The Kalman Filter and Related Algorithms: A Literature Review

Article · May 2011

1 author:

Corey Montella
Lehigh University
9 PUBLICATIONS **37** CITATIONS

SEE PROFILE

The Kalman Filter and Related Algorithms

A Literature Review

Corey Montella

Abstract

Bayesian state estimation is the process of recursively estimating the state of a system. In this paper we will summarize three highly influential algorithms that have been implemented in fields as diverse as signal analysis, space flight control, and robotics: the Kalman Filter, the Extended Kalman Filter, and the Particle Filter. We will introduce each algorithm, analyze its complexity, correctness, and accuracy, and finally compare the three in a practical example.

1. Introduction

The ultimate goal of algorithms research is to find an optimal solution for a given problem. However, there are few problems in computer science that can be considered completely solved. That is, it is rare that one can show an algorithm is completely optimal for its problem domain. One of the few areas where a provably optimal algorithm can be found, however, is in Bayesian state estimation. Here, the Kalman Filter, an algorithm that propagates a system's varying quantities over time, can be shown to be the best algorithm possible for its domain. This paper is an introduction to the Kalman Filter and several related Bayesian state estimators. In the rest of this introduction we will introduce Bayesian Filters. In the next section we will describe the Kalman Filter in detail. Then, we will detail the Extended Kalman Filter, and finally the Particle Filter. For each filter, we will provide a sketch of the algorithm, analyze its computational time complexity, and finally sketch a proof of its correctness, or analyze how well it approximates the optimal solution..

1.1. General Bayesian State Estimation

The focus of this paper is the Kalman Filter and its related algorithms. These are examples of Bayesian Filters, named after their application of Bayes' law, expressed in Equation 1.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \qquad \text{Equation 1}$$

Stated simply, Bayes' law says the probability of estimating A given B has occurred is equal to the normalized probability of B given A has occurred, multiplied by the probability of A occurring. Bayes' law is useful when we cannot measure P(A|B) explicitly, but we can measure P(B|A) and P(A). A typical example used to illustrate Bayes' law is estimating the probability of having a disease given a positive test result. In this scenario, the probability of a false-positive and false-negative are the observable quantities used to calculate this.

Bayesian Filters use Bayes' law to estimate an unobservable state of a given system using observable data. They do this by propagating the posterior probability density function of the state using a transition model. For example, in robotics, the system's state is typically the pose of a robot in its environment, or the configuration of joints in an actuator. This is usually denoted s_t. Also, in robotics the observable data is typically a control that tells the robot how to move, and an observation about the environment it makes with its sensors. These are usually denoted u_t and z_t respectively. The control can be velocity command give to motors, or perhaps desired angles given to joints. The observation can be a range and bearing to an obstacle. Finally, using Bayes' law and the Markov assumption (that the current sate depends only on the previous state and not any state before it) we can state the Bayesian Filter below, which is derived from Equation 1 [4].

$$P(s_t|z_t, u_t) = \eta P(z_t|s_{t-1})P(s_{t-1}|u_t) \qquad \text{Equation 2}$$

Here, η is a normalization factor, z_t is an observation made by the system, u_t is a transition that modifies the state, and s_t is the current state vector. Equation 2 is the basis of all Bayesian state estimators. The difference between them is how the probabilities are estimated, and the form of the input probability distributions. In this paper we will look at three different Bayesian filters, the Kalman Filter, the Extended Kalman Filter, and the Particle Filter. The Kalman Filter is a linear Gaussian estimator, and is optimal. The Extended Kalman Filter is a nonlinear version of the Kalman Filter that can handle more problem instances, but is not optimal. The Particle Filter is a non-parametric estimator that is more flexible than the two but is more computationally expensive.

2. The Kalman Filter

The Kalman Filter is a very rare algorithm, in that it is one of the few that are provably optimal. It was first published by Rudolf E. Kalman in his seminal 1960 paper titled *A New Approach to Linear Filtering and Prediction Problems* [1]. It is used in areas as diverse as aeronautics, signal processing, and futures trading. At its core, it propagates a state characterized by a Gaussian distribution using linear transition functions in an optimal way. Since it is optimal, it has remained relatively unchanged since it was first introduced, but has received many extensions to apply it to more than just linear Gaussian systems. In this section we sketch the Kalman Filter algorithm, and analyze its computational complexity in the time domain. Finally, we show it is an optimal algorithm, and that no algorithm can do better. This section is largely based off Kalman's original paper.

2.1. Outline of the Kalman Filter

Since the Kalman Filter is a Bayesian filter, our goal is to solve Equation 2. However, we first must note the Kalman Filter comes with several assumptions:

1. The state transition is linear in the form

$$s_{t+1} = As_t + Bu_t + w_k \quad \text{Equation 3}$$

   where s_t is the state, u_t is the control, and w_k is added Gaussian noise.
2. The measurement is linear in the form

$$z_t = Hs_t + v_k \quad \text{Equation 4}$$

   where z_t is the observation, and v_k is added Gaussian noise.
3. The system is continuous.

