Comparison of Kalman and Particle Filters for Altitude Control of Aircraft

Noah J. Epstein
Department of Mechanical Engineering
Tufts University
Medford, MA 02155
noahjepstein@gmail.com

I. INTRODUCTION

Kalman filters are one of the most commonly used state estimators because of their effectiveness, ease of use, predictability, and applicability to many real-world systems. However, traditional Kalman filters assume a linear dynamic model and Gaussian noise inputs, which limits their applicability to systems without these constraints. Extended and unscented Kalman filter variants may be useful for systems which have some nonlinearities or non-Gaussian disturbances. For highly nonlinear or non-Gaussian systems, the particle filter can provide effective state estimation.

This paper attempts to provide a particle filter implementation that performs as well with respect to state-error as a Kalman filter in a longitudinal aircraft-control application. Both estimation techniques will be compared with respect to several performance characteristics, considering each technique's suitablity to longitudinal aircraft control.

While Kalman filters (especially when including the extended and unscented variety) are quite successful in aircraft control applications, I aim to provide a controller with similar reference tracking performance that uses a particle filter to estimate state. The technique could give a more flexible approach to state estimation with respect to model linearity and noise. I will compare the two methods with respect to the following:

1) Ability to minimize state error;
2) Computational performance (both usually are applied with real-time constraints);
3) Implementation difficulty

A. Kalman Filter Background

Kalman filters have a long history of successful use in aircraft guidance applications. The Kalman filter recursively estimates state with respect to the state estimate's difference from measured state, per equation 1.

$$\hat{\mathbf{x}}_{k+1} = \mathbf{F}\hat{\mathbf{x}}_k + \mathbf{G}\mathbf{u}_k + \mathbf{L}(\mathbf{y_k} - \mathbf{H}\hat{\mathbf{x}}_\mathbf{k}) \tag{1}$$

The Kalman filter gives matrix L such that the state estimator is optimal 1) with respect to the dynamic system model, process noise model, and sensor model and 2) for a linear system with Gaussian noise. While a general model of aircraft dynamics would involve nonlinear behavior, gain-scheduled approaches are often employed such that linear models can be used with the traditional Kalman filter. The ubiquity of Kalman filters for state-estimation in aircraft lead us to use the technique as a baseline for comparison with a particle filter approach.

B. Particle Filter Background

The particle filter is a sequential Monte-Carlo based estimation technique that is often used because of its flexibility with respect to nonlinear, non-Gaussian models. Rather than using some gain matrix to correct state estimates with respect to measurements, the particle filter propagates a distribution of particles though time which represent state by:

1) Prediction: Updating each particle with respect to model dynamics;

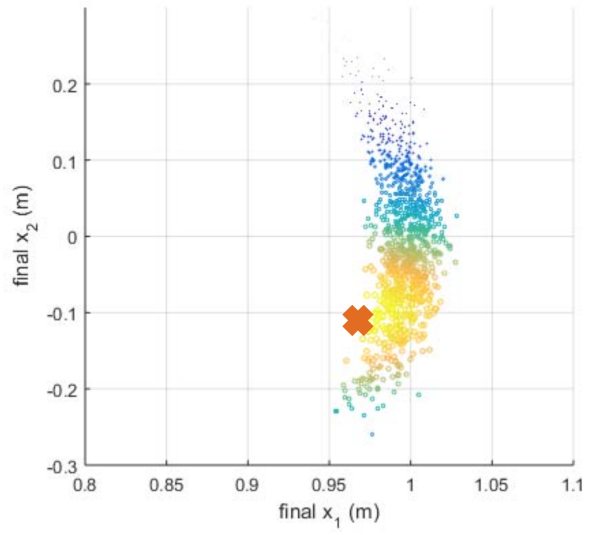

Fig. 1: An example distribution of particles in a particle filter, with color indicating a given particle's likelihood. [ME293 Course Materials, Jason Rife, 2016]

2) Correction: Finding a likelihood value for each particle by generating a probability of its position occurring with respect to a measurement;
3) Resampling: selecting with respect to likelihood some subset of the particles which will represent the state estimate.

Detail on the particle filter implementation used in this work is given in section V.

II. AIRCRAFT DYNAMIC MODEL

The system state will be descibed with vector $\mathbf{x}$ that contains horizontal and vertical velocity, denoted u and w, respectively, pitch, denoted θ, pitch rate, denoted $q = \dot{\theta}$, and altitude h. I will hereafter describe the system state at timestep k as:

$$\mathbf{x}_k = \begin{bmatrix} u \\ w \\ q \\ \theta \\ h \end{bmatrix}_k \tag{2}$$

where translational velocities are in units of *feet/sec*, pitch rate q in units of *rad/second*, pitch θ in units of *rad*, and altitude in units of feet. The system is sampled at a rate of 100Hz, giving $\Delta t = 0.01$ seconds.

