On the Kalman Filter and Its Variations

By

Theodore S. Lindsey

Submitted to the Department of Mathematics
and the Graduate Faculty of the University of Kansas
in partial fulfillment of the requirements for the degree of
Master of Arts

Prof. Bozenna Pasik-Duncan, Chair

Prof. Tyrone Duncan

Prof. Xuemin Tu

Date defended: April 18, 2014

The Thesis Committee for Theodore S. Lindsey certifies
that this is the approved version of the following thesis:

On the Kalman Filter and Its Variations

Prof. Bozenna Pasik-Duncan, Chair

Date approved: April 23, 2014

Abstract

The objective of this paper is to explore the standard Kalman filter and two non-linear variations. Additionally, we will discuss the derivation of the Kalman filter using Newton's method. Next we will consider the implementation of both the Extended Kalman filter and the Unscented Kalman filter, paying special attention to the cases where the Unscented Kalman filter performs better than the Extended Kalman filter. Finally, we will make a comparison between these two Kalman filter variations and consider a few other modifications to the standard Kalman filter.

Acknowledgements

I would first like to thank my advisor, Prof. Bozenna Pasik-Duncan, for guiding and supporting me through this process.

I wish to thank Prof. Tyrone Duncan and Prof. Xuemin Tu for serving on my Thesis Committee.

I would also like to thank the KU Mathematics faculty as a whole for providing instruction and advice.

I would like to recognize Yi Cao for his assistance with the implementation of the Extended Kalman filter and the Unscented Kalman filter in MATLAB.

I wish to thank my undergraduate advisor Andrew Parker for his inspiration and mentorship as I worked on my bachelor's degree and since then.

I would like to thank Mark Yannotta for inspiring me to study mathematics when I was uncertain of my academic interests and attempting to find an area of study for a 4-year degree.

Finally, I would like to thank my mother and father for their encouragement and support of my education; especially my mother, who originally encouraged me to approach the world with curiosity and wonder through her teaching. Also, I would like to thank my wife who supported me through the entire process, and who's encouragements made this thesis possible.

Contents

0.1 Introduction - Filtering

The essence of a filter is to determine the states of a system (say, x_k) which are not directly observed by forming estimates of the states based on outputs from the system (say, z_k) which are effected by the states of the system (x_k). Generally, there are three goals of filtering. When working towards first goal, we want to know information about the state of a system (eg velocity or position). In the second, we want to control a system described by a state space model in which case we use state feedback controls which take the form $u_k = u(k, x_k)$ (often, x_k isn't known, so system state estimates are used for this, too). When working towards the third goal, we want to replace the state space model with an equivalent model, such as an ARMAX system or an innovations-from state-space model. [4, p.100,127]

When approaching the filtering problem, we aim to estimate the value of a random variable Y_0 given a set of random variables $Y_1, \ldots, Y_n$ who's values we have observed. We use $\vec{Y}^T = \langle Y_0, \ldots, Y_n \rangle$ to represent a vector random variable with a particular joint distribution. $\vec{Y}^T = \langle Y_1, \ldots, Y_n \rangle$ represents the observed vector. When evaluating our filter, we want to minimize the mean squared error between our estimator $g(Y)$ and the random variable Y_0: $\mathbb{E}\left[Y_0 - g(Y)\right]^2$. According to Davis, Vinter [4], the mean squared error is minimized by $g(Y) = \mathbb{E}\left[Y_0|Y\right] = \int_{-\infty}^{\infty} y_0 dF_{Y_0|Y}\left(y_0|Y\right)$. Unfortunately, this isn't easy to work with, and will likely cause issues when attempting to compute it, so instead, we consider the linear estimation given by $g(Y) = \alpha_1 Y_1 + \alpha_2 Y_2 + \cdots + \alpha_n Y_n$. Now, $\mathbb{E}\left[Y_0 - g(Y)\right]^2 = \mathbb{E}\sum_{i,j=0}^{n} \alpha_i \alpha_j Y_i Y_j = \sum_{i,j}^{n} \alpha_i \alpha_j \mathbb{E}\left[Y_i Y_j\right]$ (we define $\alpha_0 = -1$). We can think of this estimation as a projection of our random variable Y_0 onto our observations. [4, p.100-101]

Also of note is that, as we go through this process, we would like to record a history of our system states X_k. In order to accomplish this, we will approach filtering with a recursive algorithm which will consider the system state X_k and observation Y_{k+1} and estimate of the system state, X_{k+1}.

Unfortunately, we run into a problem if we are at time n and we would like to estimate the random variable Y_0 at time n (we'll call this $Y_{0,n}$). We'll use $Y_1, \ldots, Y_n$ to come up with

an estimate. The problem is that in order to achieve this estimation, we end up needing to invert increasingly larger $(n \times n)$ covariance matrices (designated by $P_n = \mathrm{cov}(Y_1, \ldots, Y_n)$) due to the fact that we want a historical record of past system states (X_1 through X_n). In order to approach this in a recursive manner rather than by referencing every previous Y_k, we can, in the simplest case, use the recursion $Y_{o,n+1} = a_n Y_{0,n} + b_n Y_{n+1}$, that is, the estimate in question is a linear combination of the current measurements with the previous estimate (this is the idea upon which the Kalman filter is based). [4, p.112]

0.2 Introduction - Kalman Filter

The Kalman filter is a method by which we can estimate unknown system states from noisy (indirect, inaccurate, or uncertain) observations. The Kalman filter is the optimal estimator for gaussian noise. [5]

Many variations of the Kalman filter are used. Each addresses particular problems that arise in the implementation of the standard Kalman filter when applied to particular types of problems. Many of the variations are vulnerable to certain situations as well. In particular, we will focus on the standard Kalman filter, the Extended Kalman filter, the Unscented Kalman filter, and the derivation of the Kalman filter via Newton's method. We will then look at implementations of a couple of these variations and then discuss the best applications and weaknesses for each variation.

