Statistical Techniques in Robotics (16-831, F14) Lecture #21 (Nov 13)

Kalman Filtering

Lecturer: Drew Bagnell Scribes: Jennifer King, Erik Nelson, and Mike Phillips, F2014[1]

1 The Kalman Filter

1.1 Introduction

The Kalman filter was invented as a technique for filtering and prediction in *linear Gaussian systems.* It implements belief computations (Bayesian filtering) for continuous spaces and is not applicable to discrete/hybrid state spaces.

Kalman filters represent beliefs by the moment parameterization, where the belief at time t is represented by a mean μ_t and covariance Σ_t. Contrast this with the Information Filter, which describes belief in terms of the natural paramaterization. All computed posteriors are *Gaussian* if the following properties hold (Markov assumption must hold as well). These three items ensure that all posteriors are always Gaussian for any time t.

1. The state transition probability $p(x_t|u_t, x_{t-1})$ must be linear in their arguments and with added Gaussian noise. This probability therefore assumes linear dynamics and in this form is called *linear Gaussian.* It is represented by the following equation :

$$x_{t+1} = Ax_t + \varepsilon, \qquad \text{where } \varepsilon \sim \mathcal{N}(0, Q) \tag{1}$$

2. The measurement probability $p(y_t|x_t)$ must also be linear with its arguments and with added Gaussian noise. This probability is represented by the following probability:

$$y_{t+1} = Cx_t + \delta, \qquad \text{where } \varepsilon \sim \mathcal{N}(0, R) \tag{2}$$

3. The initial belief must be normally distributed with some mean μ_0 and covariance Σ_0.

Recall that last time, we went over the Lazy Gauss Markov Filter, which did the motion step in the moment paramaterization, and the observation step in the natural paramaterization. Today, we're going to derive both steps in the moment paramaterization, deriving the Kalman Filter. Combining (1) and (2) together, we can show how the state and measurment are thought to come from a normal distribution with a mean vector μ and covariance matrix Σ:

$$\begin{bmatrix} x \\ y \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_x \\ C\mu_x \end{bmatrix}, \begin{bmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_{yy} \end{bmatrix} \right) \tag{3}$$

We can further equate each of the terms in the covariance matrix in terms of the top left term Σ_{xx}.

[1] Some content adapted from previous scribes: Matt Klingensmith and Mike Taylor

Since $\mu_y = C\mu_x$, then we know,

$$\Sigma_{yy} = R + C\Sigma_{xx}C^T \tag{4}$$

Therefore, all that is left is to determine Σ_{xy} since $\Sigma_{yx} = \Sigma_{xy}^T$.

$$\Sigma_{xy} = E\left[\begin{array}{cc} x & y^T \end{array}\right]$$

Since $y = Cx + \delta$, where δ has a mean of 0 and is independent of all other observations, we have

$$\begin{aligned}
\Sigma_{xy} &= E\left[x(Cx+\delta)^T\right] \\
&= E\left[xx^TC^T\right] + E\left[x\delta^T\right] \\
&= E\left[xx^T\right]C^T + E\left[x\right]E\left[\delta^T\right] \\
&= \Sigma_{xx}C^T + 0 \\
\Rightarrow \Sigma_{xy} &= \Sigma_{xx}C^T
\end{aligned} \tag{5}$$

1.2 The Kalman Filter Algorithm

The Kalman filter algorithm is depicted below. The input to the filter is the belief represented by the mean μ_t and covariance Σ_t at the previous timestep.

`forall` (t)
1. $\mu_{t+1}^- = A\mu_t$
2. $\Sigma_{t+1}^- = A\Sigma_t A^T + Q$
3. $\mu_{t+1} = \mu_{t+1}^- + \Sigma_{t+1}^- C^T(C\Sigma_{t+1}^- C^T + R)^{-1}(y_t - C\mu_{t+1}^-)$
4. $\Sigma_{t+1} = \Sigma_{t+1}^- - \Sigma_{t+1}^- C^T(C\Sigma_{t+1}^- C^T + R)^{-1}C\Sigma_{t+1}^-$

The first two lines of the algorithm calculate predicted beliefs represented by μ^- and Σ^- at the new time $(t+1)$ without incorporating the measurement y_t. The mean is then updated using the deterministic version of the state transition function shown in (1) using the variables μ_t^- and Σ_t^-.

