Interactive Robotics Letters
Tutorial

Understanding Nonlinear Kalman Filters, Part II: An Implementation Guide

Matthew Rhudy* and Yu Gu†

West Virginia University, Morgantown, WV 26506, USA

***Abstract*— Kalman filters provide an important technique for estimating the states of engineering systems. With several variations of nonlinear Kalman filters, there is a lack of guidelines for filter selection with respect to a specific research and engineering application. This creates a need for an in-depth discussion of the intricacies of different nonlinear Kalman filters. Particularly of interest for practical state estimation applications are the Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF). This tutorial is divided into three self-contained articles. Part I gives a general comparison of EKF and UKF, and offers a guide to the selection of a filter. Part II presents detailed information about the implementation of EKF and UKF, including equations, tips, and example codes. Part III examines different techniques for determining the assumed noise characteristics of the system as well as tuning procedures for these nonlinear Kalman filters.**

***Index Terms*—Kalman Filters, Nonlinear Filters, Extended Kalman Filter, Unscented Kalman Filter**

I. Introduction

The Unscented Kalman Filter (UKF) is a versatile engineering tool that once understood can provide good nonlinear estimation results for many practical problems (Julier and Uhlmann, 1997). The UKF has been effectively implemented for a variety of applications, such as attitude estimation (Rhudy et al., 2011; Gross et al., 2012; Rhudy et al., 2013a), wind estimation (Rhudy et al., 2013c), and airspeed estimation (Gururajan et al., 2013a; Gururajan et al., 2013b). Due to the lack of Jacobian matrix calculations, the UKF can also be constructed and altered more easily than the Extended Kalman Filter (EKF) (Kalman and Bucy, 1961) in the prototyping stages of filter design (Rhudy and Gu, 2013). Although there are existing works that detail the implementation of UKF, e.g. (Julier and Uhlmann, 1997; van der Merwe et al., 2004), these technical works are written in high-level language and can be a bit challenging to understand for those who are less familiar with the Unscented Transformation (UT). This guide aims to clarify the technical literature to make the UKF a more accessible tool for a variety of users.

The rest of this paper is organized as follows. First, the linear and nonlinear stochastic state estimation problems are discussed, including definitions of the Linear and Extended Kalman Filters. Then the theoretical equations of the UKF are provided and discussed for both additive and non-additive noise assumptions. Finally, an example problem is given to help illustrate the application of the UKF to a practical situation. Additionally, detailed outlines of the additive and non-additive UKF are provided in the appendices, along with the complete MATLAB code for the considered example problem.

* Ph.D. Candidate, Department of Mechanical and Aerospace Engineering
† Assistant Professor, Department of Mechanical and Aerospace Engineering and Adjunct Assistant Professor, Lane Department of Computer Science and Electrical Engineering

Citation: Rhudy, M., and Gu, Y., "Title of Paper," *Interactive Robotics Letters*, West Virginia University, June 2013. Link: http://www2.statler.wvu.edu/~irl/page13.html

II. STOCHASTIC STATE ESTIMATION

A. *Linear Stochastic State Estimation*

A general discrete-time linear state space system can be written as

$$\begin{aligned}\mathbf{x}_k &= \mathbf{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{G}_{k-1}\mathbf{u}_{k-1} + \mathbf{w}_{k-1} \\ \mathbf{y}_k &= \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k\end{aligned} \tag{1}$$

where **F** is the state transition matrix, **G** is the control input matrix, **H** is the observation matrix, **x** is the state vector, **y** is the output vector, **u** is the input vector, **w** is the process noise vector, **v** is the measurement noise vector, and *k* is the discrete-time index. The dimensions of **x**, **y**, **u**, **w**, and **v** are n_x, n_y, n_u, n_w, and n_v respectively. The process and measurement noise covariance terms are considered to be uncorrelated, white, and Gaussian with zero mean and known covariance matrices **Q** and **R** respectively, as in

$$\begin{aligned}\mathbf{w}_k &\sim N\left(\mathbf{0}, \mathbf{Q}_k\right) \\ \mathbf{v}_k &\sim N\left(\mathbf{0}, \mathbf{R}_k\right) \\ E\left[\mathbf{w}_k \mathbf{v}_k^T\right] &= \mathbf{0}\end{aligned} \tag{2}$$

It is also assumed that the initial state, $\hat{\mathbf{x}}_0$, is known with corresponding uncertainty given by the initial error covariance matrix, $\mathbf{P}_0$. Next, a common approach to solving this type of problem is discussed.