0.3 Standard Kalman Filter

Consider the system

$$x_{k+1} = A(k)x_k + B(k)u_k + C(k)w_k$$

and

$$y_k = H(k)x_k + G(k)w_k$$

with "$\{w_k\}$, an l-vector white-noise process with unit covariance ($Ew_k w_k^T = I_l$) and the initial random variable x_0 is uncorrelated with w_k, with known mean and covariance m_0, P_0, respectively. The coefficient matrices $A(k)$, etc, may be time-varying, as indicated by their dependence on k." [4, p.117]

Then, for the above system and assumptions, "the estimator $\hat{x}_{k|k-1}$ {the estimate of the current system state given previous information} satisfies the recursive equation

$$\hat{x}_{k+1|k} = A(k)\hat{x}_{k|k-1} + B(k)u_k + K(k)\left[y_k - H(k)\hat{x}_{k|k-1}\right]$$

for $k = 0, 1, \dots$

$$\hat{x}_{0|-1} = m_0$$

The $n \times r$ gain matrix $K(k)$ is given by

$$K(k) = \left[A(k)P(k)H^T(k) + C(k)G^T(k)\right]\left[H(k)P(k)H^T(k) + G(k)G^T(k)\right]^{-1}$$

where $P(k)$ is the error covariance $P(k) = \mathbb{E}\left[(x_k - \hat{x}_{k|k-1})(x_k - \hat{x}_{k|k-1})^T\right]$, $P(k)$ satisfies the recursive Riccati equation

$$\begin{aligned} P(k+1) = {} & A(k)P(k)A^T(k) + C(k)C^T(k) - \left[A(k)P(k)H^T(k) + C(k)G^T(k)\right] \\ & \left[H(k)P(k)H^T(k) + G(k)G^T(k)\right]^{-1} \cdot \left[A(k)P(k)H^T(k) + C(k)G^T(k)\right]^T \\ P(0) = {} & P_0 \end{aligned}$$

The innovations process $v_k := y_k - H(k)\hat{x}_{k|k-1}$ is a wide-sense white-noise {a sequence of random variables with mean zero and finite variance} process with covariance function $\mathbb{E}\left[v_k v_j^T\right] = \left[H(k)P(k)H^T(k) + G(k)G^T(k)\right]\delta_{kj}$. If in addition to the above assumptions $(x_0, w_0, w_1, \dots)$ are jointly normally distributed, so that in particular $\{w_k\}$ is a Gaussian white-noise process, then $\hat{x}_{k|k-1} = \mathbb{E}\left[x_k | y_{k-1}\right]$." [4, p.118]

The Kalman filter operates in a recursive fashion in an effort to estimate certain states

of a system, given noisy measurements. It does this by minimizing the mean-squared error of the current state estimate. The Kalman filter compares the estimate based on previous states to the noisy measurement-based state information in an attempt to correct errors in previous estimates and to mitigate noise in current observations. For non-gaussian noise (but with a given mean and standard deviation), the Kalman filter is the best linear estimator. [7, p.4]

The Kalman filter takes previous estimates along with the current observations as the inputs. The recursion operates in a two-step process: the prediction step and the correction step. First, the Kalman filter performs a "time update" (the prediction step) by taking estimated knowledge of the current state, say X_k and, using the physical model of the system, it predicts the future system state, giving $X_{k+1|k} = F_k X_k + G_k u_k$ (where F_k is the state transition model, X_k is the "true" current system state, G_k is the input matrix, and u_k is the input control). With the state prediction, the Kalman filter then predicts the expected measurement reading, giving $Z_{k+1|k} = H_k X_{k+1|k}$. From here, the Kalman filter then performs a "measurement update" (the correction step) in which it compares the predicted measurement $Z_{k+1|k}$ to the actual measurements, Z_{k+1} which gives us the measurement residual, $V_{k+1} = Z_{k+1} - Z_{k+1|k}$. Next, the measurement residual is used to find the new corrected measurement-based state estimate, $X_{k+1} = X_{k+1|k} + W_{k+1} v_k$ (where W_{k+1} is the Kalman gain, v_{k+1} is the residual), which we accept as the new "true" system state and then repeat the process. The process in which we find the Kalman gain is the more complicated portion of this step. First, we calculate the state estimate (or prediction) covariance, $P_{k+1|k} = F_k P_{k|k} F'_{k+1} + Q_{k+1}$ (where $P_{k|k}$ is the previous state covariance and Q_{k+1} is the process noise covariance). Using the State estimate covariance, we then calcuate the measurement prediction covariance, $S_{k+1} = H_{k+1} P_{k+1|k} H'_{k+1} + R_{k+1}$ (where R_{k+1} is the measurement noise covariance). Finally, we find the Kalman gain, $W_{k+1} = P_{k+1|k} H'_{k+1} S^{-1}_{k+1}$. At this point, the "true" system state can be calculated by the previous formula, however, before moving on to the next iteration of the Kalman filter, we will perform one final calculation: we will find the updated state covariance, $P_{k+1|k+1} = P_{k+1|k} - W_{k+1} S_{k+1} W'_{k+1}$.

The Kalman filter is most commonly found in use in guidance systems such as in

aircraft, watercraft, and spacecraft. In these cases, the Kalman filter will first take data from sensors, then attempt to filter out the noise and overcome possibly overcome insufficient information about the system provided by the measurements, and finally, estimate the position, velocity, etc. An example of the Kalman filter in action is as follows. Imagine we want to determine the true position of a vehicle as it moves. In order to do so, we only have a few sensors to work with. First, we have a GPS sensor, which is not accurate on a local scale, but can give us excellent position information relative to the size of the globe. Next, we have a compass which can tell us our direction of travel, but isn't extremely precise. Last, we have the speedometer of the vehicle, which tells us the speed. In the first step, the time update step (or prediction step), the Kalman filter takes the knowledge of our current "true" location and, with a standard Newtonian motion model, it predicts the future location of the vehicle, $X_{k+1|k}$, given the currently predicted speed (from speedometer), position (from GPS), heading (from compass), etc. From there, we also predict the current sensor readings, $Z_{k+1|k}$ based on the physics model and the current state. Next, in the measurement correction (or update) step, the Kalman filter calculates the measurement residual - the difference between predicted sensor readings and actual (but noisy) sensor readings. Using the measurement residual, the Newtonian motion model, and state covariance, we then calculate the current "true" position.

0.4 Extended Kalman filter

The basic Kalman filter is limited to linear systems. In order to estimate the states of non-linear systems, we must turn to variations of the Kalman filter. In particular, the Extended Kalman filter and the Unscented Kalman filter are of particular usefulness when working with non-linear systems [5, p.811][3, p.108]. Both the Extended Kalman filter and the Unscented Kalman filter allow us to work with non-linear systems, however, the Unscented Kalman filter will improve upon several flaws in the Extended Kalman filter, as observed by Wan [12].