While these assumptions restrict the applicability of the Kalman Filter, we will show later they also ensure its optimality.

The algorithm is structured in a predictor-corrector format. The general idea is to project the state forward, using a state transition function. Then this state is corrected by incorporating a measurement of the system's observable quantities. The algorithm can be divided into two distinct phases: a time update phase and a measurement update phase.

2.1.1. Time Update

In the time update phase, the state is projected forward using Equation 3. However, we also must propagate the uncertainty in the state forward. Since the state is a Gaussian distribution, and is fully parameterized by a mean s_t and covariance P_t, we can update the covariance as in Equation 5.

$$P_{t+1} = AP_tA^T + Q \quad \text{Equation 5}$$

Here, A is the same matrix used to propagate the state mean, and Q is random Gaussian noise. Equations 3 and 5 exploit a general property of Gaussians: adding two Gaussians results in a Gaussian, and applying a linear transformation to a Gaussian yields a Gaussian. There properties of Gaussian are crucial to the optimality of the filter. After the time update phase, the original Gaussian characterized by s_t and P_t is a new Gaussian, now characterized by s_{t+1} and P_{t+1}. This concludes the time update phase, and represents the prediction step of the algorithm.

2.1.2. Measurement Update

The measurement update phase is the correction step of the Kalman Filter, wherein a measurement of an observable variable is made and fused with the prior distribution to estimate the posterior. First, we make a measurement of the system using our linear measurement model in Equation 4. After the measurement is made, we form what is known as the Kalman Gain, depicted in Equation 6. This is the key step of the Kalman Filter.

$$K = PH^T(HPH^T + Q)^{-1}$$ Equation 6

In section 2.3 we will derive this gain, and show it is this that makes the Kalman Filter optimal.

Next, we calculate what is known as the innovation (Equation 7). This is the difference between the expected observation, and the actual observation.

$$\hat{z} = (z_t - H_t s_t)$$ Equation 7

Now we are ready to calculate the posterior distribution, by combining Equations 6 and 7. Equation 8 corrects the mean while equation 9 corrects the covariance.

$$s_t^- = s_t + K\hat{z}$$ Equation 8

$$P_t^- = (I - K_t H_t)P_t$$ Equation 9

Here, s_t^- and P_t^- fully parameterize the posterior distribution. The preceding steps represent a single iteration of the Kalman filter. This output is then used as input to a subsequent observation, along with a new control and observation. Figure 1 depicts the Kalman Filter graphically, while Figure 2 summarizes the algorithm.

Figure 1: (a) the prior distribution in black. (b) State is propagated forward in blue, adding uncertainty. (c) A measurement it made in red. (d) Kalman Filter produces the prior distribution in green. This distribution has less uncertainty compared to both state and measurement as a result of Bayes' Law.

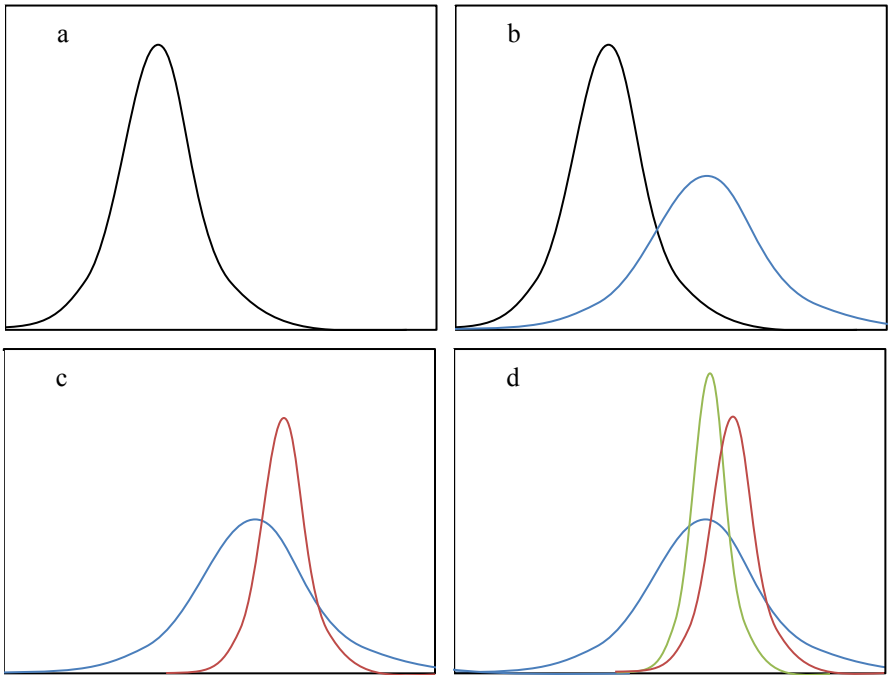


Figure 2: The full Kalman Filter algorithm in pseudocode.