B. *Linear Kalman Filter*

For the system described in (1), the Linear Kalman Filter (LKF) can be applied to estimate the state vector, **x**. The LKF is implemented in two steps: prediction and update. The prediction equations determine the *a priori* estimates of the state and error covariance, given by

$$\begin{aligned}\hat{\mathbf{x}}_{k|k-1} &= \mathbf{F}_{k-1}\hat{\mathbf{x}}_{k-1} + \mathbf{G}_{k-1}\mathbf{u}_{k-1} \\ \mathbf{P}_{k|k-1} &= \mathbf{F}_{k-1}\mathbf{P}_{k-1}\mathbf{F}_{k-1}^T + \mathbf{Q}_{k-1}\end{aligned} \tag{3}$$

In the update step, the *a priori* estimates are updated using the calculated Kalman gain matrix, **K**, resulting in the *a posteriori* estimates

$$\begin{aligned}\mathbf{K}_k &= \mathbf{P}_{k|k-1}\mathbf{H}_k^T\left(\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k\right)^{-1} \\ \hat{\mathbf{x}}_k &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\left(\mathbf{y}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k|k-1}\right) \\ \mathbf{P}_k &= \left(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k\right)\mathbf{P}_{k|k-1}\end{aligned} \tag{4}$$

where **I** is an identity matrix and **y** is the measurement of the output vector provided by an independent measurement system. The LKF is limited to linear systems. For a more detailed discussion of the LKF, see (Anderson and Moore, 1979) or (Simon, 2006). Next, the nonlinear stochastic state estimation problem is discussed.

C. *Nonlinear Stochastic State Estimation*

A general discrete-time nonlinear state space system can be written as

$$\begin{aligned}\mathbf{x}_k &= \mathbf{f}\left(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}, \mathbf{w}_{k-1}\right) \\ \mathbf{y}_k &= \mathbf{h}\left(\mathbf{x}_k, \mathbf{u}_k, \mathbf{v}_k\right)\end{aligned} \tag{5}$$

where **f** is the vector-valued state prediction function, **h** is the vector-valued observation function, **x** is the state vector, **y** is the output vector, **u** is the input vector, **w** is the process noise vector, **v** is the measurement noise vector, and *k* is the discrete-time index. The dimensions of **x**, **y**, **u**, **w**, and **v** are n_x, n_y, n_u, n_w, and n_v respectively. The process and measurement noise covariance terms are considered to be uncorrelated, white, and Gaussian with zero mean and known covariance matrices **Q** and **R** respectively, as in (2). It is also assumed that the initial state, $\hat{\mathbf{x}}_0$, is known with corresponding uncertainty given by the initial error covariance matrix, $\mathbf{P}_0$. A diagram of the general nonlinear state estimation problem using Kalman-based

filters is shown in Figure 1 (left), while the simplified additive noise case is provided in Figure 1 (right). Next, a common approach to solving this type of problem is discussed.