In the final two lines, the predicted belief is updated into the desired belief by incorporating the measurement y_t. The quantity $\Sigma_{t+1}^- C^T(C\Sigma_{t+1}^- C^T + R)^{-1}$ is known as the *Kalman Gain*. It details the degree to which each measurement is incorporated into the new state estimate. The third line manipulates the predicted mean in proportion to the Kalman Gain with the quantity $(y_t - C\mu_t^-)$, known as the *innovation* (the difference between what is seen and what we expect to see).

This algorithm is computationally quite efficient, with the only complexities arising from the matrix inversions in lines 3 and 4 ($O(d^2)$) and the multiplications in line 4 ($O(n^2)$).

Taking (4) and (5) from above, we can evaluate the update rules in terms of the mean vector and covariance matrix.

$$\mu_{x|y} = \mu_x + \Sigma_{xx} C^T (R + C\Sigma_{xx} CT)^{-1} (y - C\mu_x) \tag{6}$$

$$\Sigma_{x|y} = \Sigma_{xx} C^T (R + C\Sigma_{xx} C^T)^{-1} C\Sigma_{xx} \tag{7}$$

$$\Rightarrow \mu_{x|y} = \mu_x + \Sigma_{xy}\Sigma_{yy}^{-1}(y - C\mu_x)$$

$$\Rightarrow \Sigma_{x|y} = \Sigma_{xx} - \Sigma_{xy}\Sigma_{yy}^{-1}\Sigma_{yx}$$

Where $\Sigma_{xy}\Sigma_{yy}^{-1}$ is the *Kalman Gain* and $(y - C\mu_x)$ is the *innovation*.

1.3 Kalman Filter Illustration

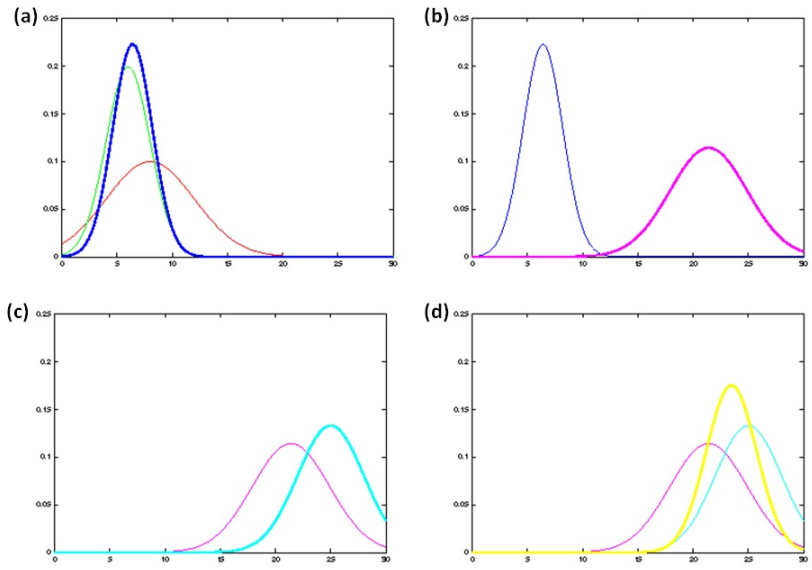


Figure 1: Kalman filtering example for 1D localization

Figure 1 above illustrates the Kalman filter algorithm for simple 1D localization. Supposing the robot moves in the horizontal direction, the prior of robot's position is shown as the red line in (a). Polling its sensors gives the distribution shown by the green line in (a), and the blue line is a result of the final two lines of the Kalman filter algorithm shown above. In (b) the robot has moved to a new position illustrated by the magenta line. It polls its sensor that report back the position given by the cyan line in (c), and the update from the algorithm gives the posterior given by the yellow line in (d).