$Algorithm: Kalman\_Filter$
$Input: s_{t-1}, P_{t-1}, u_t, z_t$
$Output: s_t^-, P_t^-$

1. $s_t = As_{t-1} + Bu_t + w_t$
2. $P_t = AP_{t-1}A^T + Q$
3. $K = PH^T(HPH^T + Q)^{-1}$
4. $\hat{z} = (z_t - H_t s_t)$
5. $s_t^- = s_t + K\hat{z}$
6. $P_t^- = (I - K_t H_t)P_t$
7. $return\ s_t^-, P_t^-$

2.2. Complexity Analysis

In the following section we analyze the computational time complexity of the Kalman Filter presented in Figure 2.

2.2.1. Time Complexity

The Kalman Filter is recursive, in that we use the output from one application of the filter as input to the next. We only need to determine the complexity of a single application of the filter to determine the complexity of the full algorithm.

Line 1: This line consists of two matrix multiplications and two matrix additions. Here, s_t and u_t are 1xn vectors, where n is the size of the state. A, B are two nxn matrices. Also, w_t is a 1xn vector. The best known algorithm for multiplying a mxn and a nxp matrix runs as O(nmp). The run time for adding two mxn matrices runs as O(mn). Therefore, the total time complexity of this line is:
$O(2n^2 + 3n)$

Line 2: This line consists of two matrix multiplications, a transpose operation, and a matrix addition. The best known algorithm for multiplying two nxn matrices runs as $O(n^{2.376})$. The transpose operation runs as $O(n^2)$. So the total complexity of this line is:
$O(2n^{2.376} + 2n^2)$

Line 3: This line consists of four matrix multiplications, two transpose operations, a matrix addition, and a matrix inversion. The inverse operation runs as $O(n^{2.376})$. So the total complexity of this line is:
$O(5n^{2.376} + 3n^2)$.

Line 4: This line consists of a matrix subtraction and a matrix multiplication. The total complexity is:
$O(n^2 + n)$.

Line 5: This line consists of a matrix addition and a matrix multiplication, so again the complexity is:

$O(n^2 + n)$.

Line 6: Our final line consists of two matrix multiplications and a matrix addition, so the total complexity is :
$O(2n^{2.376} + n^2)$.

Therefore the total time complexity of a single application of the Kalman Filter is $O(9n^{2.376} + 10n^2 + 5n) \in O(n^{2.376})$.

2.3. Correctness

Several times in this paper, we made the claim that the Kalman Filter propagates the state s_t in an optimal manner. This is due to the choice of the Kalman Gain, which we will now show minimizes the mean square error in the posterior state estimate. That is, we want our choice of the Kalman Gain to minimize

$$E[|s_t - s_t^-|^2] \qquad \text{Equation 10}$$

Equation 10 is identical to minimizing the trace of P_t^-. From first principles, we know that the standard covariance update formula is:

$$P_t^- = (I - KH)P_{t-1}(I - KH)^T + KQK^T \qquad \text{Equation 11}$$

By expanding Equation 11 and collecting terms, we arrive at the following formula

$$P_t^- = P_{t-1} - KHP_{t-1} - P_{t-1}H^TK^T + K(HP_{t-1}H^T + Q)K^T \qquad \text{Equation 12}$$

We want to minimize the trace of this, so we take the derivative of the trace and set it equal to zero.

$$\frac{\partial tr(P_t^-)}{\partial K} = -2(HP_{t-1})^T + 2K(HP_{t-1}H^T + Q) = 0 \qquad \text{Equation 13}$$

Solving for K we arrive at Equation 6, the optimal Kalman Gain that minimizes the sum of the square error in the posterior. Therefore we can be sure that if we satisfy all the assumptions of the Kalman Filter, that the output will be optimal [4].

3. Extended Kalman Filter

One major limitation of the Kalman Filter is the strict set of problems to which it applies: problems with linear state transition and linear measurements with added Gaussian noise. While a great many problems can be modeled this way, it would be nice to apply the Kalman Filter to other problems, due to its optimality.

The Extended Kalman Filter was invented just for this purpose. It works through a process of linearization, where the nonlinear transition and observation functions are approximated by a Taylor Series expansion. This section derives from a seminal paper on the Unscented Kalman Filter, which is similar to the Extended Kalman Filter [2].

3.1. Outline of the Extended Kalman Filter

With the Extended Kalman Filter, we can no longer assume a linear process update or observation model. We express this condition in Equations 14 and 15

$$s_t = g(s_{t-1}, u_t) \quad \text{Equation 14}$$

$$\hat{z}_t = h(s_{t-1}) \quad \text{Equation 15}$$

Here, the process update and observation models are characterized by two potentially nonlinear functions $g(s_{t-1}, u_t)$ and $h(s_{t-1})$. Since the Kalman Filter works on only linear inputs, we must linearize these functions to propagate the state covariance matrix forward. We do this by approximating the function as a line tangent to the actual function at the mean value. This line is found by expanding the nonlinear functions in a Taylor Series around the mean, and taking the first order approximation. This expansion is expressed as $y = f(x+\varepsilon) \approx f(x) + J\varepsilon$, where J is the Jacobian of f(x). Equation 16 demonstrates how we use the Jacobian to propagate our covariance matrices.