The Extended Kalman filter is, unlike the standard Kalman filter, a generally biased estimator of a non-linear system. Moreover, the Extended Kalman filter works to estimate the system by performing a first-order linearization on the system, for which it is the best linear unbiased estimator. [5]

The Extended Kalman filter takes the non-linear system and linearizes the system in the vicinity of the previous state estimate. [3, p.108]

As in the standard Kalman filter, the Extended Kalman filter is a recursive process, but with an additional step. Each iteration, the Extended Kalman filter first linearizes the system dynamics in $X_{k+1} = f(X_k) + w_k$ around the previous state estimate $X_{k|k-1}$. We take the observations $Z_{k+1} = h(X_{k+1}) + v_{k+1}$. Next, the Extended Kalman filter applies the prediction step of the filter to the just-calculated linearized system dynamics to obtain $X_{k+1|k} = f(X_k)$ and $P_{k+1|k} = J_f(X_k) P_k J_f^T(X_k) + Q_k$ (state prediction and covariance, with J_f as the Jacobian of f). Then, the Extended Kalman filter linearizes the observation dynamical system $y_k = h(X_k) + v_k$ in the vicinity of $X_{k+1|k}$. Finally, the filter updates the linearized system state, obtaining $X_{k+1|k+1}$ and $P_{k+1|k+1}$. [10, p.2] [9, p.34]

0.5 Unscented Kalman Filter

In order to improve upon the flaw mentioned previously, in which the Extended Kalman filter has the potential to propagate error through it's linearization of the non-linear system, the Unscented Kalman filter instead uses deterministic sampling and can achieve 3rd order accuracy compared to the Extended Kalman filter's 1st order accuracy. Additionally, the Unscented Kalman filter is able to perform the estimation with an algorithm that is within the same order of complexity as the Extended Kalman filter.

As in the Extended Kalman filter, the Unscented Kalman filter represents the state distribution with a Gaussian random variable, but instead uses a minimal set of sample points which accurately represent the mean and covariance of the Gaussian random variable. When the Unscented Kalman filter propagates this Gaussian random variable through the system, the sample points also accurately represent the posterior mean and

covariance (to the 3rd order), regardless of the non-linearity. [12, p.2]

The principle of the unscented Kalman filter lies in the unscented transformation, which is a technique for calculating statistics of a random variable which is transformed in a non-linear manner. The unscented transformation method is preferable to other methods such as the Monte-Carlo method because it requires far fewer sample points to provide an accurate propagation. This is largely due to the assumption that we are working with a Gaussian random variable. For non-Gaussian random variables, the performance drops to a minimum of second order accuracy.

If we start with the general concept of $\hat{x}_k = (\text{prediction of } x_k + \mathcal{K}_k\,[y_k - \text{prediction of } y_k])$ and apply the unscented transformation, the unscented Kalman filter is a direct extension. We now consider the state random variable to be the joining of the original system's state and noise. Of particular note is that it is not necessary to explicitly calculate Jacobians or Hessians in the process of estimating the system state. [12]

Put another way, "The Unscented Kalman filter is founded on the intuition that it is easier to approximate a probability distribution that it is to approximate an arbitrary nonlinear function or transformation." [11]

0.6 Kalman Filter via Newton's method

In addition to it's standard derivation, the standard Kalman filter can be derived using Newton's root finding method. This derivation uses three main steps: linear estimation, weighted least squares, and newton method in a recursive least squares approach. [6]

Following the process outlined in Humpherys and West (2009), we start with the linear system

$$b = Ax + \epsilon$$

where b is our set of (imprecise) measurements, x is the set of system states we are attempting to estimate, ϵ is the errors in the measurement (mean zero, covariance Q, Q is positive definite), and A is a known matrix ($m \times n$, rank n). Because we want our estimator of x ($\hat{x}$) to be unbiased, we need for $\mathbb{E}\,[\hat{x}] = x$. Also, since the estimator is

linear, we know that $\hat{x} = Kb$ for some matrix K. With a few substitutions, we see that

$$\mathbb{E}[\hat{x}] = \mathbb{E}[Kb] = \mathbb{E}[K(Ax + \epsilon)] = KAx + K\mathbb{E}[\epsilon] = KAx.$$

In order for $\hat{x}$ to be unbiased, then, it is necessary that $KA = I$. Since A is already known, we need to find a K which satisfies this. Also of note is that, since we are looking for the best estimator $\hat{x}$ of x, we are attempting to minimize $\mathbb{E}[\|\hat{x} - x\|^2]$ by our choice of K. According to Humpherys and West [6, p.3], the K which satisfies both criteria is $K = (A^T Q^{-1} A)^{-1} A^T Q^{-1}$ (recall that A is the known $m \times n$ matrix, Q is the covariance of our measurement error). Then, $\hat{x} = Kb = (A^T Q^{-1} A)^{-1} A^T Q^{-1} b$ (recall that $b = Ax$ plus some error). Swapping b out, we have $\hat{x} = (A^T Q^{-1} A)^{-1} A^T Q^{-1} (Ax + \epsilon) = x + (A^T Q^{-1} A)^{-1} A^T Q^{-1} \epsilon$. The covariance is given by $\mathbb{E}\left[(\hat{x} - x)(\hat{x} - x)^T\right]$
$= (A^T Q^{-1} A)^{-1} A^T Q^{-1} \mathbb{E}[\epsilon \epsilon^T] Q^{-1} A (A^T Q^{-1} A)^{-1} = (A^T Q^{-1} A)^{-1}$.

Newton's root finding method takes a smooth function f which at some point near x_0 has a root ($f(\overline{x}) = 0$ for some $\overline{x}$ sufficiently close to x_0). We require that $Df(x)$ be non-singular, then we get the recursive process

$$x_{n+1} = x_n - (Df(x_n))^{-1} f(x_n)$$

and $\lim_{n \to \infty} x_n = \overline{x}$ at a quadratic rate.