The Kalman filter alternates the *sensor update* step (lines 3-4), which incorporates sensor data into the present belief, with the *motion update* step (lines 1-2), where the belief is predicted according to the action taken by the robot.

2 The Extended Kalman Filter

2.1 Why should we linearize?

The assumptions of linearity for both the measurement and state transition are essential for the correctness of the Kalman filter. The observation that any linear transformation of a Gaussian random variable yields another Gaussian is important in the derivation of the Kalman filter algorithm, so what happens if state transitions and measurements are no longer linear?

Figure 2 shows the effect of a nonlinear transformation of a Gaussian random variable. The lower right graph shows the random variable distributed about its mean. It is passed through the nonlinear function represented by the upper right graph. The resulting distribution (upper left) is no longer gaussian in shape, rendering Kalman filtering useless over the domain of the nonlinear function.

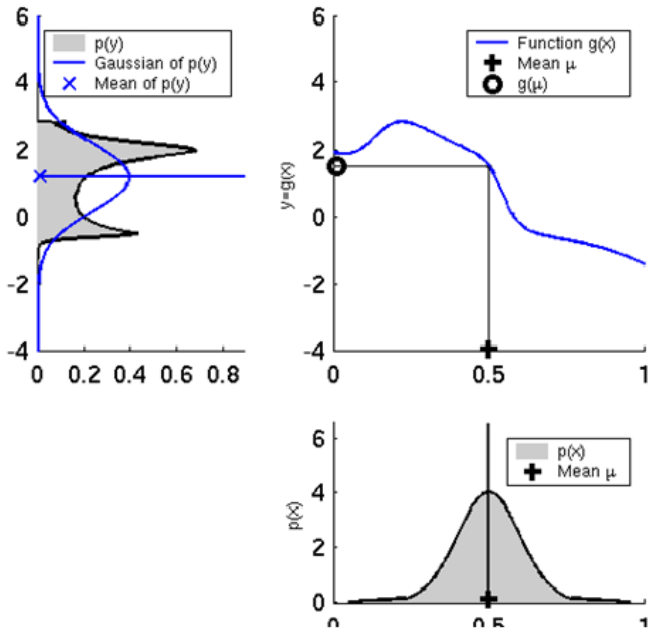


Figure 2: Effect of nonlinear transformation of Gaussian random variable.

2.2 The Extended Kalman Filter

Unfortunately, state transitions and measurements are rarely linear in practice. Thus, we would like to be able to model non-linear transformations with our filter. The *Extended Kalman Filter* or *EKF* relaxes the linearity assumption by assuming that the state transition and measurement probabilities are governed *nonlinear* functions f and g, such that

$$x_{t+1} = f(x_t) + \varepsilon_{t+1}, \quad \text{where } \varepsilon \sim \mathcal{N}(0, Q) \tag{8}$$

$$y_{t+1} = g(x_{t+1}) + \delta_{t+1}, \quad \text{where } \delta \sim \mathcal{N}(0, R) \tag{9}$$

Notice that the function f replaces the matrix A and the function g replaces C. For arbitrary

functions f and g, the belief is no longer Gaussian. To combat this, the EKF calculates a Gaussian approximation to the true belief. This is shown in figure 3