$$C_y = E[(y - \bar{y})(y - \bar{y})^T] \approx E[J\varepsilon\varepsilon^T J^T] = JC_x J^T \quad \text{Equation 16}$$

Here, C_x is our current covariance; J is the Jacobian of our nonlinear state transition function; ε is zero mean Gaussian noise; and C_y is our transformed covariance matrix. However, this result is only approximate due to our use of the Jacobian. Thus, we need to linearize both equations 14 and 15 by taking the gradient of each with respect to the state s_t, as in Equations 17 and 18.

$$G_t = \nabla_s g(s_{t-1}, u_t) \quad \text{Equation 17}$$

$$H_t = \nabla_s h(s_{t-1}) \quad \text{Equation 18}$$

Thus we now follow the same predictor-corrector form of the original Kalman Filter in two distinct phases.

3.1.1. Time Update

First, we propagate the mean forward to find the mean of our prior distribution. This is simply applying Equation 14. Next, we have to propagate the covariance forward, which was shown in Equation 14 to just be the covariance convolved with the appropriate Jacobian. In this case we use G_t.

$$P_t = G_t P_{t-1} G_t^T + Q \qquad \text{Equation 19}$$

Here, Q is again zero mean Gaussian noise of the process model. These two equations provide us with a fully parameterized Gaussian prior. We now move to the measurement update phase.

3.1.2. Measurement Update

The measurement update phase is largely the same as in the Kalman filter, two important exceptions. First, the Kalman Gain is calculated using a Jacobian. This has the unfortunate consequence that Kalman Gain can no longer be considered optimal, and thus the Extended Kalman Filter cannot be the best possible algorithm for filtering nonlinear systems. We calculate the Kalman Gain as in Equation 6, but this time H_t is found using Equation 18.

The second difference is in calculating the innovation. In the Kalman Filter, the observation model had to be linear. This requirement was only to ensure the calculation for the Kalman Gain worked correctly. Here, our observation model can be nonlinear, but the way we calculate our innovation is straight forward.

$$\hat{z} = (z_t - h(s_t)) \qquad \text{Equation 20}$$

With this last equation, we can estimate the posterior in the standard Kalman Filter way. The pseudocode for the Extended Kalman Filter is depicted in Figure 3.

Figure 3: The full Extended Kalman Filter algorithm in pseudocode.

$Algorithm: Extended\_Kalman\_Filter$
$Input: s_{t-1}, P_{t-1}, u_t, z_t$
$Output: s_t^-, P_t^-$

1. $s_t = g(s_{t-1,} u_t)$
2. $P_t = G_t P_{t-1} G_t^T + Q$
3. $K = P_t H^T (H P_t H^T + Q)^{-1}$
4. $\hat{z} = (z_t - h(s_t))$
5. $s_t^- = s_t + K\hat{z}$
6. $P_t^- = (I - K_t H_t) P_t$
7. $return\ s_t^-, P_t^-$

3.2. Complexity Analysis

Computationally, the algorithm for the Extended Kalman Filter is exactly the same as the original Kalman Filter. However, there are two differences that could affect the time complexity of this version: the functions $g(s_{t-1,}u_t)$ and $h(s_t)$. These two functions are specific to the system, so we cannot say exactly what their computational complexity is. Therefore, we can just express the complexity of the whole algorithm as max(O($n^{2.376}$,O($g(s_{t-1,}u_t)$),O($h(s_t)$)). In most cases, the complexity of these functions should be close to constant, but there could be a situation where propagating the state forward is very complex.

3.3. Accuracy

We showed earlier that the Kalman Filter is optimal, in that its choice in the Kalman Gain minimizes the sum of the square of the error in the posterior state estimate (that is, it minimizes Equation 10). This optimal Kalman Gain was only possible because the transition and observation models were linear in their arguments. With the Extended Kalman Filter, we relaxed this requirement, by linearizing nonlinear models and calculating their Jacobians to propagate uncertainty. This means the Kalman Gain in the Extended Kalman Filter is no longer optimal, but an approximation. But exactly how far from optimal is the Extended Kalman Filter? The answer is dependent on two factors:

First, the accuracy of the filter largely depends on how linear the transition and observation models are around the mean. If these functions are relatively, linear, the approximation will be good. However, it is possible that these will be highly nonlinear, and thus the performance of the Extended Kalman Filter will suffer [4].

Second, the modality of the transition and observation models will affect the Extended Kalman Filter's performance; if these functions are multimodal, then this filter can diverge. For this reason, nonparametric filters like the Particle Filter, which we are about to introduce, are better suited for these types of systems [4].

