{2n} w^{(i)} \left(y^{(i)} - \mu_y\right) \left(y^{(i)} - \mu_y\right)^\top. \end{aligned} \tag{3.31}$$

Example Say we have $x = \begin{bmatrix} r \\ \theta \end{bmatrix}$ with $\mu_x = [1, \pi/2]$ and $\Sigma_x = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\theta^2 \end{bmatrix}$. We would like to compute the probability distribution of $y = f(x) = \begin{bmatrix} r\cos\theta \\ r\sin\theta \end{bmatrix}$ which is a polar transformation. Since x is two-dimensional, we will have 4 sigma points with equal weights $w^{(i)} = 0.25$. The square root in the sigma point expression is

$$\sqrt{n\Sigma} = \begin{bmatrix} \sqrt{2}\sigma_r & 0 \\ 0 & \sqrt{2}\sigma_\theta \end{bmatrix}$$

and the sigma points are

$$\begin{aligned} x^{(1)} &= \begin{bmatrix} 1 \\ \pi/2 \end{bmatrix} + \begin{bmatrix} \sqrt{2}\sigma_r \\ 0 \end{bmatrix}, \quad x^{(3)} = \begin{bmatrix} 1 \\ \pi/2 \end{bmatrix} - \begin{bmatrix} \sqrt{2}\sigma_r \\ 0 \end{bmatrix} \\ x^{(2)} &= \begin{bmatrix} 1 \\ \pi/2 \end{bmatrix} + \begin{bmatrix} 0 \\ \sqrt{2}\sigma_\theta \end{bmatrix}, \quad x^{(4)} = \begin{bmatrix} 1 \\ \pi/2 \end{bmatrix} - \begin{bmatrix} 0 \\ \sqrt{2}\sigma_\theta \end{bmatrix}. \end{aligned}$$

❓ Compute the mean and covariance of y by linearizing the function $f(x)$.

The transformed sigma points are

$$
\begin{aligned}
y^{(1)} &= \begin{bmatrix} r^{(1)}\cos\theta^{(1)} \\ r^{(1)}\sin\theta^{(1)} \end{bmatrix} = \begin{bmatrix} 0 \\ 1+\sqrt{2}\sigma_r \end{bmatrix} \\
y^{(2)} &= \begin{bmatrix} r^{(2)}\cos\theta^{(2)} \\ r^{(2)}\sin\theta^{(2)} \end{bmatrix} = \begin{bmatrix} \cos(\pi/2+\sqrt{2}\sigma_\theta) \\ \sin(\pi/2+\sqrt{2}\sigma_\theta) \end{bmatrix} \\
y^{(3)} &= \begin{bmatrix} 0 \\ 1-\sqrt{2}\sigma_\theta \end{bmatrix} \\
y^{(4)} &= \begin{bmatrix} \cos(\pi/2-\sqrt{2}\sigma_\theta) \\ \sin(\pi/2-\sqrt{2}\sigma_\theta) \end{bmatrix}.
\end{aligned}
$$

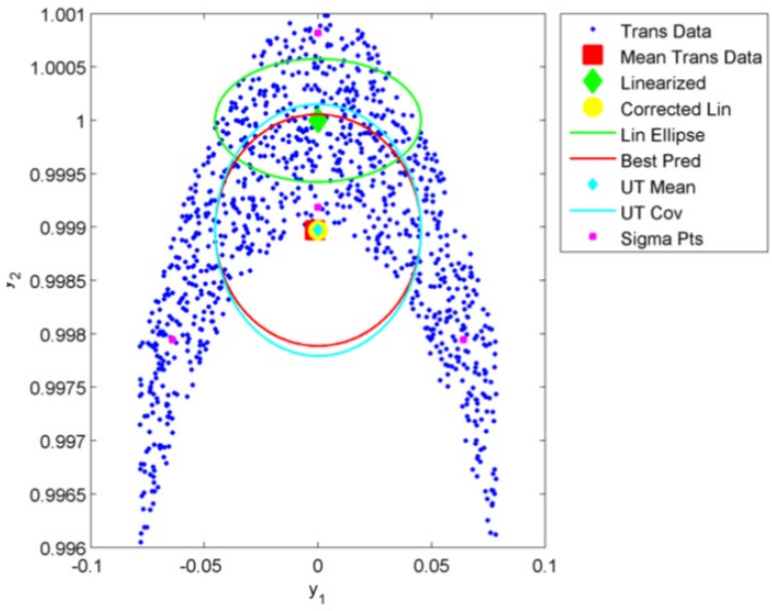


Figure 3.3: Note that the true mean is being predicted very well by the UT and is clearly a better estimate than the linearized mean.

3.7.2 The UT with tuning parameters

The UT is a basic template for a large suite of techniques that capture the covariance Σ_x as a set of points and transform those points through the nonlinearity. You will see many alternative implementations of the UT that allow for user-tunable parameters. For instance, sometimes the UT is implemented with an additional sigma point $x^{(0)} = \mu$ with weight $w^{(0)} = \frac{\lambda}{n+\lambda}$ and the weights of the other points are adjusted to be $w^{(i)} = \frac{1}{2(n+\lambda)}$ for a user-chosen parameter λ. You may also see people using one set of weights $w^{(i)}$ for computing the mean μ_y and and another set of weights for computing the covariance Σ_y.