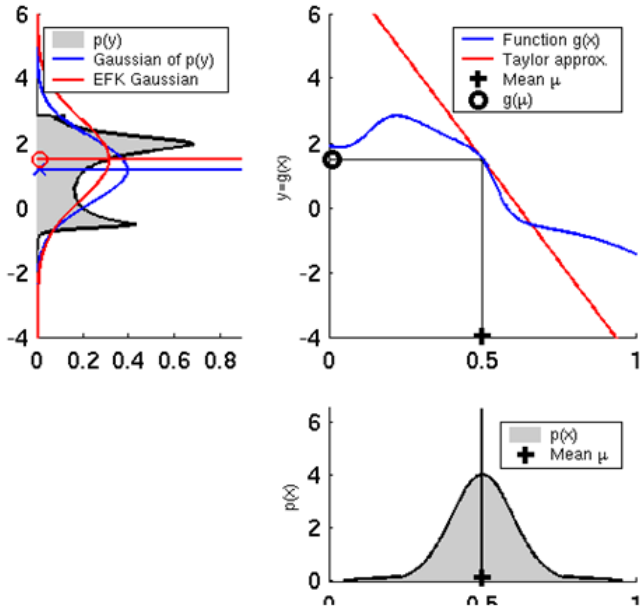


Figure 3: Illustration of EKF

The dashed line on the upper left graph represents the Gaussian approximation (*Linearization*) to the density of the random variable y. Because it represents belief at some time t by some mean μ_t and covariance Σ_t, it still keeps from the original Kalman filter the gaussian belief distribution, but this belief is now an approximation. The goal of the EKF is therefore to efficiently estimate the mean and covariance of a posterior.

Linearization approximates the nonlinear function f and g at the mean of the Gaussian. Projecting it through the linear function shown in the upper right graph of 3 a linear approximation is shown in red of the upper left graph.

2.3 Linearization via the Taylor Series Expansion

One way to perform this linearization is to use the Taylor expansion. In this case we approximate the transformations using the first order Taylor expansion evaluated at the mean estimate of x at time t, μ_t:

$$x_{t+1} \approx f(\mu_t) + \frac{\partial f}{\partial x}(\mu_t)(x_t - \mu_t) + \varepsilon \tag{10}$$

$$y_{t+1} \approx g(\mu_{t+1}^-) + \frac{\partial g}{\partial x}(\mu_{t+1}^-)(x_{t+1} - \mu_{t+1}^-) + \delta \tag{11}$$

The first order derivative terms $\frac{\partial f}{\partial x}(\mu_t)$ and $\frac{\partial g}{\partial x}(\mu_{t+1}^-)$, also referred to as the Jacobian, is used here as the linear transformation, while the $f(\mu_t)$ term just serves to shift the mean of the transformation.

2.4 New Update Rules

Using the non-linear transformation above, we get the following new update equations for our mean and variance estimates for the motion model:

$$\mu_{t+1}^{-} = f(\mu_t^{+}) \tag{12}$$
$$\Sigma_{t+1}^{-} = J_f \Sigma_t^{+} J_f^T + Q \tag{13}$$

And for the sensor model:

$$\mu_{t+1} = \mu_{t+1}^{-} + \Sigma_{t+1}^{-} J_g^T (J_g \Sigma_{t+1}^{-} J_g^T + R)^{-1} [y_{t+1} - g(\mu_{t+1}^{-})] \tag{14}$$
$$\Sigma_{t+1} = \Sigma_{t+1}^{-} - \Sigma_{t+1}^{-} J_g^T (J_g \Sigma_{t+1}^{-} J_g^T + R)^{-1} J_g \Sigma_{t+1}^{-} \tag{15}$$

where $J_f = \frac{\partial f}{\partial x}(\mu_t^{+})$ and $J_g = \frac{\partial g}{\partial x}(\mu_{t+1}^{-})$. These lines replace lines 3 and 4 in the Kalman filter algorithm presented above.

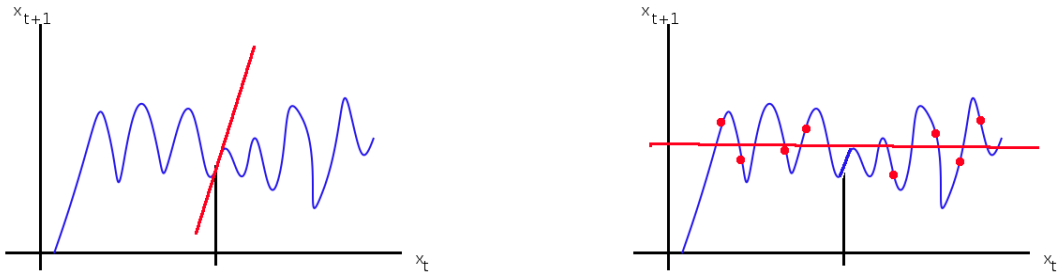
2.5 Problems with Extended Kalman Filtering