4. Particle Filter

Until now, we have focused exclusively on parametric Bayesian Filters that manipulate Gaussian distributions, propagating with a transition model. We now turn to a nonparametric filter, with a unique approach to calculating the posterior distribution: the Particle Filter. As its name suggests, it uses a set of hypothesis called particles as guesses for the true configuration of the state. In the following section we present the basic Particle Filter algorithm. This section derives from this seminal 1993 paper by N. Gordon, D. Salmond, and A. Smith titled *Novel approach to nonlinear/non-Gaussian Bayesian state estimation* [3], where they called the algorithm the "Bootstrap Filter".

4.1. Outline of the Particle Filter

The particle filter is different from previous filters in that it is not limited by linear models or Gaussian noise. This flexibility is due to how it represents the probability density function as a set of samples known as particles. Each sample is taken from a proposal distribution, and weighted according to how well it matches a target distribution. After weighting, the particle distribution does not match the posterior, so we carry out the key step in the Particle Filter algorithm: importance sampling. Here, we pick particles from the proposal particle set and add them to the posterior particle set with a frequency proportional to their weight. This cycle is summarized in Figure 4. We now look at each phase of the Particle Filter in detail.

Figure 4: The basic cycle of the Particle Filter involves 3 steps. The first step involves sampling from a proposal distribution. The second step involves applying an importance weight to those samples. The final step is the key step, that resamples the particle set based on the assigned weights.

1.1 **Sample** – Sample M particles from the proposal distribution
2.1 **Weight** – Weight the samples based on how well they match the posterior distribution
3.1 **Resample** – Pick samples from the set proportional to their weight

4.1.1. Sample

The first step in the particle filter is to go through every particle and sample from a proposal distribution.

$$sample\ s_t^{[m]} \sim p(s_t | s_{t-1}^{[m]}, u_t) \qquad \text{Equation 21}$$

Just as the Bayesian Filters we looked at in previous sections, the Particle Filter is a recursive algorithm, so we therefore sample the current state using the previous state. Here, the superscript [m] on both state variables implies that the state s_t is derived from the same particle in the previous particle set. The same control input u_t as the previous methods is used to propagate the state forward. Since the transition function is noisy, each particle goes through a different transition, which adds variety to the particle set.

Note that we do not explicitly state how to obtain this sample. This is dependent on the exact system, and is one of the requirements of the Particle Filter, that a proposal distribution must be available to sample from. In many applications, this is readily available. If not the case, it sometimes suffices to sample from a Gaussian distribution with appropriate mean and covariance.

4.1.2. Weight

Next each particle is weighted based on how well it matches the posterior distribution. This is expressed in Equation 22.

$$w_t^{[m]} = p(z_t|x_t^{[m]}) \qquad \text{Equation 22}$$

This is equivalent to the measurement update phase in the Kalman Filter, where the observation is incorporated into the belief. There are many different methods to weight particles, but most involve estimating the following quotient

$$w_t^{[m]} = \frac{f(s_t^{[m]})}{g(s_t^{[m]})} \qquad \text{Equation 23}$$

Where $g(s_t^{[m]})$ is our proposal distribution, and $f(s_t^{[m]})$ is our target distribution. When we have completed these two steps, we are finally free to add the new weighted particle to a temporary particle set.

4.1.3. Resample

Also known as importance sampling, the resample step uses the newly generated temporary particle set to generate the final posterior distribution. Just as with weighting the particles, there are many ways to accomplish importance sampling, but doing this incorrectly can lead to the wrong posterior distribution. Essentially, the resampling step reduces variance in the particle set, which decreases the accuracy of the posterior approximation. In general, we sample with particles with replacement with a frequency proportional to their weight. However there are myriad variations on this general method. Below are several guidelines to follow when implementing a resampling routine [4].

- **Do not resample too frequently** – resampling too often can lead to a situation where the particle variance is artificially low. A common example used to illustrate this is a stationary robot with no sensing. If this robot uses a particle filter to estimate its position, and is allowed to resample the particle set, the robot will uniquely determine its position after a long enough time, since we sample with replacement. Yet a robot that does not sense should never be able to determine its position. Therefore, the most you should resample is once per control and observation. Sometimes it is even beneficial to resample only when the variance in the particle set is too high [4].

- **Do not resample independently** – The simplest method for resampling would be to simply choose a particle at random and generate a random number. If the weight of that particle exceeds the random number, add it to the

particle set. This method will quickly lower the variance it particles; a particle with a very high weight will be sampled many times. A better implementation is dependent on the particles already chosen. One method that does this is known as stratified random sampling, which breaks the particle set into strata and chooses a representative number of particles from each tier. This has the desirable effect of maintaining particle variance in the posterior distribution [4].

- **Add random particles after resampling** – The laws of probability work against the particle filter at the resample step. The probability of picking the least probable particle is non-zero, so if the filter is run long enough, eventually a case will arise where every particle in the resampled set is the worst estimate. In practice this has an infinitesimally low chance of occurring, especially with a large particle set, but it is non-zero nonetheless. A popular approach to mitigate this problem is to add random particles into the particle set after it has been resampled. This increases diversity it the particle set and makes the filter more robust [4].