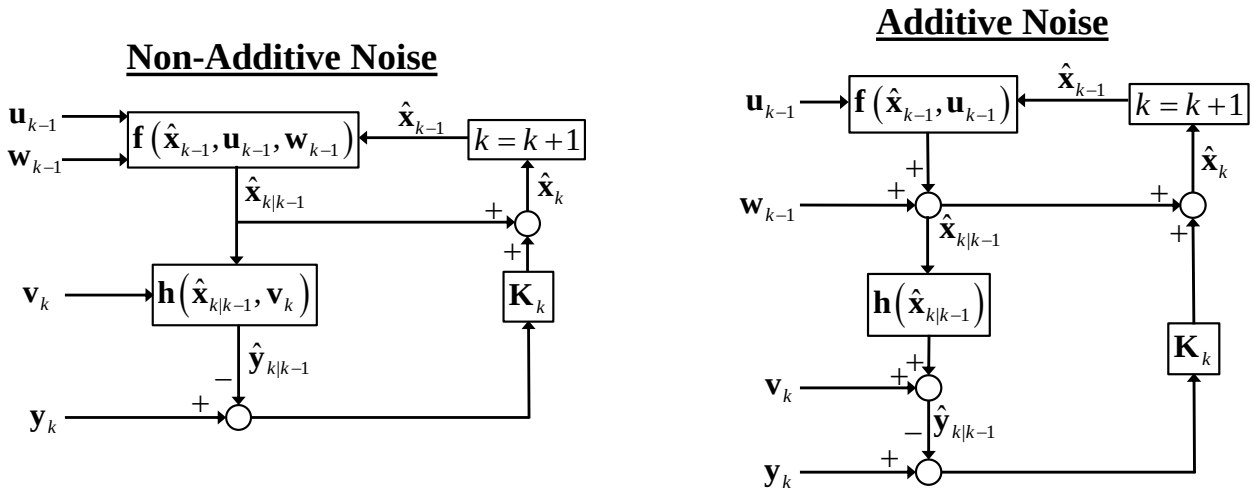

Figure 1. Conceptual Representation of Noise Assumptions for Nonlinear State Estimation

D. Extended Kalman Filter

The Extended Kalman Filter (EKF) (Kalman and Bucy, 1961) is a standard approach for nonlinear stochastic state estimation. The implementation of the EKF is similar to the LKF, except that Jacobian matrices need to be calculated at each time step to determine the local linearized model of the system.

$$\mathbf{F}_{k-1} = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}_{k-1}}\right|_{\hat{\mathbf{x}}_{k-1}}, \quad \mathbf{H}_k = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{x}_k}\right|_{\hat{\mathbf{x}}_{k|k-1}}, \quad \mathbf{L}_{k-1} = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{w}_{k-1}}\right|_{\hat{\mathbf{x}}_{k-1}}, \quad \mathbf{M}_k = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{v}_k}\right|_{\hat{\mathbf{x}}_{k|k-1}} \tag{6}$$

Note that **F** and **H** are differentiated with respect to the state estimate, and **L** and **M** are differentiated with respect to the process and measurement noise terms respectively. The prediction matrices **F** and **L** are evaluated at the previous state estimate, while the update matrices **H** and **M** are evaluated at the *a priori* state estimate. If possible, these Jacobian matrices should be calculated offline in order to obtain closed-form solutions which can be programmed directly (i.e., no numerical derivatives are required in the coding). Once the Jacobian matrices have been calculated, then the LKF technique is applied to the local linearized system, using

$$\begin{aligned}
\hat{\mathbf{x}}_{k|k-1} &= \mathbf{f}\left(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1}, \mathbf{0}\right) \\
\mathbf{P}_{k|k-1} &= \mathbf{F}_{k-1}\mathbf{P}_{k-1}\mathbf{F}_{k-1}^T + \mathbf{L}_{k-1}\mathbf{Q}_{k-1}\mathbf{L}_{k-1}^T \\
\mathbf{K}_k &= \mathbf{P}_{k|k-1}\mathbf{H}_k^T\left(\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{M}_k\mathbf{R}_k\mathbf{M}_k^T\right)^{-1} \\
\hat{\mathbf{x}}_k &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\left(\mathbf{y}_k - \mathbf{h}\left(\hat{\mathbf{x}}_{k|k-1}, \mathbf{u}_k, \mathbf{0}\right)\right) \\
\mathbf{P}_k &= \left(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k\right)\mathbf{P}_{k|k-1}
\end{aligned} \tag{7}$$

Note that these equations represent the non-additive noise equations for the EKF. For the additive noise case, the Jacobian matrices **L** and **M** simply become identity matrices. For more discussion of the EKF, see (Simon, 2006). In the following section, an alternative to the EKF, the Unscented Kalman Filter (UKF), is developed.