- Taylor expansion is a poor approximation of most non-linear functions. The goodness of the linearization depends on two factors: the degree of uncertainty and the amount of local nonlinearity in the functions being approximated.
- Differentiation of the functions f and g can be slightly bothersome, both for you, personally, because you might have forgotten calculus, and because the function might not be continuous across the range in which the linearization is used. An example of a discontinuous function is the hinge loss function for SVM's. Furthermore, for SVM's, you will get driven towards this discontinuity. (Why? Someone figure this out and explain it to me.)
- If the linearization is a poor approximation, the the confidence ellipse will not encapsulate the correct answer unless you make it very large. For this reason, the ellipses are often overconfident. An overconfident filter can cause divergence, because it starts rejecting the measurements it is given. It prances off into a state of egotistical oblivion, giving no respect to the poor measurements that are just trying to help it out.
- EKFs are notoriously bad at handling multiple distinct hypotheses.

Extended Kalman Filters are only as good as their approximation about the mean of the estimate. It is important to keep the uncertainty of the state estimate small when applying EKFs to avoid negative effects from the nonlinearities. Next time, we'll discuss parallels between the Kalman filter and Bayes Linear Regression, and present different kinds of Kalman filters which address some of the shortcomings of the EKF.

3 Statistical Linearization

One drawback to Extended Kalman Filters is that linearization is performed via Taylor expansion around a single point on the function. Linearizing in this manner can result in a poor estimate of



(a) (b)

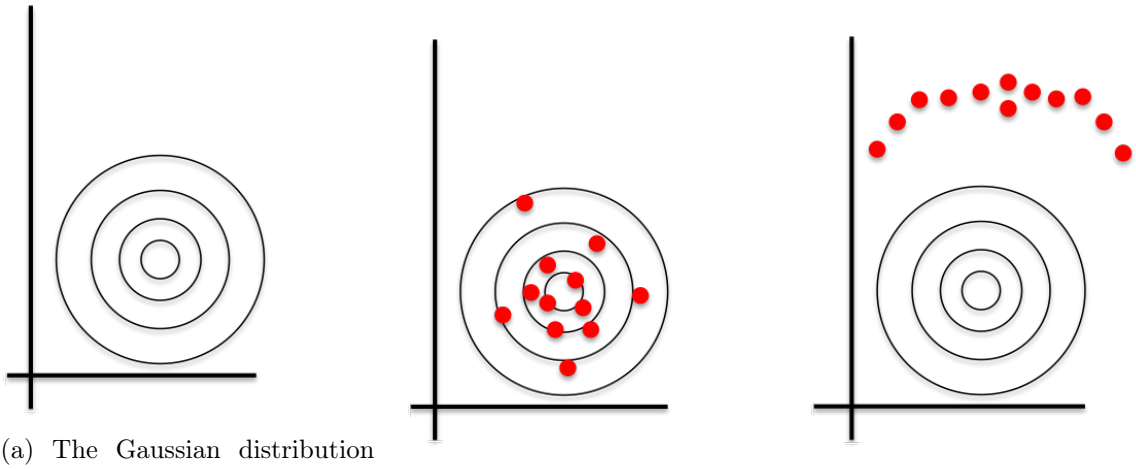
Figure 4: In this motion model, linearization about a point (4a) will pitch steeply back and forth. A better approach would be to fit a line by probing the nonlinear function in multiple places, and performing linear regression to estimate the linearization (4b).

the motion or measurement model if these functions are quickly varying (Figure 4). Instead, one may wish to estimate the linearization matrices A and C empirically by fitting a line to several samples from the nonlinear functions using linear regression. This idea motivates the Unscented Kalman Filter (UKF), which will approximate A and C from points chosen in a structured manner. The UKF will be detailed in the following lecture. Alternatively, a more basic and naïve approach could randomly sample points to use for linear regression. While this approach is not generally used in practice, we derived and analyzed it in class under the name *Monte-Carlo Kalman Filtering*.