After the resampling step is complete, the new particle set should match the target distribution. This particle set is then used as input to a subsequent iteration of the particle filter algorithm. Figure 5 depicts the algorithm graphically, while Figure 6 presents the pseudocode for the algorithm.

Figure 5: Pseudocode for the Particle Filter algorithm.

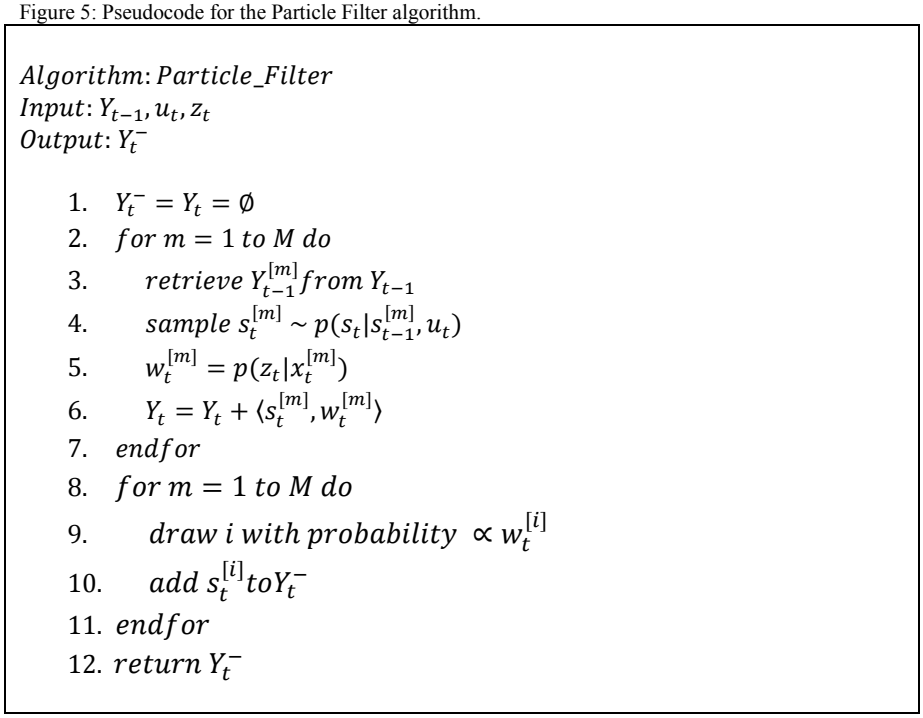
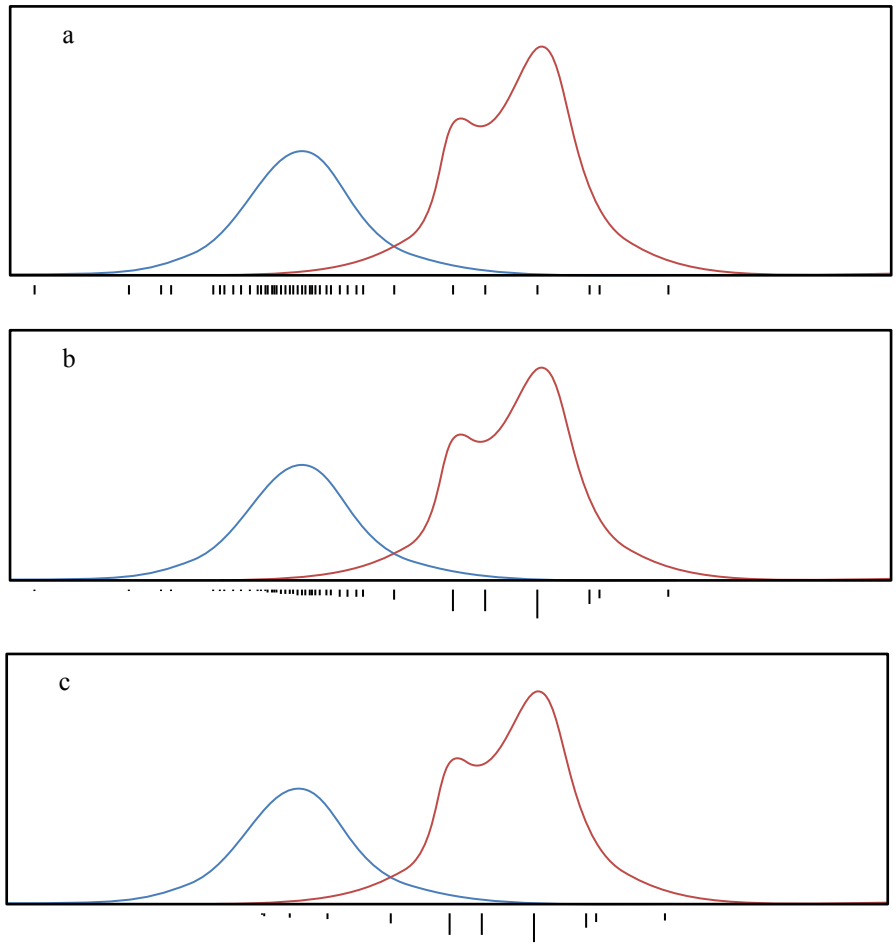


Figure 6: (a) The blue distribution to the left represent the proposal posterior. The red distribution to the right represents the target posterior. The black lines under the figure represent the particles, which are sampled from the blue distribution. (b) The particle set after weighting. The height of the lines is indicative of the weight of the particles. (c) The particle set after resampling. The particles with low weight were not added to the new set. Here, a single line may represent multiple particles, since we sample with replacement.