❓ Are the transformed sigma points $y^{(i)}$ the sigma points of $\mathrm{P}(y) = N(\mu_y, \Sigma_y)$?

❓ We are left with a big lingering question. Why do you think this method is called the "unscented transform"?

3.7.3 Unscented Kalman Filter (UKF)

The Unscented Transform gives us a way to accurately estimate the mean and covariance of the transformed distribution through a nonlinearity. We can use the UT to modify the EKF to make it a more accurate state estimator. The resultant algorithm is called the Unscented Kalman Filter (UKF).

Step 1: Propagating the dynamics by one timestep Given our current state estimate $\mu_{k|k}$ and $\Sigma_{k|k}$, we use the UT to obtain the updated estimates $\mu_{k+1|k}$ and $\Sigma_{k+1|k}$. If $x^{(i)}$ are the sigma points with corresponding weights $w^{(i)}$ for the Gaussian $N(\mu_{k|k}, \Sigma_{k|k})$, we set

$$\begin{aligned} \mu_{k+1|k} &:= \sum_{i=1}^{2n} w^{(i)} f(x^{(i)}, u_k) \\ \Sigma_{k+1|k} &:= R + \sum_{i=1}^{2n} w^{(i)} \left(f(x^{(i)}) - \mu_{k+1|k} \right) \left(f(x^{(i)}) - \mu_{k+1|k} \right)^\top \end{aligned} \tag{3.32}$$

Step 2.1: Incorporating one observation The observation step is also modified using the UT. The key issue in this case is that we need a way to compute the Kalman gain in terms of the sigma points in the UT. We proceed as follows.

Using *new* sigma points $x^{(i)}$ for the updated state distribution $N(\mu_{k+1|k}, \Sigma_{k+1|k})$ with equal weights $w^{(i)} = 1/2n$, we first compute their mean after the transformation

$$\hat{y} = \sum_{i=1}^{2n} w^{(i)} g(x^{(i)}) \tag{3.33}$$

and covariances

$$\begin{aligned} \Sigma_{yy} &:= Q + \sum_{i=1}^{2n} w^{(i)} \left(g(x^{(i)}) - \hat{y} \right) \left(g(x^{(i)}) - \hat{y} \right)^\top \\ \Sigma_{xy} &:= \sum_{i=1}^{2n} w^{(i)} \left(x^{(i)} - \mu_{k+1|k} \right) \left(g(x^{(i)}) - \hat{y} \right)^\top . \end{aligned} \tag{3.34}$$

Step 2.2: Computing the Kalman gain Until now we have written the Kalman gain using the measurement matrix C. We will now discuss a more abstract formulation that gives the same expression.

Say we have a random variable x with known μ_x, Σ_x and get a new observation y. We saw how to incorporate this new observation to obtain a better estimator for x in Section 3.2.3. We will go through a similar analysis as before but in a slightly different fashion, one that does not involve the matrix C. Let

$$p = \begin{bmatrix} x \\ y \end{bmatrix}$$

and $\mu_p = \begin{bmatrix} \mu_x & \mu_y \end{bmatrix}$ and

$$\Sigma_p = \begin{bmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_{yy} \end{bmatrix}.$$

Finding the best (minimum variance estimator) $\hat{x} = \mu_x + K(y - \mu_y)$ amounts to minimizing

$$\min_K \mathrm{E}\left[\mathrm{tr}\left(\hat{x} - x\right)\left(\hat{x} - x\right)^\top\right].$$

This is called the least squares problem, which you have seen before perhaps in slightly different notation. You can solve this problem to see that the best gain K is given by

$$K^* = \Sigma_{xy}\Sigma_{yy}^{-1}. \tag{3.35}$$

and this gain leads to an error of

$$\left(\hat{x} - x\right)\left(\hat{x} - x\right)^\top = \Sigma_{xx} - \Sigma_{xy}\Sigma_{yy}^{-1}\Sigma_{yx} = \Sigma_{xx} - K^* Q_{yy} {K^*}^\top.$$

The nice thing about the Kalman gain in (3.35) is that we can compute it now using expressions of Σ_{xy} and Σ_{yy} in terms of the sigma points. This goes as as follows:

$$\begin{aligned} K^* &= \Sigma_{xy}\Sigma_{yy}^{-1} \\ \mu_{k+1|k+1} &= \mu_{k+1|k} + K\left(y_{k+1} - \hat{y}\right) \\ \Sigma_{k+1|k+1} &= \Sigma_{k+1|k} - \Sigma_{xy}\Sigma_{yy}^{-1}\Sigma_{yx} \\ &= \Sigma_{k+1|k} - K^*\Sigma_{yy}{K^*}^\top. \end{aligned} \tag{3.36}$$

Summary of UKF

1. The Unscented Transform (UT) is an alternative to linearization. It gives a better approximation of the mean and covariance of the random variable after being transformed using a nonlinear function than taking the Taylor series approximation.
2. The UKF uses the UT and its sigma points for propagation of uncertainty through the dynamics (3.32) and observation nonlinearities (3.36).

3.7.4 UKF vs. EKF

As compared to the Extended Kalman Filter, the UKF is a better approximation for nonlinear systems. Of course, if the system is linear, both EKF and UKF are equivalent to a Kalman Filter.