The point of newton's method is that if we call an particular objective function, say $J(x)$, then if we find the root of the derivative ($f(x) = \nabla J = 0$), we will find the local extreme values. So, using newton's method, we consider

$$x_{n+1} = x_n - D^2 J(x_n)^{-1} \nabla J(x_n)$$

which converges to $\hat{x}$ (which minimizes the weighted least squares problem) when we have a sufficiently close initial value x_0. It actually turns out that newton's method is unnecessarily powerful in terms of it's rate of convergence, so instead, we will simplify a few of the computations by instead solving the normal equations ($A^T W A \hat{x} = A^T W b$)

directly. If we did use newton's method, the above expression of x_{n+1} would converge to $\hat{x}$. In any case, we the have

$$\hat{x} = x - \left(D^2 J\right)^{-1} \left(A^T W A x - A^T W b\right).$$

Something we have not yet taken into account, though, is that observations of our system are constantly incoming, as opposed to just looking at a single system state and the corresponding observations. Again, we want to solve the least squares problem, but now we introduce a recursive process in order to deal with the steady stream of observation data. We would like to find the best unbiased linear estimate $\hat{x}_k$ of x according to $\beta_k = \mathcal{A}_k x + \epsilon_k$ with $\beta_k^T = [b1, b2, \ldots, b_k]$, $\mathcal{A}_k^T = [A_1, A_2, \ldots, A_k]$, and noise term $\epsilon_k = [v_1, v_2, \ldots, v_k]$ with each v_k having a mean of zero and all uncorrelated and $\mathrm{Cov}[v_j] = R_i > 0$. Also, $\mathcal{A}_k$ is full column rank. Then we get the estimate $\hat{x} = \max_x \|\beta_k - \mathcal{A}_k x\|^2_{\mathcal{W}_k}$ with $\mathcal{W}_k = \mathrm{Cov}[\epsilon_k]^{-1} = \mathrm{diag}\left(R_1^{-1}, \ldots, R_k^{-1}\right)$. As time progresses, the system grows, and the least squares solution changes. We can, however, use our previous estimate $\hat{x}_{k-1}$ to calculate the current estimate $\hat{x}_k$. We can rewrite our estimate element-by-element as $J_k(x) = \frac{1}{2}\sum_{i=1}^{k} \|b_i - A_i x\|^2_{R_i^{-1}}$. The sum portion can be split into $J_k(x) = J_{k-1}(x) + \frac{1}{2}\|b_k - A_k x\|^2_{R_k^{-1}}$. After diferentiating and simplifying, we see that this gives the recursive formula $\hat{x}_k = \hat{x}_{k-1} - \left(D^2 J_k\right)^{-1} A_k^T R_k^{-1} \left(A_k \hat{x}_{k-1} - \beta_k\right)$. Simplifying the above (since $\nabla J_{k-1}(\hat{x}_{k-1}) = 0$, and choosing $x = \hat{x}_{k-1}$), we have $\hat{x}_k = \hat{x}_{k-1} - K_k A_k^T R_k^{-1} \left(A_k \hat{x}_{k-1} - \beta_k\right)$ (with $K_k = \left(D^2 J_k\right)^{-1}$, covariance of the estimate). But, $K_k^{-1} = K_{k-1}^{-1} + A_k^T R_k^{-1} A_k$, so $K_k = \left(K_{k-1}^{-1} + A_k^T R_k^{-1} A_k\right)^{-1} = K_{k-1} - K_{k-1} A_k^T \left(W_k^{-1} + A_k K_{k-1} A_k^T\right)^{-1} A_k K_{k-1}$. THe recursive least squares method, then, gives us

$$K_k = K_{k-1} - K_{k-1} A_k^T \left(R_k + A_k K_{k-1} A_k^T\right)^{-1} A_k K_{k-1}$$
$$\hat{x}_k = \hat{x}_{k-1} - K_k A_k^T R_k^{-1} \left(A_k \hat{x}_{k-1} - b_k\right)$$

Putting this all together, we have

$$K_k = \left[\left(Q_{k-1} + F_{k-1} K_{k-1} F_{k-1}^T \right)^{-1} + H_k^T R_k^{-1} H_k \right]^{-1}$$
$$\hat{x}_k = F_{k-1}\hat{x}_{k-1} + G_{k-1}u_{k-1} - H_k H_k^T R_k^{-1} \left[H_k \left(F_{k-1}\hat{x}_{k-1} + G_{k-1}u_{k-1} \right) - y_k \right]$$

[6, p.7]

0.7 Implementation and Comparison of Two Kalman Filter Extensions

Here, we will compare the effectiveness of the Extended Kalman filter and the Unscented Kalman filter for the same arbitrarily chosen non-linear system ($f(X) = x_2; x_3; 0.05 * x_1 * (x_2 + x_3)$, $h(x) = x_1$). In particular, we will compare the computational time for each filter, and be alert to errors introduced by the Extended Kalman filter estimate that are not present when using the Unscented Kalman filter estimate and vice versa.

In this particular simulation, we will consider a three system state and we will set the noise as gaussian with a standard deviation of 0.1. From there, we use both the Extended Kalman filter and the Unscented Kalman filter to estimate the state of the system, and compare the results. Our expectation is that the Unscented Kalman filter will preform better due to the improvements made to it over the Extended Kalman filter regarding the propagation of error during the linearization in the Extended Kalman filter.

Additionally, we will consider the difference in least squared error between the two methods to observe where each method performed best. Finally, we will observe the differances in computation time between the Extended Kalman filter and the Unscented Kalman filter for this given system. We will first consider time from between 1 and 10, and then between 1 and 100 (the x-axis is time, the y-axis gives the value of the system's state).

Because our system observations provided so little information about the system state, we see that we had a difficult time estimating the second and third system states in figures