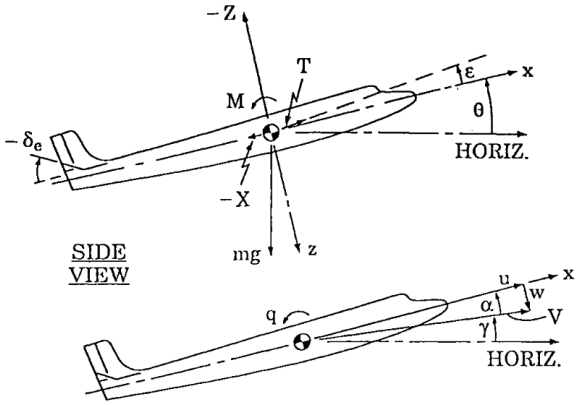

Fig. 2: Diagram showing aircraft state and control variables.
Top: elevator displacement δ_e, pitch θ.
Bottom: pitch rate q, velocity vectors u and v.
Source: [Bryson, 1994.]

The system's control inputs are described with $\mathbf{u}$ which contains δ_e and δ_t, the deviations of the elevators and throttle, respectively, from their nominal positions. This gives the control vector at timestep k as:

$$\mathbf{u}_k = \begin{bmatrix} \delta_e \\ \delta_t \end{bmatrix}_k \tag{3}$$

Using equations 2 and 3, the system is described with the state update equation

$$\mathbf{x}_{k+1} = \mathbf{F}\mathbf{x}_k + \mathbf{G}\mathbf{u}_k + \tilde{\mathbf{G}}\mathbf{q}_k \tag{4}$$

where $\mathbf{F}$ is the state transition matrix, $\mathbf{G}$ is the control input matrix, and the term $\tilde{\mathbf{G}}\mathbf{q}_k$ represents a process noise input. Matrices $\mathbf{F}$, $\mathbf{G}$, and $\tilde{\mathbf{G}}$ are populated using parameters of a dynamic longitudinal model of a Boeing 747 aircraft, given in [Bryson, 1994.].

I refer to the system's measured state using $\mathbf{y}_k$, per equation 5:

$$\mathbf{y}_k = \mathbf{H}\mathbf{x}_k + \mathbf{J}\mathbf{u}_k + \mathbf{r}_k \tag{5}$$

This model sets $\mathbf{J}$ to zero, and with the assumption that one can directly measure state, system output $\mathbf{y}_k = \mathbf{H}\mathbf{x}_k + \mathbf{r}_k$, where $\mathbf{H}$ is the identity matrix.

III. CLIMB-RATE CONTROL

Closed loop control was implemented using a command-hold type reference input to achieve a

constant climb rate $c = 10$ feet/second. At each timestep, the reference is updated as follows:

$$\mathbf{ref}_{k+1} = \begin{bmatrix} u_0 \\ w_0 \\ q_0 \\ \theta_0 \\ h + c\Delta t \end{bmatrix}_k \tag{6}$$

where Δt is the sampling rate of the discrete system. Using the reference update equation, the control input at timestep k is given as:

$$\mathbf{u}_k = \mathbf{K}(\hat{\mathbf{x}}_\mathbf{k} - \mathbf{ref}_\mathbf{k}) \tag{7}$$

where $\hat{x}_k$ is the estimated state determined either by the Kalman filter estimate or the particle filter estimate. The gain matrix $\mathbf{K}$ was determined using LQR methods, employing MATLAB's `lqr` command as discussed in [Epstein, 2016], my previous paper on LQR closed-loop longitudinal control of aircraft.

IV. KALMAN FILTER IMPLEMENTATION

The Kalman filter provides an optimal estimator for a linear system with Gaussian process and measurement noise. That is, it provides a gain matrix L which, given accurate parameters for a system's linear dynamic model, process noise covariance matrix $\mathbf{Q}$, and measurement noise covariance $\mathbf{R}$, minimizes the mean squared error of the state estimate with respect to the actual state.

This work uses MATLAB's `kalman` command to produce estimator gain matrix $\mathbf{L}$, which remains constant for the duration of the simulation. The `kalman` command uses the system with dynamics given by equation 4 and measurement dynamics from equation 5 to produce equation 1, the estimator update equation. Derivation of the Kalman filter was originally presented in [R.E. Kalman, 1960].