In practice, we typically use the UKF with some tuning parameters in the Unscented Transform as discussed in Section 3.7.2. In practice, The EKF also has tuning parameters where we may wish to perform multiple updates of the dynamics equations with a smaller time-discretization before the next observation comes in to alleviate the effect of linearizing the dynamics. A well-tuned EKF is often only marginally worse than an UKF: the former requires us to compute Jacobians at each step which the latter does not, but the latter is often a more involved implementation.

UKF/EKF approximate filtering distribution as a Gaussian An important point to remember about both the UKF and EKF is that even if they can handle nonlinear systems, they still approximate the filtering distribution

$$\mathrm{P}(x_k \mid y_1, \ldots, y_k)$$

as a Gaussian.

3.8 Particle Filters (PFs)

We next look at particle filters (PFs) which are a generalization of the UKF and can handle non-Gaussian filtering distributions. Just like the UT forms the building block of the UKF, the building block of a particle filter is the idea of importance sampling.

3.8.1 Importance sampling

Consider the following problem, given a probability distribution $p(x)$, we want to approximate it as a sum of Dirac-delta distributions at points $x^{(i)}$, also called "particles", each with weight $w^{(i)}$

$$p(x) \approx \sum_{i=1}^{n} w^{(i)} \delta_{x^{(i)}}(x).$$

Say all weights are equal $1/n$. Depending upon how we pick the samples $x^{(i)}$, we can get very different approximations

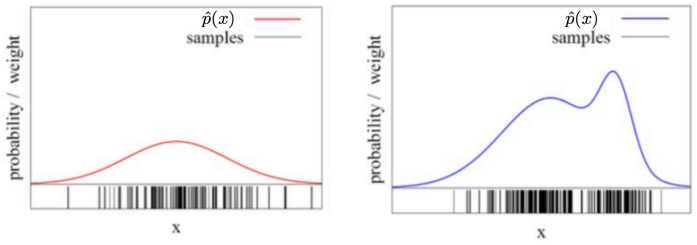


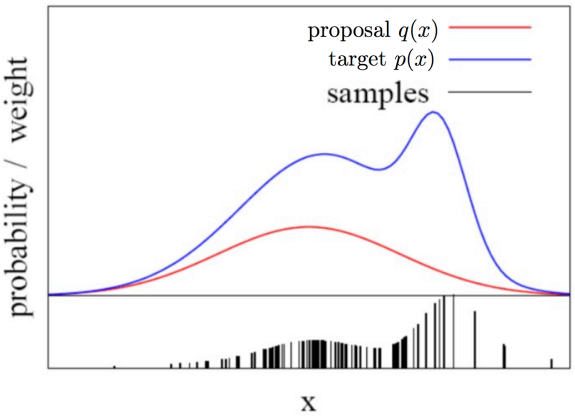
Figure 3.4: Black lines denote particles $x^{(i)}$, while red and blue curves denote the approximations obtained using them. If there are a large number of particles in a given region, the approximated probability density of that region is higher.

We see in Figure 3.4 that depending upon the samples, the approxi-

mated probability distributions $\hat{p}(x)$ can be quite different. Importance sampling is a technique to sample the particles to approximate a given probability distribution $p(x)$. The main idea is to use another *known* probability distribution, let us call it $q(x)$ to *generate particles* $x^{(i)}$ and account for the differences between the two by assigning weights to each particle

$$\text{For } i = 1, \ldots, n,$$
$$x^{(i)} \sim q$$
$$w^{(i)} = \frac{p(x^{(i)})}{q(x^{(i)})}.$$

The original distribution $p(x)$ is called the "target" and our chosen distribution $q(x)$ is called the "proposal". If the number of particles n is large, we can expect a better approximation of the target density $p(x)$.



3.8.2 Resampling particles to make the weights equal

A particle filter modifies the weights of each particle as it goes through the dynamics and observation update steps. This often causes some particles to have very low weights and some others to have very high weights.

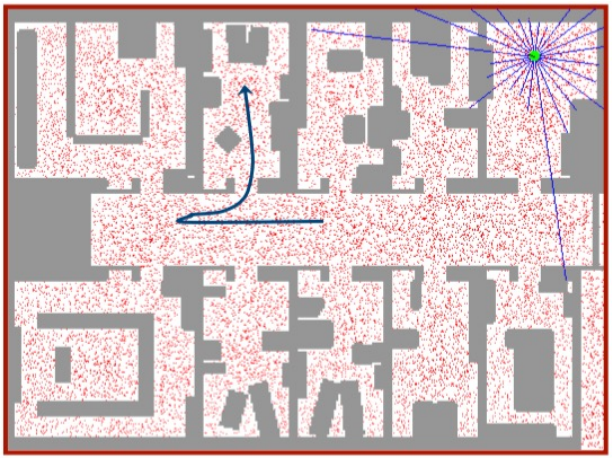


Figure 3.5: An example run of a particle filter. The robot is shown by the green dot in the top right. Observations from a laser sensor (blue rays) attached to the robot measure its distance in a 360-degree field of view around it. Red dots are particles, i.e., possible locations of the robot that we need in order to compute the filtering density $\mathrm{P}(x_k \mid y_1, \ldots, y_k)$. You should think of this picture as being similar to Problem 1 in Homework 1 where the robot was traveling on a grid. Just like the the filtering density in Problem 1 was essentially zero in some parts of the domain, the particles, say in the bottom left, will have essentially zero weights in a particle filter once we incorporate multiple observations from the robot in top right. Instead of having to carry around these null particles with small weights, the resampling step is used to remove them and sample more particles, say in the top right, where we can benefit from a more accurate approximation of the filtering density.