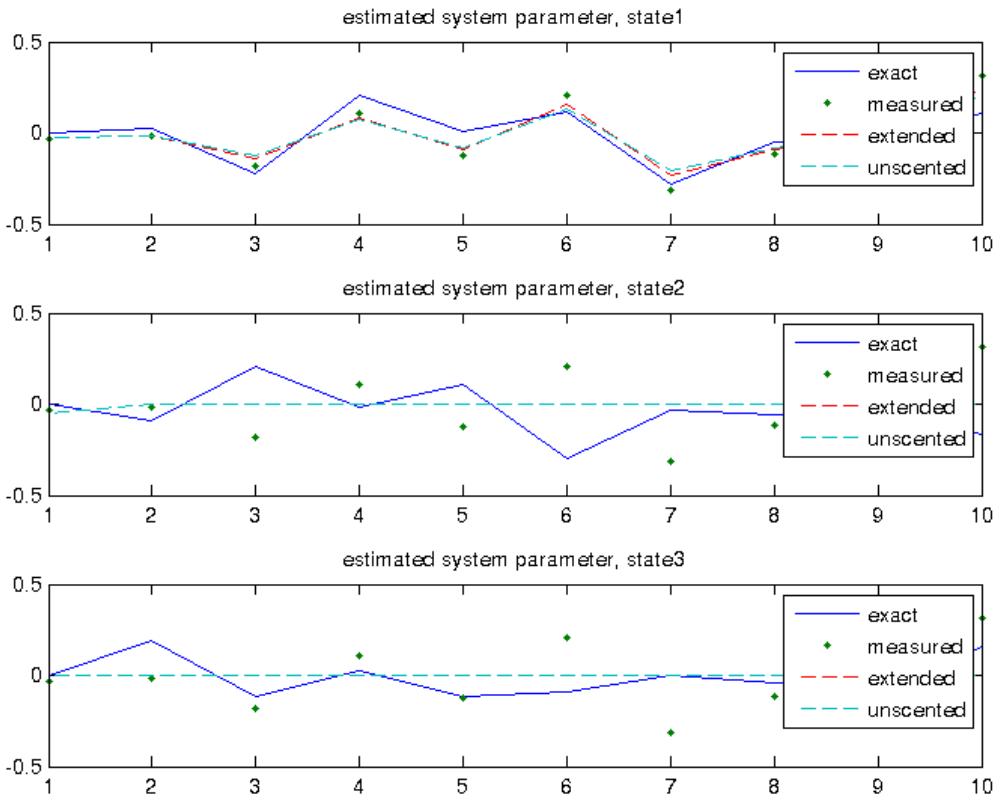

Figure 1: N=10; Estimates from both Extended and Unscented Kalman filter
0.0057s to compute EKF, 0.0059s to compute UKF

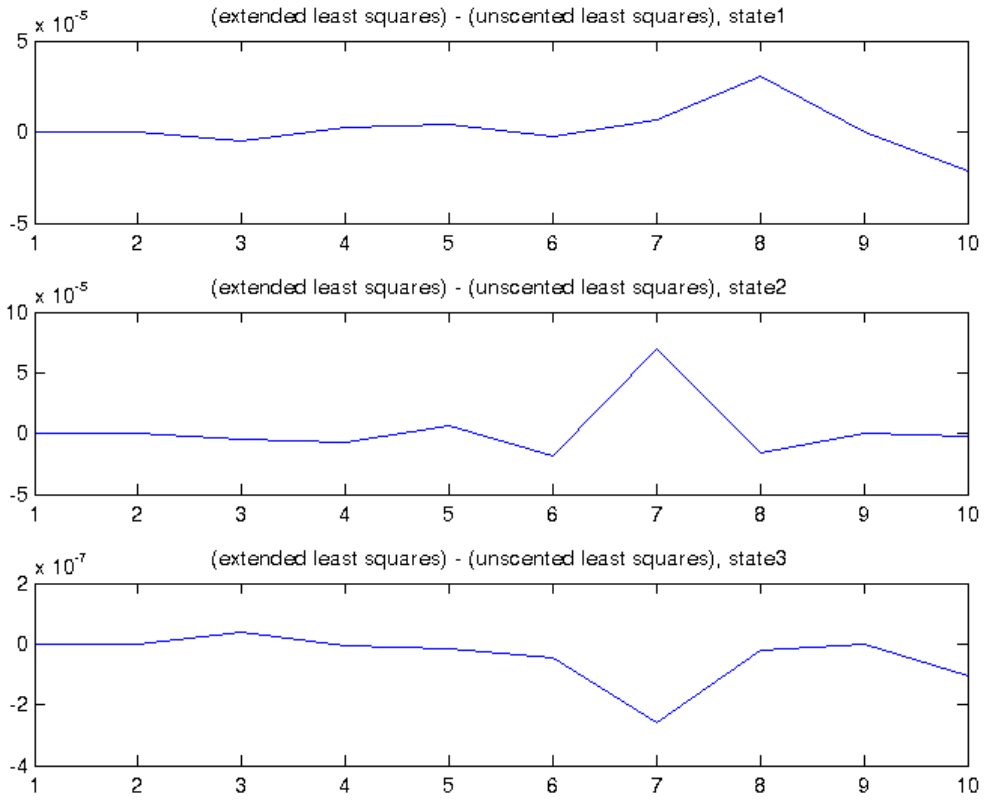

Figure 2: N=10; Comparison between Extended and Unscented Kalman filters
Below the line $y = 0$ corresponds to a worse estimate by the Unscented Kalman filter, above the line $y = 0$ corresponds to a worse estimate by the Extended Kalman filter
Note that the MATLAB file nonlinear_runner.m can be modified slightly to also display a table of the results displayed here.

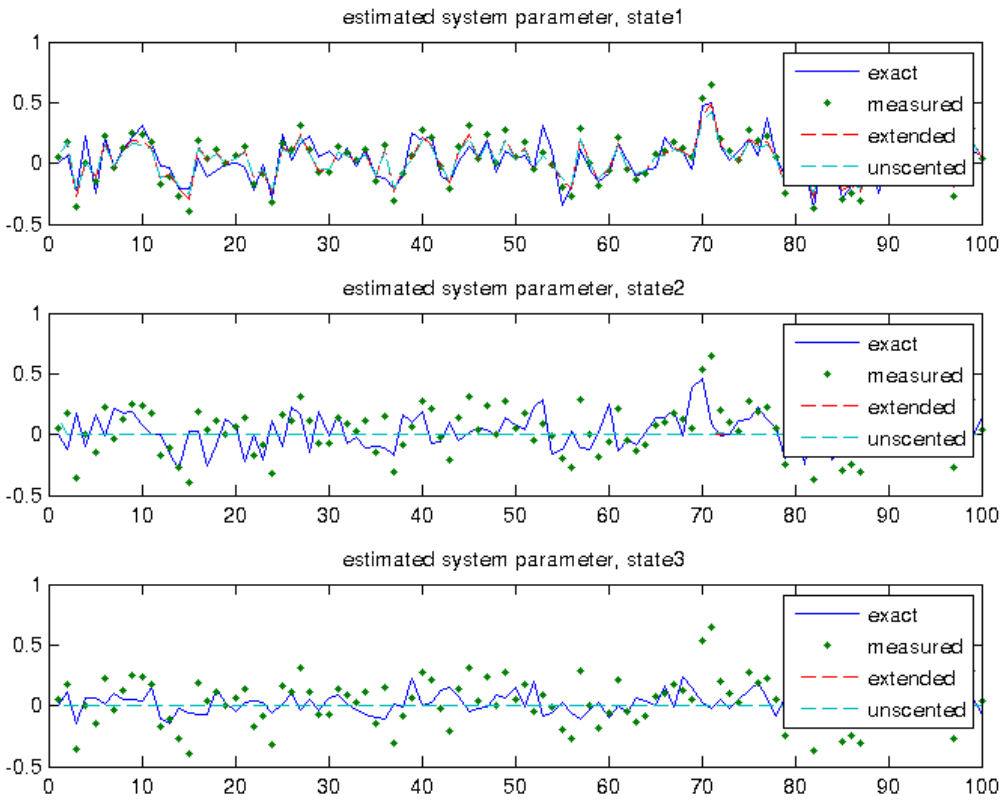

Figure 3: N=100; Estimates from both Extended and Unscented Kalman filter
0.043s to compute EKF, 0.047s to compute UKF