4.2. Complexity Analysis

Now we analyze the complexity of the Particle Filter. For all its flexibility, the Particle Filter has its drawbacks. We will show that one of these drawbacks is an expensive asymptotic time complexity.

We are interested in comparing the computational complexity of the Particle Filter to that of the Kalman Filter and the Extended Kalman Filter. Therefore, we have to express the computational complexity of the Particle Filter in terms of the size of the state, n. We know in the Kalman Filter, when we increase the state

space we just increase the dimension of s_t and the relevant covariances. We showed the complexity of this scales as $O(n^3)$.

The Particle Filter is not so simple. When we increase the state space, we have to increase the number of particles in order to properly approximate the posterior distribution. It turns out the number of particles needed to do this grows exponentially with the size of the state space, or $O(M^n)$.

To see this, consider a system with only two possible states, x=1 and x=2. The number of particles we need to represent this distribution is 2. Now, let us increase the size of this system in accommodate two more states: y=1 and y=2. Now we need four different particles to cover each state. At this point it should be obvious that this is a combinatorial problem, with a solution that grows exponentially as we add states.

4.3. Accuracy

Because of its sample-based nature, the Particle Filter is an approximate algorithm; the posterior it generates is only an estimate of the true distribution. However, unlike the Extended Kalman Filter, this error can be mitigated and the accuracy of the Particle Filter can be arbitrarily improved at the expense of performance.

The most obvious source of inaccuracy in the Particle Filter is the amount of particles. As the number of particles approaches infinity, the particle distribution approaches the true value of the posterior distribution. Therefore, if we want to make the filter more accurate, one easy way is to just increase the particle count. However, in practice, if the state space is small, around 100 particles will be sufficient. Of course, this is a floor, and it raises as the size of the state space increases [4].

Another source of inaccuracy stems from the difference between the proposal posterior distribution and the target posterior distribution. In Figure 6, this difference is exaggerated for illustrative purposes. If the actual distributions differed this much, the filter will very likely diverge, for the reason that only a few particles from the proposal distribution are highly weighted. To compensate for this we either increase the particle count or use a better proposal distribution.

The proposal comes from Equation 21, which we conditioned only on s_{t-1} and u_t. However, the target posterior we want to find is conditioned on z_t as well (this fact comes from Equation 2). Therefore, if we want a better proposal, we can incorporate the measurement in to this distribution.

5. Discussion and Conclusions

So far, we have presented three different Bayesian Filters: The Kalman Filter, the Extended Kalman Filter, and the Particle Filter. In this final section, we will compare the different filters and discuss their applicability in the context of robotics, but with implications for other fields.

The following table summarizes the major aspects of each algorithm.

	Kalman Filter	**Extended Kalman Filter**	**Particle Filter**
Input	$s_{t-1}, P_{t-1}, u_t, z_t$	$s_{t-1}, P_{t-1}, u_t, z_t$	Y_{t-1}, u_t, z_t
Output	s_t^-, P_t^-	s_t^-, P_t^-	Y_t^-
Assumption	Linear transition Linear observation Gaussian noise	Nonlinear transition Nonlinear observation Gaussian noise	Nonparametric
State representation	s_t, P_t	s_t, P_t	Y_t
Hypothesis	Single	Single	Multiple
Accuracy	Exact	Approximate, fixed by model	Approximate, but accuracy can be increasing by adding particles
Asymptotic Time Complexity	$O(n^{2.376})$	$\max(O(n^{2.376}, O(g(s_{t-1}, u_t)), O(h(s_t)))$	$O(M^n)$

5.1. An Example Application

In this paper, we introduced each filter in very broad terms. Now, we will use a practical scenario to motivate the application of each algorithm. Consider the problem of mobile robot localization. For this problem, the state vector we want to find is the location of the robot in some map. This quantity is unobservable, but through Bayes' Law we use observable quantities like the range to walls and the velocity of the motors to determine the state. The following three vectors are then the state, control and observation for this problem.

$$s_t = [x\ y\ \theta\ ]^T \quad \text{Equation 24}$$

$$u_t = [v\ \omega]^T \quad \text{Equation 25}$$

$$z_t = [\rho\ \phi]^T \quad \text{Equation 26}$$

Here, x and y are the Cartesian coordinates of the robot, while θ is its orientation; v and ω are respectively the linear and angular velocity commands sent to the robot; and ρ and ϕ are respectively the range and bearing to obstacles. We also have a transition model and observation model for this problem. These correspond to equations 14 and 15.