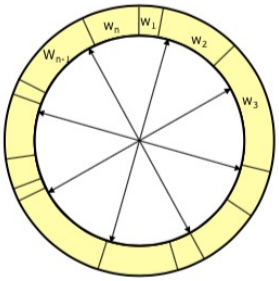
The resampling step takes particles $\left\{w^{(i)}, x^{(i)}\right\}_{i=1}^{n}$ which approximate a probability density $p(x)$

$$p(x) = \sum_{i=1}^{n} w^{(i)} \delta_{x^{(i)}}(x)$$

and returns a new set of particles $x'^{(i)}$ with equal weights $w'^{(i)} = 1/n$ that approximate the same probability density

$$p(x) = \frac{1}{n} \sum_{i=1}^{n} \delta_{x'^{(i)}}(x).$$

The goal of the resampling step is to avoid particle degeneracy, i.e., remove unlikely particles with very low weights and effectively split the particles with very large weights into multiple particles.



Consider the weights of particles $\left\{w^{(i)}\right\}$ arranged in a roulette wheel as shown above. We perform the following procedure: we start at some location, say $\theta = 0$, and move along the wheel in random increments of the angle. After each random increment, we add the corresponding particle into our set $\left\{x'^{(i)}\right\}$. Since particles with higher weights take up a larger angle in the circle, this procedure will often pick those particles and quickly move across particles with small weights without picking them too often. We perform this procedure n times for n particles. As an algorithm

1. Let r be a uniform random variable in interval $[0, 1/n]$. Pick $c = w^{(1)}$ and initialize $i = 1$.
2. For each $m = 1, \ldots, n$, let $u = r+(m-1)/n$. Increment $i \leftarrow i+1$ and $c \leftarrow c + w^{(i)}$ while $u > c$ and set new particle location $x'^{(m)} = x^{(i)}$.

It is important to notice that the resampling procedure does not actually change the locations of particles. Particles with weights much lower than $1/n$ will be eliminated while particles with weights much higher than $1/n$ will be "cloned" into multiple particles each of weight $1/n$.

There are many other methods of resampling. We have discussed here, something known as "low variance resampling", which is easy to remember and code up. Fancier resampling methods also change the locations of the particles. The goal remains the same, namely to eliminate particles with low weights.

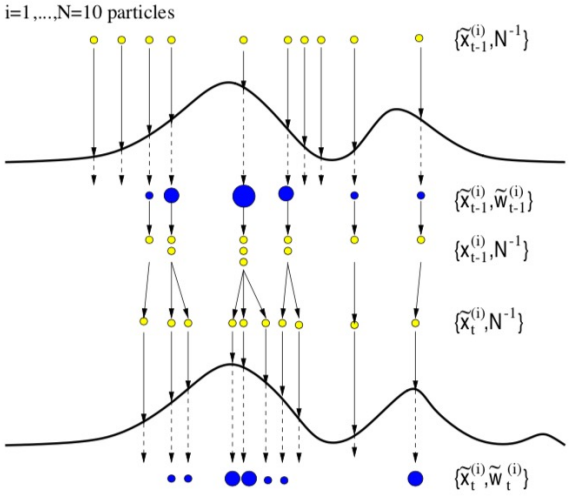


Figure 3.6: A cartoon depicting resampling. Disregard the different notation in this cartoon. Resampling does not change the probability distribution that we wish to approximate; it simply changes the particles and their weights.

3.8.3 Particle filtering: the algorithm

The basic template of a PF is similar to that of the UKF and involves two steps, the first where we propagate particles using the dynamics to estimate $\mathrm{P}(x_{k+1} \mid y_1, \ldots, y_k)$ and a second step where we incorporate the observation to compute the updated distribution $\mathrm{P}(x_{k+1} \mid y_1, \ldots, y_{k+1})$.

Before we look at the theoretical derivation of a particle filter, it will help to go through the algorithm as you would implement on a computer. We assume that we have access to particles $x^{(i)}_{k|k}$

$$\mathrm{P}(x_k \mid y_1, \ldots, y_k) = \frac{1}{n} \sum_{i=1}^{n} \delta_{x^{(i)}_{k|k}}(x),$$

all with equal weights $w^{(i)}_{k|k} = 1/n$.