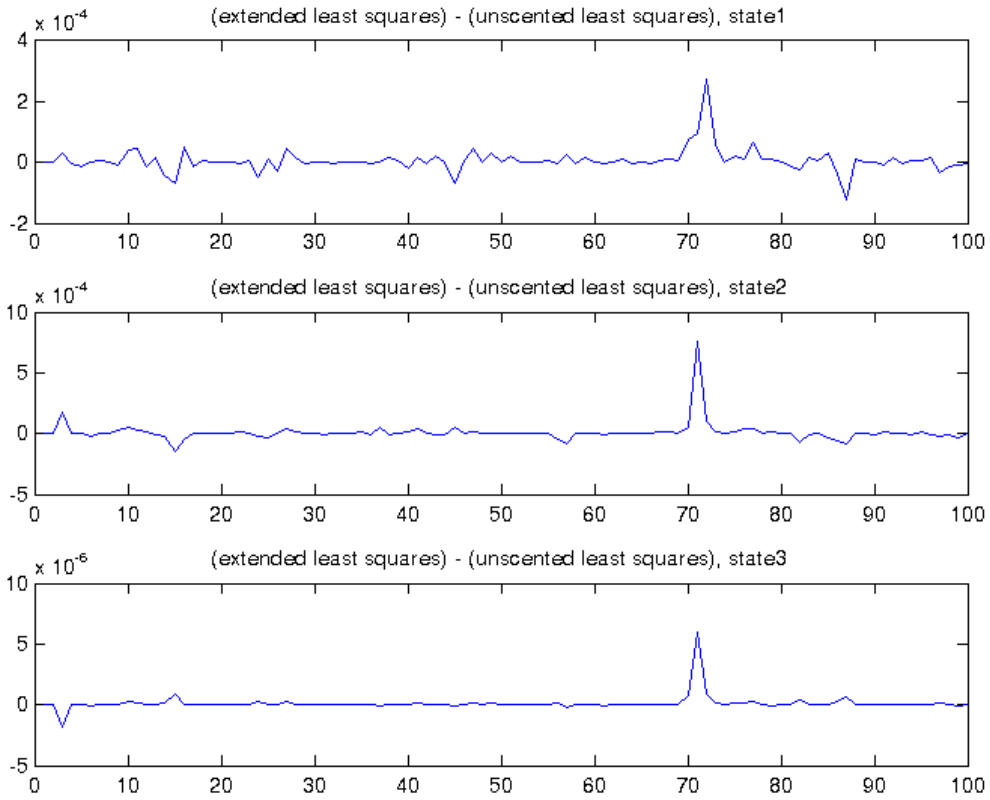

Figure 4: N=100; Comparison between Extended and Unscented Kalman filters
Below the line $y = 0$ corresponds to a worse estimate by the Unscented Kalman filter, above the line $y = 0$ corresponds to a worse estimate by the Extended Kalman filter

1 and 3. In figures 2 and 4, we can see that although the Extended Kalman filter and the Unscented Kalman filter performed similarly, the Unscented Kalman filter had a slight performance lead over the Extended Kalman filter. Finally, notice that, as we expected, both filters took similar amounts of time to complete their computations. In the end, however, the Extended Kalman filter clearly has a slight lead in terms of computational speed.

It was also observed by LaViola [8] (in an experimental setting) that in certain applications, the Extended Kalman filter and the Unscented Kalman filter perform similarly. In particular, he observed that any small benefit he may have gained by using the Unscented Kalman filter was greatly outweighed by the extra computational time. In the simulations which generated figures 1, 2, 3, 4, the difference in computational time was very slight, however, this is likely due to performing the simulations on such a small number of data points, in an effort to produce visually significant graphics.

0.8 Future Investigations: Several Kalman Filter Modifications

Several modifications to the standard Kalman filter are mentioned by Humpherys and West [6]. Future investigations may explore the extent of potential improvements provided by these modifications.

Smoothed estimates [5, p.808]: Normally, as new system measurements become available, the standard Kalman filter only updates with the most recent system state estimate. In this method, we can instead perform smoothing by updating previous states as well. Whenever we make new observations of our system (say up to time k), we will first update the estimated states $X_1, \ldots, X_K$ with the new information. From here, we now use X_k and the most recent measurement to estimate X_{k+1}.

Fixed-lag smoothing [5, p.809]: Similar to smoothed estimates, in this technique, instead of re-estimating all the way back in time to X_1, we instead only re-estimate a fixed amount into the past, up to time l, so we instead use observations up until time k

to re-estimate X_{k-l} through X_k and then we use X_k and the most recent observation to estimate X_{k+1}. This saves on the computation of re-estimating ancient estimates of the system state which aren't likely to change any more.

Fading memory [5, p.810]: When performing an update to the state space estimate $X_{k+1|k}$, the standard Kalman filter weights every previous measurement based on their covariance. In other words, the Kalman filter may consider a measurement from a comparatively long time ago to be just as relevant as a measurement which was just taken. In the case of fading memory, we weight older observations less heavily than newer observations. Taking the objective function from the Kalman filter derivation, we modify it slighly into

$$J_k(z_k) = \frac{\lambda^k}{2}\|x_0 - \mu_0\|^2_{Q_0^{-1}} + \frac{1}{2}\sum_{i=1}^{k}\lambda^{k-i}\|y_i - H_i x_i\|^2_{R_i^{-1}} + \frac{1}{2}\sum_{i=1}^{k}\lambda^{k-i}\|x_i - F_i x_{i-1} - G_i u_i\|^2_{Q_i^{-1}}$$

where we call $0 < \lambda \leq 1$ the forgetting factor, or the factor by which we bias towards the more recent observations. If $\lambda = 1$, then that corresponds to "perfect memory" in which all measurements are weighted as before. As λ decreases, the method becomes more "forgetful" as we bias towards more recent measurements.

