The Extended Kalman and Particle Filter

Josiah Hanna

February 2, 2026

Background: Bayes filter, Kalman filter, Gaussian distributions, Taylor expansions, Jacobian matrix. See last week's reading or Appendix A for a review.

1 Introduction

Last week, we introduced the Bayes filter and said that its prediction and update step are intractable if the number of possible states is large or states are continuous. However, we showed that one special case admitted an exact solution. Specifically, if the state transition and observation function are linear Gaussians and the belief distribution is initialized to be Gaussian then the Kalman filter provides an exact instantiation of the Bayes filter for continuous states and controls.

In practice the linearity assumption is often violated. In this reading, we introduce two new filter methods that approximate the Bayes filter even when this assumption is violated: the extended Kalman and particle filter.

2 Extended Kalman Filter

The Extended Kalman Filter (EKF) is a nonlinear extension of the Kalman Filter that linearizes the system dynamics using a first-order Taylor expansion (see the appendix for a quick review of what this is). The EKF mitigates the KF's assumption that the system is linear.

2.1 Nonlinear System Model

For a nonlinear system:

$$x_t = f(x_{t-1}, u_t) + w_t, \tag{1}$$

$$z_t = h(x_t) + v_t, \tag{2}$$

where f and h are nonlinear functions describing the state transition and observation models, respectively. Alternatively, we can write that $x_t \sim \mathcal{N}(f(x_{t-1}, u_t), Q)$ and $z_t \sim \mathcal{N}(h(x_t), R)$.

2.2 Linearization Using Jacobians

The key idea of the EKF is to make the system approximately linear by replacing the nonlinear f and h with a linear approximation. We linearize the system using the method of Taylor series expansion (see Appendix A for a brief review) around the current state estimate, μ_{t-1}. Specifically, we define matrices A_t and H_t as the Jacobian matrices of f and h respectively, both evaluated at $x = \mu_{t-1}$.

$$A_t = \frac{\partial f}{\partial x}\Big|_{x=\mu_{t-1}}, \quad H_t = \frac{\partial h}{\partial x}\Big|_{x=\mu_{t-1}}. \tag{3}$$

In multivariate calculus, the Jacobian matrix represents the first-order partial derivatives of a vector-valued function. Given a function $f : \mathbb{R}^n \to \mathbb{R}^m$, the Jacobian matrix J is defined as:

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}. \tag{4}$$

Each row represents one of the outputs of f and each column represents one of the inputs. Intuitively, each entry, i, j, captures how fast output i is changing as input j changes.

The system is then linearly approximated as:

$$x_t = A_t x_{t-1} + f(\mu_{t-1}, u_t) + w_t, \tag{5}$$

$$z_t = H_t x_t + h(\mu_{t-1}) + v_t, \tag{6}$$

where w_t and v_t are zero-mean Gaussian noise as defined above.

Note that the Taylor expansion may only be a good approximation around the expansion point. Consequently, the matrices, A_t and H_t are recomputed as the belief, μ_t, changes over time.

2.3 Prediction and Update

The EKF follows very similar updates to the Kalman Filter except using the linearized system in place of the true system model:

$$\bar{\mu}_t = f(\mu_t, u_t) \tag{7}$$

$$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^\top + Q \tag{8}$$

We then compute the Kalman gain:

$$K_t = \overline{\Sigma}_t H_t^\top (H_t \overline{\Sigma}_t H_t^\top + Q)^{-1} \tag{9}$$

And finally, apply the updates:

$$\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t)) \tag{10}$$

$$\Sigma_t = (I - K_t H_t)\overline{\Sigma}_t \tag{11}$$

3 Strengths and Weaknesses of Extended Kalman Filters

3.1 Extended Kalman Filter (EKF)

Strengths:

- Extends the Kalman Filter to nonlinear systems.
- Works well for systems that can be *locally* linearized. The Taylor expansion is only an accurate representation of the transition/observation function for x close to μ_t. The closer these functions are to linear or the tighter the belief covariance matrices are, the more accurate the Taylor expansion approximation will be.