1. **Step 1: Propagating the dynamics**. Each particle $i = 1, \ldots, n$ is updated by one timestep

   $$x^{(i)}_{k+1|k} = f(x^{(i)}_{k|k}, u_k) + \epsilon_k$$

   where f is the system dynamics using Gaussian noise $\epsilon_k \sim N(0, R)$. Weights of particles are unchanged $w^{(i)}_{k+1|k} = w^{(i)}_{k|k} = 1/n$.
2. **Step 2: Incorporating the observation**. Given a new observation y_{k+1}, we update the weight of each particle using the likelihood of receiving that observation

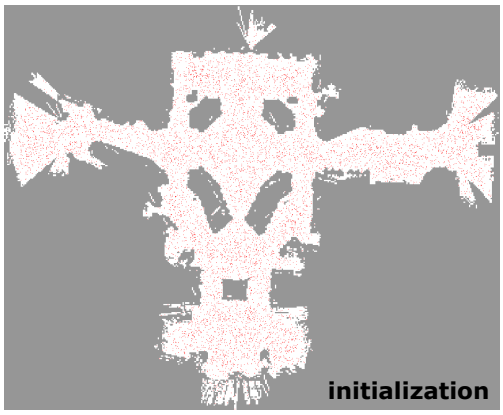
   $$w^{(i)}_{k+1|k+1} \propto \mathrm{P}(y_{k+1} \mid x^{(i)}_{k+1|k})\, w^{(i)}_{k+1|k}.$$

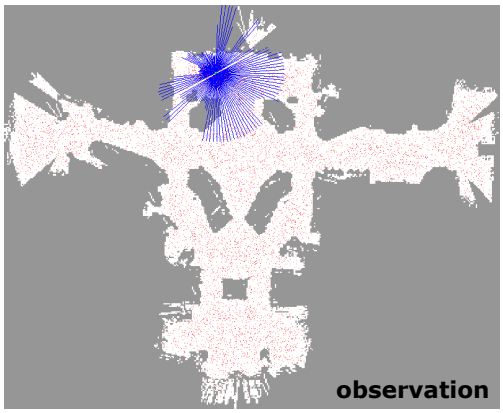
   Note that $\mathrm{P}(y_{k+1} \mid x^{(i)}_{k+1|k})$ is a Gaussian and depends upon the Gaussian observation noise ν_k. The mean of this Gaussian is $g(x^{(i)}_{k+1|k})$ and its variance is equal to Q, i.e.,

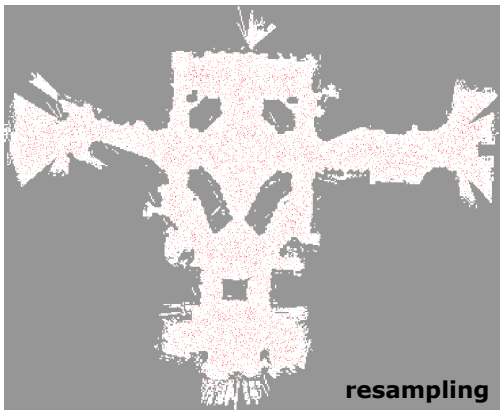
   $$\begin{aligned}\mathrm{P}(y_{k+1} \mid x^{(i)}_{k+1|k}) &= \mathrm{P}(\nu_{k+1} \equiv y_{k+1} - g(x^{(i)}_{k+1|k})) \\ &= \frac{1}{\sqrt{(2\pi)^p \det(Q)}} \exp\left(-\frac{\nu^\top_{k+1} Q^{-1} \nu_{k+1}}{2}\right).\end{aligned}$$

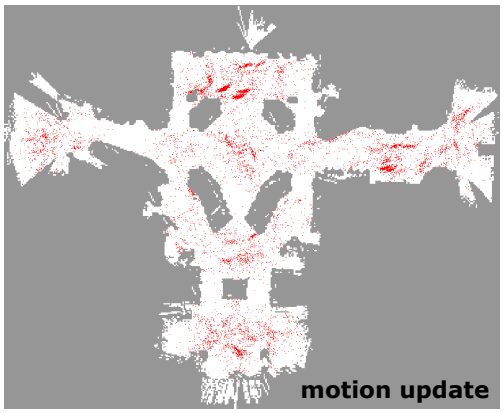
   Normalize the weights $w^{(i)}_{k+1|k+1}$ to sum up to 1.
3. **Step 3: Resampling step** Perform the resampling step to obtain new particle locations $x^{(i)}_{k+1|k+1}$ with uniform weights $w^{(i)}_{k+1|k+1} = 1/n$.