0.9 Concluding Remarks

In exploring the Kalman filter and several of its variations, we observed the utility and an application of the standard Kalman filter. We saw how several of the Kalman filter variants improved upon certain aspects of the Kalman filter, and upon each other. Specifically, we observed an example situation in which the standard Kalman filter works well in a linear situation, and we saw how the Extended Kalman filter improves upon the standard Kalman filter by working with non-linear systems through a linearization process. We then saw how the Unscented Kalman filter improved upon the Extended Kalman filter due to being able to handle more complicated non-linear systems since it does not need to linearize the system. Next, we observed a derivation of the standard Kalman filter using Newtons method. From there, we saw a comparison of the effectiveness of

the extended Kalman filter versus the unscented Kalman filter in estimating a non-linear system. We concluded with proposed investigations into possible improvements of the standard Kalman filter as suggested by Humpherys and West.

Bibliography

[1] Yi Cao. Learning the Extended Kalman Filter, January 2008.

[2] Yi Cao. Learning the Unscented Kalman Filter, January 2008.

[3] Charles Chui and Guanrong Chen. *Kalman Filtering: with Real-Time Applications*. Springer, 4 edition, 2009.

[4] M. H. A. Davis and R. B. Vinter. *Stochastic Modelling and Control*. Chapman and Hall, 1985.

[5] Jeffery Humpherys, Preston Redd, and Jeremy West. A Fresh Look at the Kalman Filter. *SIAM Review*, 54(4):801–823, 2012.

[6] Jeffery Humpherys and Jeremy West. Kalman Filter via Newton's Method. *Control Systems, IEEE*, 30(6):101–106, December 2010.

[7] Lindsay Kleeman. Understanding and Applying Kalman Filtering. July 2007.

[8] Joseph LaViola. A Comparison of Unscented and Extended Kalman Filtering for Estimating Quaternion Motion. In *American Control Conference, 2003. Proceedings of the 2003*. American Control Conference, 2003.

[9] Maria Ribeiro. Kalman and Extended Kalman Filters: Concept, Derivation and Properties. Pedagogical publications, February 2004.

[10] Gabriel Terejanu. Extended Kalman Filter Tutorial. Technical report, Department of Computer Science and Engineering, University at Buffalo, Buffalo, New York, 2008.

[11] Gabriel Terejanu. Unscented Kalman Filter Tutorial. Technical report, Department of Computer Science and Engineering, University at Buffalo, Buffalo, New York, 2008.

[12] Eric Wan and Rudolph van der Merwe. The Unscented Kalman Filter for Nonlinear Estimation. In *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, October 2000.

0.10 Non-Linear Kalman Filter Runner

```
1  % This file runs a simulation of the Extended and Unsecented Kalman ...
       filters, both on the same dataset and observations.  The ...
       implementation of these two filters is provided by Yi Cao (I used ...
       this implementation rather than one provided by MATLAB in order ...
       to be able to explicityly view the source code.
2  %
3  % Theodore S. Lindsey
4
5  clear
6
7  n=3;              %number of state
8  q=0.1;            %std of process
9  r=0.1;            %std of measurement
10 Q=q^2*eye(n);     % covariance of process
11 R=r^2;            % covariance of measurement
12 f=@(x)[x(2);x(3);0.05*x(1)*(x(2)+x(3))];   % nonlinear state ...
       equations - Process nonlinear vector function
13 h=@(x)x(1);                                % measurement equation - ...
       Observation nonlinear vector function
14 s=[0;0;0];                                 % initial state
15 ux=s+q*randn(3,1);  %initial state         % initial state with noise
16 ex=ux
17
18 uP = eye(n);                               % initial state covraiance
19 eP = uP
20 N=10;                                      % total dynamic steps
21 uxV = zeros(n,N);          %estmate        % allocate memory, unscented
22 exV = uxV;                                 % allocate memory, extended
23 sV = zeros(n,N);           %actual
24 zV = zeros(1,N);           %observations
25
26 eDuration=0;               %set timer
27 uDuration=0;               %set timer
```

```
28
29
30 for k=1:N
31   z = h(s) + r*randn;                          % measurments
32   sV(:,k)= s;                                  % save actual state record
33   zV(k)  = z;                                  % save measurment record
34   tic
35   [ex, eP] = ekf(f,ex,eP,h,z,Q,R);             % ekf
36   eDuration= eDuration + toc;
37   tic
38   exV(:,k) = ex;                               % save ekf estimate
39   [ux, uP] = ukf(f,ux,uP,h,z,Q,R);             % ukf
40   uDuration= uDuration + toc;
41   uxV(:,k) = ux;                               % save ukf estimate
42   s = f(s) + q*randn(3,1);                     % update process
43 end
44
45
46 %calculate least squares
47 iLS = zeros(3,N);
48 eLS = iLS;
49 uLS = iLS;
50 for k=1:N
51     iLS(:,k) = (1/2)*abs(f(sV(:,k)) - zV(k)).^2;
52     eLS(:,k) = (1/2)*abs(f(exV(:,k)) - zV(k)).^2;
53     uLS(:,k) = (1/2)*abs(f(uxV(:,k)) - zV(k)).^2;
54 end
55
56
57 %graph least squares
58 LeastSquares = figure('Position', [100, 100, 900, 900]);
59 for k=1:3
60     subplot(3,1,k)
61     plot(1:N,eLS(k,:)-uLS(k,:),'-')
```

```
62      title(strcat('(extended least squares) - (unscented least ...
            squares), state ',num2str(k)))
63  end
64  print(LeastSquares,'-dpng','-r100','LeastSquares.png')
65
66
67  %graph estimates
68  Estimates = figure('Position', [100, 100, 900, 900]);
69  for k=1:3                                   % plot results
70    subplot(3,1,k)
71    plot(1:N, sV(k,:), '-', 1:N, zV(:),'.', 1:N, exV(k,:), '--', 1:N, ...
          uxV(k,:), '--')
72    %plot(1:N, sV(k,:), '-', 1:N, zV(:),'-')
73    legend('exact','measured','extended','unscented');
74      title(strcat('estimated system parameter, state ',num2str(k)))
75    %legend('exact','measured');
76  end
77  print(Estimates,'-dpng','-r100','Estimates.png')
78
79  disp(strcat('EKF computation time: ',num2str(eDuration),'s'))
80  disp(strcat('UKF computation time: ',num2str(uDuration),'s'))
```