Weaknesses:

- Accuracy depends on the quality of linearization. See discussion above. Highly nonlinear systems will have poor accuracy with an EKF.
- Computationally more expensive than the standard Kalman Filter due to need to compute Jacobians.
- May diverge if the system is highly nonlinear.

4 Particle Filters

We will now introduce a class of methods that can approximate $\texttt{bel}(x_t)$ in continuous state spaces and for complex, non-linear probability distributions over next states and observations. This is the class of particle-based methods.

The high-level idea of the family of particle-based methods is to represent the robot's $\texttt{bel}(x_t)$ by a finite collection of weighted *particles*. Each particle represents a possible value for the state x_t. For example, if the state is defined as the pose of the robot, then each particle corresponds to a possible pose. Let x_t^i represent the i^{th} particle and w_i represent its weight. Weights will be defined such that the particles for states that should be very likely under $\texttt{bel}(x_t)$ have a large weight and particles for unlikely states have a small weight. At each time-step, t, a particle-based algorithm updates the weights or re-samples the set of particles using the latest control and sensor data, u_t and z_t.

5 Normalized Importance Sampling

The first particle-based algorithm we will encounter is the normalized importance sampling (NIS) filter. The NIS filter uses a set of N particles. For example, let's consider a robot moving around in a square room where its state is fully described by its pose, (x, y, θ). Each particle, x_i, is then defined as a pose (x_i, y_i, θ_i) and the N particles are scattered across different possibilities for the robot's pose. Note that there are an infinite number of possible values for the robot's pose (since x, y, θ are real-valued) but only a finite number of particles. Each particle is given an initial weight, w_i. By default, the weight is $w_i \leftarrow \frac{1}{N}$ so that initially no particle is more likely than the others. Of course, if prior knowledge means that some states are known to be more likely than others then larger weight can be given to matching particles.

At each time-step, t, the algorithm receives the robot's control, u_t, and new sensor observation, z_t. The NIS filter takes the following steps:

1. For each particle, x_{t-1}^i, sample $x_t^i \sim p(\cdot|x_{t-1}^i, u_t)$.
2. For each particle, update the weight $w_i \leftarrow w_i \cdot p(z_t|x_t^i)$.
3. Normalize the weights such that $\sum_i w_i = 1$.

Let's consider what the NIS filter is doing. First, the state value associated with each particle evolves stochastically according to the state transition function and the robot's control. Second, the weight on each particle is updated based on how well the state value matches the observed z_t. If $p(z_t|x_t^i)$ is small relative to other states then the weight (once normalized) will decrease and will conversely increase if $p(z_t|x_t^i)$ is larger.

An important strength of the NIS (and other particle-based) filters is that the state transition model does not have to be known in closed form, provided we have a means to sample from it. This is convenient as it is often easier to generate samples than it is to write out a full probability density function.

6 Extracting a State Estimate from the Particles

Particle-based methods represent the robot's belief distribution as a discrete set of particles. However, in robotics applications, we often want to compute an estimate of the most likely state and use this for decision-making. One way to do this would be to just return the particle with maximum weight:

$$\hat{x}_t = x_t^i \text{ with } i \leftarrow \arg\max_i w_i.$$

Doing so will restrict the filter to only returning one of the particle values, even if multiple particles have high weight. A better alternative is to return a weighted average:

$$\hat{x_t} = \sum_i w_i x_t^i.$$

Unlikely particles will receive near-zero weight and won't affect the result while other particles with high weight will affect the average. In the extreme case where only a single particle has high weight ($w_i \approx 1$) then the weighted average is the same as just returning the most likely particle.

A disadvantage of using a weighted average is that the belief could be multi-modal, resulting in the weighted average returning a state value that is very unlikely. For example, in the room example, if the NIS filter has equal weight on two particles representing distinct locations then the weighted average returns a location that is halfway between these location. That location itself may be very unlikely and yet the weighted average returns it because it collapses the multi-modal belief into a single point. A possible solution is to run a clustering algorithm on the particles, compute a weighted average for each cluster, and then return a set of candidate states.