3.8.4 Example: Localization using particle filter



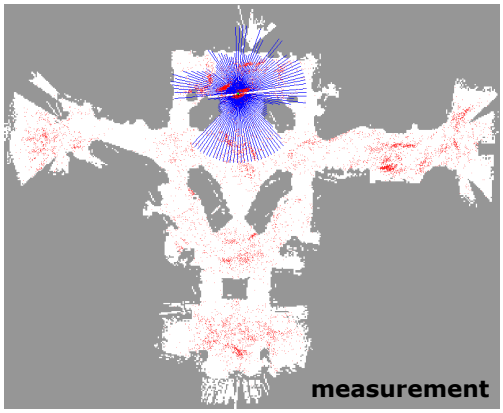






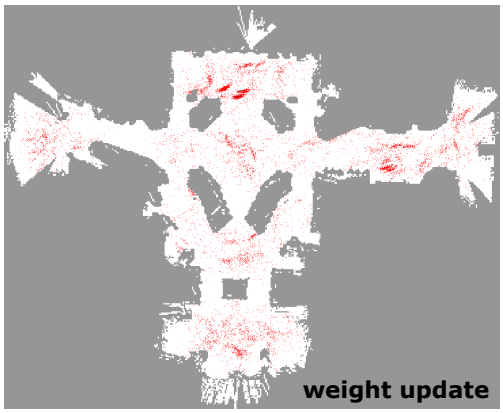
21

1



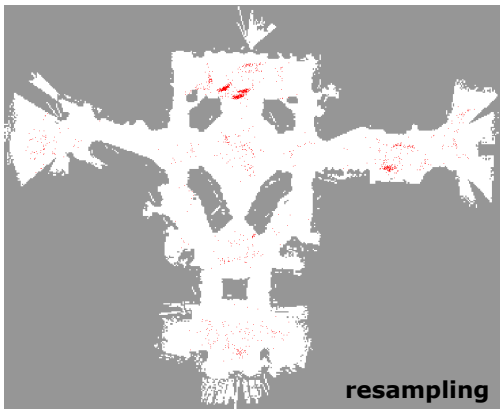
22

2



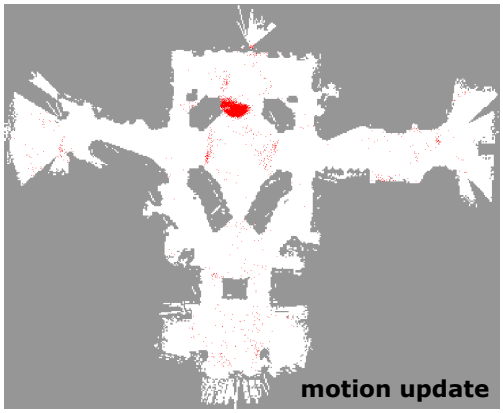
23

3



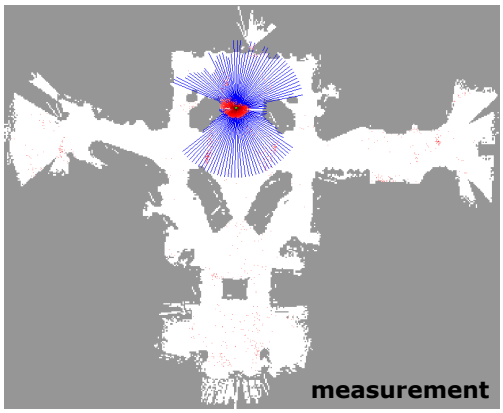
24

1



25

2

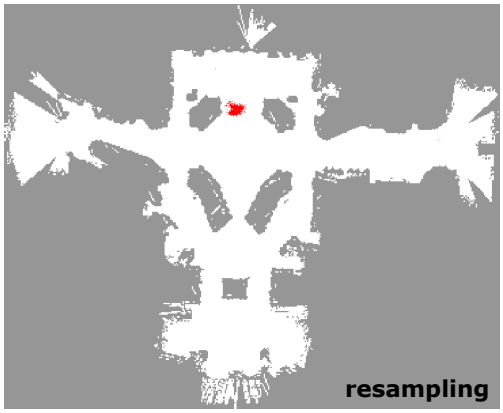


26

3

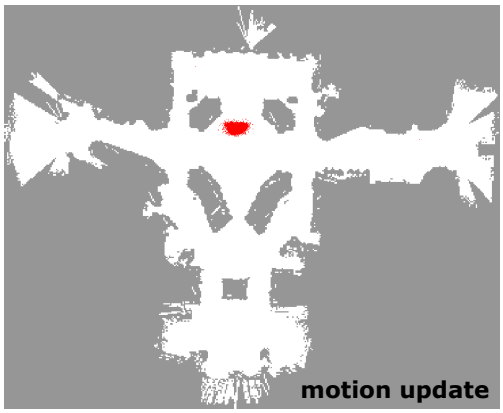
27

1



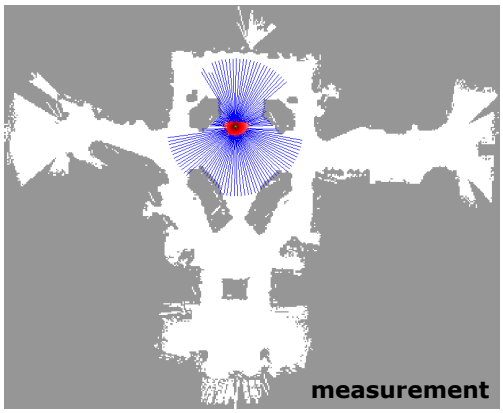
28

2



29

3

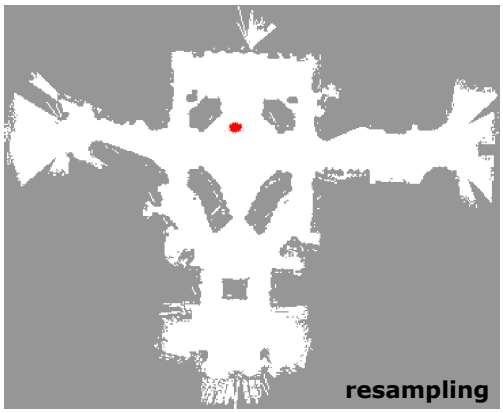


30

1

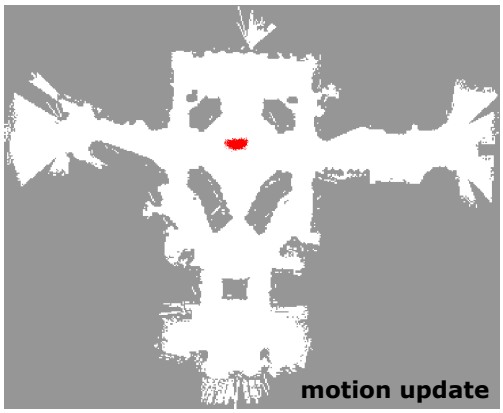
31

2

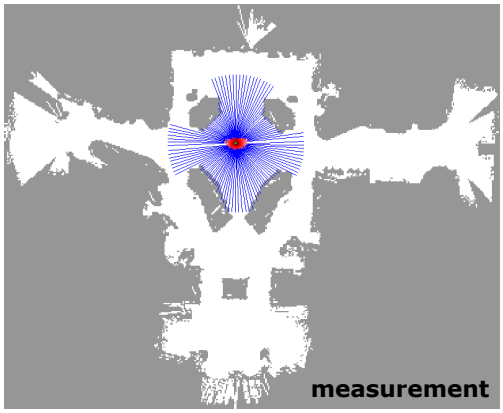


32

3



33



34

3.8.5 Theoretical insight into particle filtering

Step 1: Propagating the dynamics As we introduced in the section on Markov Decision Processes (MDPs), a stochastic dynamical system

$$x_{k+1} = f(x_k, u_k) + \epsilon_k$$

is equivalent to a probability transition matrix $x_{k+1} \sim \mathrm{P}(x_{k+1} \mid x_k, u_k)$. Our goal is to approximate the distribution of $x_{k+1|k}$ using particles. What proposal distribution show we choose? The "closest" probability distribution to $x_{k+1|k}$ that we have available is $x_{k|k}$. So we set

$$\begin{aligned} \text{target} &: \mathrm{P}(x_{k+1} \mid y_1, \ldots, y_k) \\ \text{proposal} &: \mathrm{P}(x_k \mid y_1, \ldots, y_k) \end{aligned}$$

Suppose we had performed resampling on our particle set from the distri-

ℹ In this sense, picking a proposal distribution to draw particles from is like linearization. Better the match between the proposal and the target, fewer samples we need to approximate the target.

bution $x_{k|k}$ and have a set of n particles $\left\{x_{k|k}^{(i)}\right\}$ with equal weights $1/n$

$$\mathrm{P}(x_k \mid y_1, \ldots, y_k) \approx \frac{1}{n} \sum_{i=1}^{n} \delta_{x_{k|k}^{(i)}}(x).$$