V. PARTICLE FILTER IMPLEMENTATION

The particle filter provides a state estimator that propagates a distribution of particles though time. This involves a prediction step, where the state of each particle i is propogated via the dynamics of the system as follows in equation 8.

$$\hat{\mathbf{x}}_{i,k} = \mathbf{F}\hat{\mathbf{x}}_{i,k-1} + \mathbf{G}\mathbf{u}_k - 1 + \tilde{\mathbf{G}}\mathbf{q}_k - 1 \tag{8}$$

I then use each particle estimate to "correct" our original measurement by finding the normalized probability of each particle $\mathbf{p}_i$ occurring given a sensor measurement $\mathbf{y}_k$, per equation 9:

$$\mathbf{p}_i = \frac{P(\hat{\mathbf{x}}_{i,k}|\mathbf{y}_k)}{\sum_{j=1}^{n} P(\hat{\mathbf{x}}_{j,k}|\mathbf{y}_k)} \tag{9}$$

where n is the total number of particles used in the simulation. (This work performs simulations with 100, 1000, and 10000 particles).

In this work, I use a weighted mean position to give a single state vector to generate control inputs. Each particle i that constitutes the mean is weighted with respect to the likelihood of each particle's occurrence, as follows:

$$\hat{\mathbf{x}}_k = \sum_{i=1}^{n} \mathbf{p}_i \hat{\mathbf{x}}_{i,k} \tag{10}$$

The result of equation 10 is used as an input to equation 7 for generating control inputs.

In an effort to keep our distribution of particles from changing significantly after many iterations, I perform a resampling step every twenty timesteps (every 0.2 seconds with $\Delta t = 0.01$ seconds). The resampling step selects a new sample of n particles from a CDF composed of the probability-weighted particles. The CDF is created by feeding the output of MATLAB's `cumsum` command to `interp1`. As such, in the resampled distribution, each sample is of equal weight.

VI. NOISE MODEL

To accurately compare the two state estimation techniques, they were used in simulation side-by-side with process and sensor noise inputs that matched at each time step. First, MATLAB's `mvnrnd` command was used to generate a random Gaussian noise vector with zero mean and covariance matrix $\mathbf{Q}$ for each particle, shown in equation 11.

$$\mathbf{q}_{particle,k} = [q_1 ... q_n]_k \tag{11}$$

where $\mathbf{q}_{\mathbf{i},\mathbf{k}}$ is a random sample of the normal distribution shown in equation 12:

$$\mathbf{q}_{\mathbf{i},\mathbf{k}} = \mathbf{N}(\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T, \mathbf{Q}) \tag{12}$$

The noise vectors for each of n particles are combined to give a single noise value to the Kalman filter state update equation. Each noise sample constituted a weighted mean as follows:

$$\mathbf{q}_{kal,k} = \Sigma_{i=1}^{n} \mathbf{q}_{i,k} \mathbf{p}_{i,k} \tag{13}$$

where $\mathbf{p}_{i,k}$ represents the normalized probability of that particle state occurring given the current measurement state, per equation 9. The process noise covariance matrix $\mathbf{Q}$ is as follows:

$$\mathbf{Q} = \begin{bmatrix} 25.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 25.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.016 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 3.03e-6 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{14}$$

The horizontal and vertical velocity variance components of the process noise model are equal and on the order of 5 feet/second per discussion on gust disturbances in [Bryson, 1994.] and [Ly, 1980]. Small pitch disturbances on the order of $\sigma_q^2 = 2^2 \deg^2/s^2$ and $\sigma_\theta^2 = 0.05^2 \deg^2$ degrees were assumed. Altitude variance of 1 ft^2 was For this work, disturbances were assumed to be uncorrelated.

A single measurement noise sample is then taken which assumes zero mean, Gaussian measurement noise and covariance given by matrix $\mathbf{R}$:

$$\mathbf{R} = \begin{bmatrix} 7.0^2 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.1^2 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.349^2 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.349^2 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 30.0^2 \end{bmatrix} \tag{15}$$

$\mathbf{R}$ is a diagonal matrix with each element corresponding to an estimate of each state's sensor variance. It's reasonable to assume the variances are uncorrelated (even though process noise may not be) because each sensor would likely be implemented in hardware as a single unit which responds uniquely to noise. Both velocity components are assumed to have a sensor variance of 1% of their nominal value, angular components are assumed to have a variance of one degree with respect to their nominal values, and the altitude sensor is estimated to have variance of 30^2 feet.

Measurement noise generated with MATLAB's `mvrnd` is simply added to the measurement equation as a single vector because a "particle cloud" of measurements is not used. There is only have a single measurement $\mathbf{y_k}$, dependent on the actual system state $\mathbf{x_k}$ and process noise $\mathbf{r_k}$.