7 The Particle Filter

A main drawback of the NIS filter is that the weights will tend toward either 0 or 1. Any particle with $w_i \approx 0$ means that we are keeping track of a very unlikely state possibility. This is wasteful of finite computational resources! We should instead try to repurpose such low weight particles so that they correspond to more likely states.

The particle filter accomplishes this goal by resampling the particles according to their weight. The particle filter follows the same steps as the NIS filter but adds two additional steps after normalizing the weights:

1. Define a probability distribution, q, over the particles such that particle i has probability w_i. Sample N particles with replacement from q. Discard the old particles.
2. Set the weight for each particle to $w_i \leftarrow 1/N$.

With these additional steps, unlikely states tend to not be resampled and are instead replaced with particles representing more likely states. Overtime, the set of particles will eventually only contain particles representing more likely states under $p(x_t|z_{1:t}, u_{1:t})$.

A Review

The following appendices provide background on some mathematical concepts used in these notes. If you are familiar with these topics, you can skip over them.

A.1 Gaussian Distributions

The Gaussian (or multi-variate normal) distribution is one of the most common probability distributions for continuous random variables. Let $x \in \mathbf{R}^d$ be a d-dimensional vector. We say that x has a Gaussian distribution if:

$$p(x) = \frac{1}{\sqrt{2\pi \det \Sigma}} e^{-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)},$$

where $\mu \in \mathbf{R}^d$ is called the mean vector and $\Sigma \in \mathbf{R}^{d\times d}$ is the covariance matrix. The matrix Σ must be positive-definite.

A few additional notes on the Gaussian distribution:

1. When $d = 1$, the Gaussian distribution is the familiar bell curve.
2. Under a Gaussian distribution, $p(x)$ is maximized when $x = \mu$, i.e., the mean is the most likely outcome.
3. The covariance controls the spread of $p(x)$. When the covariance is large, then $p(x)$ is spread out and values of x that are farther from μ receive more probability density. When the covariance is small then most sampled values of x will be close to μ.

For notation, we will denote Gaussian distributions as $\mathcal{N}(\mu, \Sigma)$ and the density of x under a Gaussian with mean μ and covariance Σ as $\mathcal{N}(x; \mu, \Sigma)$.

A.2 Jacobian Matrix

In multivariate calculus, the Jacobian matrix represents the first-order partial derivatives of a vector-valued function. Given a function $f : \mathbb{R}^n \to \mathbb{R}^m$, the Jacobian matrix J is defined as:

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}. \tag{12}$$

The Jacobian is used in the Extended Kalman Filter to linearize nonlinear functions by approximating them with a first-order Taylor series expansion.

A.3 Taylor Series Expansion

The Taylor series is a mathematical tool used to approximate a function by expanding it around a given point. For a function $f(x)$ that is differentiable, the first-order Taylor expansion around a fixed point x_0 is given by:

$$f(x) \approx f(x_0) + \left.\frac{df}{dx}\right|_{x_0} (x - x_0). \tag{13}$$

Note that the right hand side of Equation (13) is a linear function in x since $\left.\frac{df}{dx}\right|_{x_0}$ and $f(x_0)$ are constant with respect to x. The approximation is perfect for $x = x_0$ and will generally become worse as x moves away from x_0. If f is linear ($f(x) = ax + b$) then the approximation is again perfect since $f(x) \approx ax_0 + b + a(x - x_0) = ax + b = f(x)$. However, for nonlinear f, the approximation has error that increases as $|x - x_0|$ grows.

In the case of multiple variables, the Taylor expansion generalizes to:

$$f(x) \approx f(x_0) + J(x_0)(x - x_0), \tag{14}$$

where $J(x_0)$ is the Jacobian matrix evaluated at x_0. The Taylor expansion is useful in the Extended Kalman Filter because it allows nonlinear functions to be locally approximated by linear functions, enabling the use of linear filtering techniques.