Propagating the dynamics in a PF involves computing importance sampling weights. If we had a particle at location x that was supposed to approximate the distribution of $x_{k+1|k}$, as we saw for importance sampling, its importance weight is the ratio of the target and proposal densities at that location

$$w_{k+1|k}(x) = \frac{\mathrm{P}(x_{k+1} = x \mid y_1, \ldots, y_k)}{\mathrm{P}(x_k = x \mid y_1, \ldots, y_k)}.$$

Let us focus on the numerator. We have

$$\begin{aligned}
\mathrm{P}(x_{k+1} = x \mid y_1, \ldots, y_k) &= \int \mathrm{P}(x_{k+1} = x, x_k = x' \mid y_1, \ldots, y_k)\, \mathrm{d}x_k \\
&= \int \mathrm{P}(x_{k+1} = x \mid x_k = x', y_1, \ldots, y_k)\ \mathrm{P}(x_k = x' \mid y_1, \ldots, y_k) \mathrm{d}x_k \\
&= \int \mathrm{P}(x_{k+1} = x \mid x_k = x')\ \mathrm{P}(x_k = x' \mid y_1, \ldots, y_k) \mathrm{d}x' \\
&\approx \frac{1}{n} \int \mathrm{P}(x_{k+1} = x \mid x_k = x')\ \sum_{i=1}^{n} \delta_{x_{k|k}^{(i)}}(x') \mathrm{d}x' \\
&= \frac{1}{n} \sum_{i=1}^{n} \mathrm{P}(x_{k+1} = x \mid x_k = x_{k|k}^{(i)}, u = u_k),
\end{aligned}$$

where the system dynamics is $f(x_k, u_k) + \epsilon_k$ and u_k is the control at time k. The denominator $\mathrm{P}(x_k = x_{k|k}^{(i)} \mid y_1, \ldots, y_k)$ when evaluated at particles $x_{k|k}^{(i)}$ is simply $1/n$. This gives us weights

$$w_{k+1|k}(x) = \sum_{i=1}^{n} \mathrm{P}(x_{k+1} = x \mid x_k = x_{k|k}^{(i)}, u = u_k). \tag{3.37}$$

Let us now think about what particles we should pick for $x_{k+1|k}$. We have from (3.37) a function that lets us compute the correct weight for any particle we may choose to approximate $x_{k+1|k}$.

Say we keep the particle locations unchanged, i.e., $x_{k+1|k}^{(i)} = x_{k|k}^{(i)}$. We then have

$$\mathrm{P}(x_{k+1} = x \mid y_1, \ldots, y_k) \approx \sum_{i=1}^{n} w_{k+1|k}(x_{k|k}^{(i)})\ \delta_{x_{k|k}^{(i)}}(x). \tag{3.38}$$

Draw a picture of how this approximation looks.