VII. RESULTS

A. Reference Tracking Comparison

Figure 3 gives an overview of the altitude-behavior of the aircraft for a 100-second simulation using Kalman and Particle Filters in-step, where the particle filter used 1000 particles. The simulation included an initial altitude offset where both estimator initial condition vectors were offset by 100ft of altitude with respect to the actual altitude initial condition.

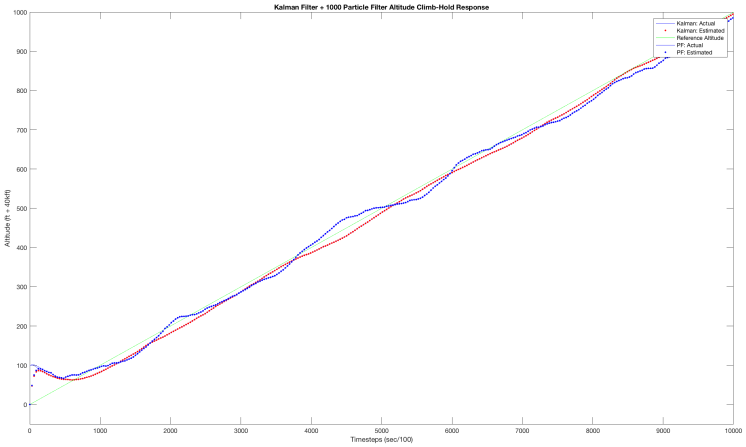

Fig. 3: shows the altitude reference track (green), Kalman filter actual and estimated (red) states, and particle filter actual and estimated (blue) states.

One can see that there is some slightly higher degree of variability in the particle filter estimate. A subsequent simulation shown in figure 4 over a shorter timeframe using 10000 particles gives similar results with a slightly tighter coupling:

For both plots and both estimator types, one can see that the estimated value takes 5-6 seconds to converge to the actual state value. One notable difference between the two estimator techniques with this initial condition offset is that the particle filter may be less likely to converge under significant error in initial conditions.

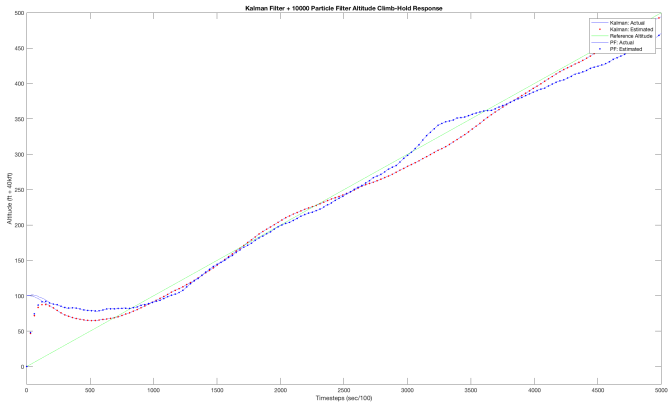

Fig. 4: shows the altitude reference track (green), Kalman filter actual and estimated (red) states, and particle filter actual and estimated (blue) states.

The Kalman filter will "steer" the estimate towards the actual value linearly per correction term $\mathbf{L}(\mathbf{y_k} - \mathbf{H}\hat{\mathbf{x}}_\mathbf{k})$. The particle filter generates probabilities on each particle given the state measurement and covariance, so if the state and state estimate differ too much, many particles may have their likelihoods go to zero and thus no longer influence the state estimate.

I found that altitude deviations beyond 100ft caused the particle filter to become unstable in just one or two timesteps, reducing the number of particles involved in estimation until the sum of probabilities for every particle was zero. This leads me to be cautious in application of such a filter, restricting use to those cases where estimates are likely to be close to their real values.

B. Error Analysis

To evaluate the accuracy of the two types of filters over the course of the simulation, one can first evaluate the deviation of the estimator from actual state values, as well as the reference that I aim to track.

I first show the mean-squared-error between the actual and estimated altitude state in table 5. Upon inspection of table 5 one notices that the difference in altitude mean-squared error between the Kalman filter and the particle filter is fairly small. The worst case (100 particles) has a MSE difference of 0.089 ft^2. This is a 12% increase in MSE between an estimator that is optimal by definition, which I consider to be fairly good. A simulation with 1000 particles gives a 4.8% gain and 10000 particles gives 2.8% MSE change. I consider this close enough for the particle filter to be a satisfactory estimator, especially with respect to an optimal estimator.