$$g(s_t, u_t) = \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} -\frac{\hat{v}}{\hat{\omega}} sin\theta + \frac{\hat{v}}{\hat{\omega}} sin(\theta + \hat{\omega}\Delta t) \\ \frac{\hat{v}}{\hat{\omega}} cos\theta + \frac{\hat{v}}{\hat{\omega}} cos(\theta + \hat{\omega}\Delta t) \\ \hat{\omega}\Delta t \end{bmatrix} \quad \text{Equation 27}$$

$$h(s_t) = \begin{bmatrix} \sqrt{(\mu_{j,x} - s_{t,x})^2 + (\mu_{j,y} - s_{t,y})^2} \\ atan2(\mu_{j,y} - s_{t,y}, \mu_{j,x} - s_{t,x}) \end{bmatrix} \quad \text{Equation 28}$$

Here, v and ω have been corrupted with noise, and μ_j is a feature in the map the robot observes. It is clear from Equations 27 and 28 that this system is nonlinear. Therefore, a Kalman Filter is not applicable to this problem; although, it would be appropriate for robot localization in one dimension. Unfortunately, this is the case for many problems in robotics and other fields, but of course we can move forward with the Extended Kalman Filter. To do this, we must calculate Equations 17 an 18 by taking the gradient of Equations 27 and 28.

$$G_t = \begin{bmatrix} 1 & 0 & -vsin(\theta)dt \\ 0 & 1 & vcos(\theta)dt \\ 0 & 0 & 1 \end{bmatrix} \quad \text{Equation 29}$$

$$H_t = \begin{bmatrix} \frac{-dx}{\sqrt{dx^2 + dy^2}} & -\frac{dy}{\sqrt{dx^2 + dy^2}} & 0 \\ \frac{dy}{dx^2 + dy^2} & -\frac{dx}{dx^2 + dy^2} & -1 \end{bmatrix} \quad \text{Equation 30}$$

Where dx is $\mu_{j,x} - s_{t,x}$ and dy is $\mu_{j,y} - s_{t,y}$.

With equations 24 through 30, we can now implement both the Extended Kalman Filter and the Particle Filter. Which one we choose to use depends largely on the environment we want to localize our robot in. For an indoor environment, the Particle Filter is preferable. The state space is smaller, and there are many corners to provide good landmarks. Further, the Particle Filter is able to solve what is known as the "Kidnapped Robot Problem." That is, given a fully localized robot, move it between Particle Filter iterations, and the Particle Filter with added random samples will re-localize the robot. The Extended Kalman Filter has no way of accomplishing this task. Outdoors, the state space is very large, so the complexity

of the Particle Filter is prohibitive. Here, we would rather use the Extended Kalman Filter. If we keep the number of tracked objects small, the computational complexity of this algorithm should not be overbearing.

5.2. Conclusions

In this paper we introduced three very influential algorithms in the field of Bayesian State Estimation. The first algorithm, the Kalman Filter, is one of the few algorithms researchers can call solved; it solves state estimation for linear Gaussian systems in an optimized manner. For this reason, even though the algorithm is over half a century old, it is still used extensively today.

The second algorithm was an extension of the Kalman Filter. It attempts to relax the requirements for a linear Gaussian system, so it is applicable to more systems. It does this by expanding the transition and observation models in a Taylor series approximation, and using their Jacobians to propagate covariances. The Extended Kalman Filter is no longer optimal, but the robustness of the original Kalman Filter allows for a great deal of latitude in the degree of nonlinearity of the model functions.

The final algorithm was the Particle Filter, a nonparametric Bayesian State Estimator. This algorithm differs from the previous two in that it has no requirements on the transition and observation models. It manages this because samples are taken from a proposal posterior, and then weighted according to observations in order to approximate the true posterior. The beauty in the Particle Filter is found in the resampling stage, where particles are chosen for the posterior according to their respective weights. Several factors affect the performance of the Particle Filter, including variance in the particle set, frequency of resampling, and how well the proposed posterior matches the target posterior. This flexibility comes at the price of high computational complexity in the size of the state space.

While these algorithms are among the most important in the field, they are certainly not the only ones. In fact, there are countless variations of these algorithms including the Information Filter, the Unscented Kalman Filter, and the Rao-Blackwellized Particle Filter. Each of these has their own take of the basic algorithms we presented here, in order to increase computational efficiency or accuracy. However, we expect to see the algorithms presented here in use for many years to come.

6. References

[1] Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. *Transaction of the ASME - Journal of Basic Engineering*, 35-45.

[2] Juler, S., & Uhlmann, J. (n.d.). A New Extension of the Kalman Filter to Nonlinear Systems. *Storage and Retrieval for Image and Video Databases.*

[3] Gordon, N., Salmond, D., & Smith, A. (1993). Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEEE Proceedings-F*, (pp. 107-113).

[4] Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic Robotics.* Cambridge, MA: MIT Press.