You will notice that keeping the particle locations unchanged may be a very poor approximation. After all, the probability density $\mathrm{P}(x_{k+1} \mid y_1, \ldots, y_k)$ is large, not at the particles $x_{k|k}^{(i)}$ (that were a good approximation of $x_{k|k}$), but rather at the transformed locations of these particles

$$f(x_{k|k}^{(i)}, u_k).$$

We will therefore update the locations of the particles to be

$$x^{(i)}_{k+1|k} = f(x^{(i)}_{k|k}, u_k) \tag{3.39}$$

with weight of the i^{th} particle given by

$$\begin{aligned} w^{(i)}_{k+1|k} := w_{k+1|k}(x^{(i)}_{k+1|k}) &= \sum_{j=1}^{n} \mathrm{P}(x_{k+1} = x^{(i)}_{k+1|k} \mid x_k = x^{(j)}_{k|k}, u = u_k) \\ &\approx \mathrm{P}(x_{k+1} = x^{(i)}_{k+1|k} \mid x_k = x^{(i)}_{k|k}, u = u_k). \end{aligned} \tag{3.40}$$

The approximation in the above equation is very crude: we are essentially saying that each particle $x^{(i)}_{k|k}$ is transformed independently of the other particles to a new location $x^{(i)}_{k+1|k} = f(x^{(i)}_{k|k}, u_k)$. This completes the first step of a particle filter and we have

$$\mathrm{P}(x_{k+1} = x \mid y_1, \ldots, y_k) \approx \sum_{i=1}^{n} w^{(i)}_{k+1|k} \delta_{x^{(i)}_{k+1|k}}(x).$$

Step 2: Incorporating the observation The target and proposal distributions in this case are

$$\begin{aligned} \text{target} &: \mathrm{P}(x_{k+1} \mid y_1, \ldots, y_k, y_{k+1}) \\ \text{proposal} &: \mathrm{P}(x_{k+1} \mid y_1, \ldots, y_k). \end{aligned}$$

Since we have particles $x^{(i)}_{k+1|k}$ with weights $w^{(i)}_{k+1|k}$ for the proposal distribution obtained from the propagation step, we now like to update them to incorporate the latest observation y_{k+1}. Let us imagine for a moment that the weights $w^{(i)}_{k+1|k}$ are uniform. We would then set weights

$$\begin{aligned} w(x) &= \frac{\mathrm{P}(x_{k+1} = x \mid y_1, \ldots, y_k, y_{k+1})}{\mathrm{P}(x_{k+1} = x \mid y_1, \ldots, y_k)} \\ &\propto \frac{\mathrm{P}(y_{k+1} \mid x_{k+1} = x)\, \mathrm{P}(x_{k+1} = x \mid y_1, \ldots, y_k)}{\mathrm{P}(x_{k+1} = x \mid y_1, \ldots, y_k)} \quad \text{(by Bayes rule)} \\ &= \mathrm{P}(y_{k+1} \mid x_{k+1} = x). \end{aligned}$$

for each particle $x = x^{(i)}_{k+1|k}$ to get the approximated distribution as

$$\mathrm{P}(x_{k+1} = x \mid y_1, \ldots, y_{k+1}) \approx \sum_{i=1}^{n} \mathrm{P}(y_{k+1} \mid x^{(i)}_{k+1|k})\, w^{(i)}_{k+1|k} \delta_{x^{(i)}_{k+1|k}}(x) \tag{3.41}$$

You will notice that the right hand side is not normalized and the distribution does not integrate to 1 (why? because we did not write the proportionality constant in the Bayes rule above). This is easily fixed by normalizing the

coefficients $\mathrm{P}(y_{k+1} \mid x^{(i)}_{k+1|k})\, w^{(i)}_{k+1|k}$ to sum to 1 as follows

$$w^{(i)}_{k+1|k+1} := \frac{\mathrm{P}(y_{k+1} \mid x^{(i)}_{k+1|k})\, w^{(i)}_{k+1|k}}{\sum_j \mathrm{P}(y_{k+1} \mid x^{(j)}_{k+1|k})\, w^{(j)}_{k+1|k}}.$$

Step 3: Resampling step As we discussed in the previous section, after incorporating the observation, some particles may have very small weights. The resampling procedure resamples particles so that all of them have equal weights $1/n$.

$$\left\{x^{(i)}_{k+1|k+1}, 1/n\right\}_{i=1}^{n} = \text{resample}\left(\left\{x^{(i)}_{k+1|k+1}, w^{(i)}_{k+1|k+1}\right\}_{i=1}^{n}\right).$$

3.9 Discussion

This brings our study of filtering to a close. We have looked at some of the most important algorithms for a variety of dynamical systems, both linear and nonlinear. Although, we focused on filtering in this chapter, all these algorithms have their corresponding "smoothing" variants, e.g., you can read about how a typical Kalman smoother is implemented at https://en.wikipedia.org/wiki/Kalman_filter#Fixed-lag_smoother. Filtering, and state estimation, is a very wide area of research even today and you will find variants of these algorithms in almost every device which senses the environment.