0.11 Extended Kalman Filter Implementation

[1]

```
1  function [x,P]=ekf(fstate,x,P,hmeas,z,Q,R)
2  % EKF   Extended Kalman Filter for nonlinear dynamic systems
3  % [x, P] = ekf(f,x,P,h,z,Q,R) returns state estimate, x and state ...
       covariance, P
4  % for nonlinear dynamic system:
5  %           x_k+1 = f(x_k) + w_k
6  %           z_k   = h(x_k) + v_k
7  % where w ~ N(0,Q) meaning w is gaussian noise with covariance Q
```

```
8  %       v ~ N(0,R) meaning v is gaussian noise with covariance R
9  % Inputs:   f: function handle for f(x)
10 %           x: "a priori" state estimate
11 %           P: "a priori" estimated state covariance
12 %           h: fanction handle for h(x)
13 %           z: current measurement
14 %           Q: process noise covariance
15 %           R: measurement noise covariance
16 % Output:   x: "a posteriori" state estimate
17 %           P: "a posteriori" state covariance
18 % Yi Cao at Cranfield University
19
20 [x1,A]=jaccsd(fstate,x);    %nonlinear update and linearization at ...
       current state
21 P=A*P*A'+Q;                %partial update
22 [z1,H]=jaccsd(hmeas,x1);    %nonlinear measurement and linearization
23 P12=P*H';                  %cross covariance
24 % K=P12*inv(H*P12+R);       %Kalman filter gain
25 % x=x1+K*(z-z1);            %state estimate
26 % P=P-K*P12';               %state covariance matrix
27 R=chol(H*P12+R);            %Cholesky factorization
28 U=P12/R;                    %K=U/R'; Faster because of back substitution
29 x=x1+U*(R'\(z-z1));         %Back substitution to get state update
30 P=P-U*U';                   %Covariance update, ...
       U*U'=P12/R/R'*P12'=K*P12.
31
32 function [z,A]=jaccsd(fun,x)
33 % JACCSD Jacobian through complex step differentiation
34 % [z J] = jaccsd(f,x)
35 % z = f(x)
36 % J = f'(x)
37 %
38 z=fun(x);
39 n=numel(x);
40 m=numel(z);
```

```
41 A=zeros(m,n);
42 h=n*eps;
43 for k=1:n
44     x1=x;
45     x1(k)=x1(k)+h*i;
46     A(:,k)=imag(fun(x1))/h;
47 end
```

0.12 Unscented Kalman Filter Implementation

[2]

```
1  function [x,P]=ukf(fstate,x,P,hmeas,z,Q,R)
2  % UKF   Unscented Kalman Filter for nonlinear dynamic systems
3  % [x, P] = ukf(f,x,P,h,z,Q,R) returns state estimate, x and state ...
       covariance, P
4  % for nonlinear dynamic system (for simplicity, noises are assumed ...
       as additive):
5  %           x_k+1 = f(x_k) + w_k
6  %           z_k   = h(x_k) + v_k
7  % where w ~ N(0,Q) meaning w is gaussian noise with covariance Q
8  %       v ~ N(0,R) meaning v is gaussian noise with covariance R
9  % Inputs:   f: function handle for f(x)
10 %           x: "a priori" state estimate
11 %           P: "a priori" estimated state covariance
12 %           h: fanction handle for h(x)
13 %           z: current measurement
14 %           Q: process noise covariance
15 %           R: measurement noise covariance
16 % Output:   x: "a posteriori" state estimate
17 %           P: "a posteriori" state covariance
18 % Yi Cao at Cranfield University
19
20 L=numel(x);                                 %numer of states
```

```
21 m=numel(z);                                  %numer of measurements
22 alpha=1e-3;                                  %default, tunable
23 ki=0;                                        %default, tunable
24 beta=2;                                      %default, tunable
25 lambda=alpha^2*(L+ki)-L;                     %scaling factor
26 c=L+lambda;                                  %scaling factor
27 Wm=[lambda/c 0.5/c+zeros(1,2*L)];            %weights for means
28 Wc=Wm;
29 Wc(1)=Wc(1)+(1-alpha^2+beta);               %weights for covariance
30 c=sqrt(c);
31 X=sigmas(x,P,c);                            %sigma points around x
32 [x1,X1,P1,X2]=ut(fstate,X,Wm,Wc,L,Q);          %unscented ...
       transformation of process
33 % X1=sigmas(x1,P1,c);                        %sigma points around x1
34 % X2=X1-x1(:,ones(1,size(X1,2)));            %deviation of X1
35 [z1,Z1,P2,Z2]=ut(hmeas,X1,Wm,Wc,m,R);       %unscented ...
       transformation of measurments
36 P12=X2*diag(Wc)*Z2';                         %transformed ...
       cross-covariance
37 K=P12*inv(P2);
38 x=x1+K*(z-z1);                               %state update
39 P=P1-K*P12';                                 %covariance update
40
41 function [y,Y,P,Y1]=ut(f,X,Wm,Wc,n,R)
42 %Unscented Transformation
43 %Input:
44 %        f: nonlinear map
45 %        X: sigma points
46 %       Wm: weights for mean
47 %       Wc: weights for covraiance
48 %        n: numer of outputs of f
49 %        R: additive covariance
50 %Output:
51 %        y: transformed mean
52 %        Y: transformed smapling points
```

```
53 %         P: transformed covariance
54 %        Y1: transformed deviations
55
56 L=size(X,2);
57 y=zeros(n,1);
58 Y=zeros(n,L);
59 for k=1:L
60     Y(:,k)=f(X(:,k));
61     y=y+Wm(k)*Y(:,k);
62 end
63 Y1=Y-y(:,ones(1,L));
64 P=Y1*diag(Wc)*Y1'+R;
65
66 function X=sigmas(x,P,c)
67 %Sigma points around reference point
68 %Inputs:
69 %        x: reference point
70 %        P: covariance
71 %        c: coefficient
72 %Output:
73 %        X: Sigma points
74
75 A = c*chol(P)';
76 Y = x(:,ones(1,numel(x)));
77 X = [x Y+A Y-A];
```