No. particles	Kalman (ft^2)	Particle Filter (ft^2)
100		4.816
1000	4.727	4.781
10000		4.747

Fig. 5: The table above gives mean-squared error (in units of ft-squared) between the estimated altitude state and actual altitude for a simulation of 100 seconds. Values are averaged over five simulations.

No. particles	Kalman (ft^2)	Particle Filter (ft^2)
100		720
1000	399	522
10000		490

Fig. 6: The table above gives mean-squared error between the actual altitude and reference altitude (averaged over five 100-second simulations).

Figure 6 shows mean-squared-error between the actual altitude and reference altitude, which I take as a metric for the overall effectiveness of the controller. Referring back to 1 shows that both estimators give an altitude slightly below that of the reference, so having summed errors on the order of hundreds of feet is unsurprising over 100 seconds. One does notice the striking change in MSE when varying the number of particles. Even the simulation with 10000 particles gives about a 22% greater MSE. This leads us to conclude that the Kalman filter is significantly more effective for tracking a climb-hold reference.

C. Computational Performance

Below, in Figure 7 I present average runtimes for three trials of 100-second simulations of the aircraft system. These times were calculated using MATLAB's `tic-toc` command, immediately before and after the main loop of the simulation to exclude any setup overhead.

No. particles	Kalman (s)	Particle Filter (s)
100	0.49 sec	1.69
1000		5.26
10000		28.16

Fig. 7: The table above gives mean run-times in seconds for each simulation type.

One can see that the Kalman filter takes less than 3x the time to simulate than even the 100-particle filter does. As particles are added (which can help increase the accuracy of the simulation) the computation time increases significantly. For this implementation, the particle filter is certainly a more computationally intensive approach.

D. Implementation Difficulty

While it only provides a very subjective idea of the complexity of each respective estimator: a somewhat verbose implementation of the Kalman filter for this dynamic model consists of 90 lines of code with comments and blank lines removed, and a similarly implemented particle filter consists of 225 lines. Further, once the Kalman estimator matrix $\mathbf{L}$ is found, estimation consists of a single update equation: most of the work (finding $\mathbf{L}$) is done once, outside of the simulation loop. However, for a particle filter, resampling and roughening are done in the loop, with both steps consisting of a number of substeps, each conducted for every step of the simulation. While the above provides a non-quantitative measure of the complexity of each filter's implementation, a particle filter implementation must consist of many more steps. This difference does constitute a criterion for selection of an estimator for some engineering application. One would always prefer a simpler solution to achieve a given result.

VIII. CONCLUSIONS AND FUTURE WORK

It should come as no surprise that I prefer a Kalman filter to a particle filter in this particular application. By inspection, a linearized aircraft model perturbed by Gaussian disturbance and measurement noise doesn't require the specific advantages of a particle filter. The Kalman filter provided:

1) Superior state estimation and thus superior reference tracking
2) Lower computational cost
3) A simpler controller implementation that introduces less propensity for error
4) Introduces less randomness into state estimates

This does not, however, imply that a Kalman filter would still be superior for other applications or using a more complex aircraft model. Future works could include similar state estimator comparisons using systems that are more nonlinear and have non-Gaussian noise models. Further work could also use particle filters with roughening techniques to reduce or control the amount of randomness that the filter introduces to a control system.

REFERENCES

[Bryson, 1994.] Arthur E. Bryson Jr. *Control of Longitudinal Motions of Aircraft* Control of Spacecraft and Aircraft, Princeton Press, 1994.

[Anderson and Moore, 1990] Brian Anderson and John Moore. *Optimal Control: Linear Quadratic Methods.* Prentice Hall, 1990.

[Ogata, 2005] Katsuhiko Ogata. *Modern Control Engineering*. Prentice Hall, 2009.

[MATLAB, 2016] The Mathworks Inc. *Control Systems Toolbox Documentation*. Accessed: October, 2016.

[University of Michigan, 2016] The University of Michigan. *Control Tutorials for MATLAB and Simulink*. Accessed: October, 2016.

[Epstein, 2016] Noah Epstein, Jason Rife. *LQR Command-Hold Control of Aircraft to Achieve Constant Rate of Ascent*. ME293 Project 1, October 2016.

[Ly, 1980] Ly, U. and Y. Chan. "Time-Domain Computation of Aircraft Gust Covariance Matrices," AIAA Paper 80-1615, presented at the 6th Atmospheric Flight Mechanics Conference, Danvers, Massachusetts, August 1980.

[R.E. Kalman, 1960] R.E. Kalman (1960). "A New Approach to Linear Filtering and Prediction Problems" (PDF). Journal of Basic Engineering: 3545.

APPENDIX

A MATLAB script to produce the above results can be accessed at: `https://goo.gl/BJCKHy`