3.1 Monte-Carlo Kalman Filter

In the Monte-Carlo Kalman Filter, we forget about linearization and instead directly estimate the joint distribution. The following steps describe this approach:

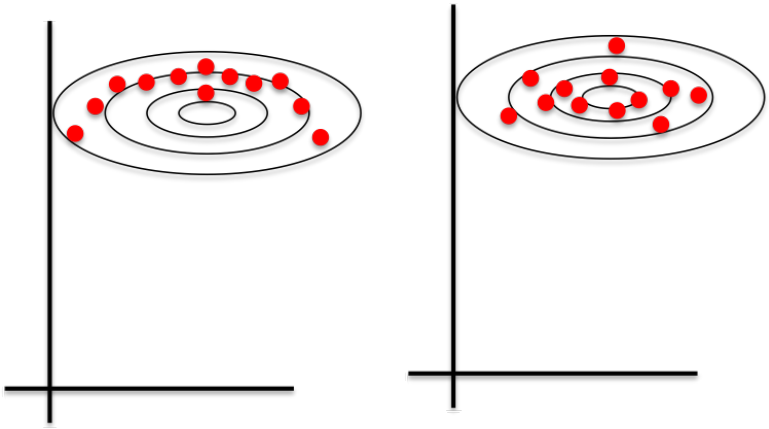
1. Sample $\hat{x}_t^i \sim \mathcal{N}(\mu_t, \Sigma_t)$ for $i = 1 \cdots N$ (Figures 5a, 5b).
2. Push the samples through f (Figure 5c).
3. Compute $\mu_{t+1}^- = \frac{1}{N} \sum_i f(\hat{x}_t^i)$ (Figure 5d).
4. Compute $\Sigma_{t+1}^- = \frac{1}{N} \sum_i (f(\hat{x}_t^i) - \mu_{t+1}^-)(f(\hat{x}_t^i) - \mu_{t+1}^-)^T + Q$ (Figure 5d).
5. Sample $\hat{x}_{t+1}^i \sim \mathcal{N}(\mu_{t+1}^-, \Sigma_{t+1}^-)$ for $i = 1 \cdots N$ (Figure 5e).
6. Push the samples through g
7. Compute $\mu_y = \frac{1}{N} g(\hat{x}_{t+1}^i)$
8. Compute $\Sigma_{yy} = \frac{1}{N} \sum_i (g(\hat{x}_{t+1}^i) - \mu_y)(g(\hat{x}_{t+1}^i) - \mu_y)^T + R$
9. Compute $\Sigma_{xy} = \frac{1}{N} \sum_i (f(\hat{x}_t^i) - \mu_{t+1}^-)(g(\hat{x}_{t+1}^i) - \mu_y)^T$
10. Compute $\mu_{x_{t+1}|y_{t+1}}$ and $\Sigma_{x_{t+1}|y_{t+1}}$ using the Gaussian conditioning rules



(a) The Gaussian distribution from the previous timestep μ_t, Σ_t (shown as a contour map).

(b) N samples drawn from previous Gaussian (μ_t, Σ_t).

(c) The N samples after being transformed by motion model f.



(d) A new Gaussian $(\mu^-_{t+1}, \Sigma^-_{t+1})$ fit to the N samples.

(e) N samples drawn from new Gaussian $(\mu^-_{t+1}, \Sigma^-_{t+1})$, which will then be put through g.

Figure 5: An illustration of some of the main steps of the Monte-Carlo Kalman Filter.

There are two keys to making Monte-Carlo Kalman Filters work:

1. Pick samples intelligently so as to fully describe the distribution. For example, pick samples along the covariance ellipse axis. This can be done by running SVD to get the principle axes and then scaling by the eigenvalues.
2. Pick weights for the samples such that if the system were really linear and Gaussian you would get back the correct μ and Σ.

For the number of samples needed (N) a good rule of thumb is $\sim 10n^2$ where n is the number of dimensions.

There are several instances of the Kalman Filter that use this approach: the Unscented Kalman Filter, the Central Difference Kalman Filter, and many more.