III. EQUATIONS OF THE UNSCENTED KALMAN FILTER

A. *Unscented Transformation*

The Unscented Transformation (UT) is the central technique of the UKF which is used to handle the nonlinearity in a nonlinear transformation $\mathbf{y} = \mathbf{f}(\mathbf{x})$, where $\mathbf{x}$ and $\mathbf{y}$ are $L \times 1$ vectors, and $\mathbf{f}$ is an $L \times 1$ vector-valued function. Here, $\mathbf{x}$ is a random variable which is typically assumed to be normally distributed (Gaussian) with mean, $\bar{\mathbf{x}}$, and covariance, $\mathbf{P}_\mathbf{x}$. The UT provides a statistical alternative to the analytical linearization approach using Jacobian matrices which is used in the EKF. For a detailed analytical comparison of these linearization techniques, see (Rhudy et al., 2013b). The UT uses a small set of deterministically selected points, called sigma-points, which are selected based on the *a priori* conditions, i.e. the points are selected from the assumed prior distribution. The spread of these points or the confidence level from the prior distribution is determined based on the selected scaling parameters for the UT. There are different representations and notations for the scaling parameters, but in the end these representations are equivalent, and affect the spread of the sigma-points as well as the weight vectors that are used in reconstructing the *a posteriori* (after the transformation) statistics.

The scaling of the UT can be fully represented by three scaling parameters (Julier and Uhlmann, 1997; Wan and van der Merwe, 2002). The primary scaling parameter, α, determines the spread of the sigma-points. This parameter can vary between 10^{-4} and 1. Smaller α leads to a tighter (closer) selection of sigma-points, while larger α gives a wider spread of sigma-points. The secondary scaling parameter, β, is used to include information about the prior distribution (for Gaussian distributions, $\beta = 2$ is optimal). The tertiary scaling parameter, κ, is usually set to 0, for more information see (Julier and Uhlmann, 1997). Using these three scaling parameters, an additional scaling parameter, λ, and weight vectors, $\boldsymbol{\eta}^m$ (mean) and $\boldsymbol{\eta}^c$ (covariance) are defined

$$\begin{aligned}
\lambda &= \alpha^2\left(L+\kappa\right)-L \\
\boldsymbol{\eta}_0^m &= \lambda/\left(L+\lambda\right) \\
\boldsymbol{\eta}_0^c &= \lambda/\left(L+\lambda\right)+1-\alpha^2+\beta \\
\boldsymbol{\eta}_i^m &= \boldsymbol{\eta}_i^c = 1/\left[2\left(L+\lambda\right)\right], \quad i=1,...,2L
\end{aligned} \tag{8}$$

where L is the length of the state vector. The parameter, λ, the prior mean, $\bar{\mathbf{x}}$, and covariance, $\mathbf{P}_\mathbf{x}$, of the random variable $\mathbf{x}$ are then used to generate $2L+1$ sigma-points, as in

$$\boldsymbol{\chi} = \left[\begin{array}{ccc} \bar{\mathbf{x}} & \bar{\mathbf{x}}+\sqrt{L+\lambda}\sqrt{\mathbf{P}_\mathbf{x}} & \bar{\mathbf{x}}-\sqrt{L+\lambda}\sqrt{\mathbf{P}_\mathbf{x}} \end{array}\right] \tag{9}$$

where $\boldsymbol{\chi}$ is an $L \times (2L+1)$ matrix of sigma-points, where each column of this matrix represents a sigma-point. Note that in (9), the sum of a vector and matrix is defined as adding the vector to each column of the matrix. Alternatively, the $L \times 1$ column vector $\bar{\mathbf{x}}$ can be multiplied by a $1 \times L$ row vector of ones resulting in an $L \times L$ matrix with which standard matrix addition can be used. Note also that (9) contains the square root of a matrix. While there are multiple methods of calculating a matrix square root, the recommended method both in terms of performance and computational efficiency is the Cholesky method (Rhudy et al., 2011). This method uses the Cholesky decomposition to calculate a lower triangular matrix which can then be used as a representation of the matrix square root, i.e.

$$\mathbf{P}_\mathbf{x} = \left(\sqrt{\mathbf{P}_\mathbf{x}}\right)\left(\sqrt{\mathbf{P}_\mathbf{x}}\right)^T \tag{10}$$

where $\sqrt{\mathbf{P}_\mathbf{x}}$ is a lower triangular matrix. Note that this representation is different from a principal matrix square root, which takes the form of (10) without the transpose, and is in general non-triangular. A different method of handling the matrix square root called the "square-root UKF (SR-UKF)" was also proposed (van der Merwe and Wan, 2001), which has improved computational complexity, but can obtain different performance results.

Once the sigma-points have been generated, each point is passed through the nonlinear function, i.e. each column of the sigma-point matrix, $\boldsymbol{\chi}$, is propagated through the nonlinearity, as in

$$\boldsymbol{\psi}^{(i)} = \mathbf{f}\left(\boldsymbol{\chi}^{(i)}\right), \quad i = 0,1,...,2L \tag{11}$$

where the superscript (i) denotes the i^{th} column of the matrix, and $\boldsymbol{\psi}$ is a matrix of transformed sigma-points. Then, the *a posteriori* mean and covariance are estimated using weighted averages of these transformed sigma-points using the weight vectors defined in (8), as in

$$\begin{aligned}\bar{\mathbf{y}} &\approx \sum_{i=0}^{2L} \boldsymbol{\eta}_i^m \boldsymbol{\psi}^{(i)} \\ \mathbf{P}_\mathbf{y} &\approx \sum_{i=0}^{2L} \boldsymbol{\eta}_i^c \left(\boldsymbol{\psi}^{(i)} - \bar{\mathbf{y}}\right)\left(\boldsymbol{\psi}^{(i)} - \bar{\mathbf{y}}\right)^T\end{aligned} \tag{12}$$

where $\bar{\mathbf{y}}$ is the estimated mean of $\mathbf{y}$, and $\mathbf{P}_\mathbf{y}$ is the estimated covariance matrix of $\mathbf{y}$. These values represent the *a posteriori* statistics, or the estimated statistical properties after the nonlinear transformation. Next, an example is provided to help clarify the purpose and usefulness of the UT.

Consider the transformation from 2D Cartesian coordinates to polar coordinates, given by

$$\mathbf{y} = \begin{bmatrix} r \\ \theta \end{bmatrix} = \mathbf{f}(\mathbf{x}) = \mathbf{f}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} \sqrt{x_1^2 + x_2^2} \\ \tan^{-1}(x_2/x_1) \end{bmatrix} \tag{13}$$

Note that for this example, $L = 2$. Assume that the Cartesian coordinates are measured to be 0.2 and 0.6 with variances of 0.8 and 0.3 respectively, with no coupled errors between the two measurements (no off-diagonal terms in covariance matrix), i.e.

$$\bar{\mathbf{x}} = \begin{bmatrix} 0.2 \\ 0.6 \end{bmatrix}, \quad \mathbf{P}_\mathbf{x} = \begin{bmatrix} 0.8 & 0 \\ 0 & 0.3 \end{bmatrix} \tag{14}$$

For this example, consider the scaling parameters $\alpha = 1$, $\beta = 2$, and $\kappa = 0$, thus giving $\lambda = 0$, and weight vectors

$$\begin{aligned}\boldsymbol{\eta}^m &= \begin{bmatrix} 0 & 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix} \\ \boldsymbol{\eta}^c &= \begin{bmatrix} 2 & 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix}\end{aligned} \tag{15}$$

Since the covariance matrix is diagonal, the matrix square root is simply a diagonal matrix containing the square roots of the diagonal terms, as in

$$\sqrt{\mathbf{P}_\mathbf{x}} = \begin{bmatrix} \sqrt{0.8} & 0 \\ 0 & \sqrt{0.3} \end{bmatrix} \tag{16}$$

Note that this matrix is technically a lower triangular matrix, and therefore also satisfies the Cholesky decomposition. Using this definition, the sigma-points are calculated using (9)

$$\boldsymbol{\chi} = \begin{bmatrix} 0.2 & 0.2 + \sqrt{2}\sqrt{0.8} & 0.2 + 0 & 0.2 - \sqrt{2}\sqrt{0.8} & 0.2 - 0 \\ 0.6 & 0.6 + 0 & 0.6 + \sqrt{2}\sqrt{0.3} & 0.6 - 0 & 0.6 - \sqrt{2}\sqrt{0.3} \end{bmatrix} \tag{17}$$

These sigma-points are shown in Figure 2 along with the mean and 1-σ ellipse (as in the curve consisting of points that are one standard deviation away from the mean) for $\mathbf{x}$ (*a priori*) and $\mathbf{y}$ (*a posteriori*). The ellipses are a visual representation of the covariance matrices.

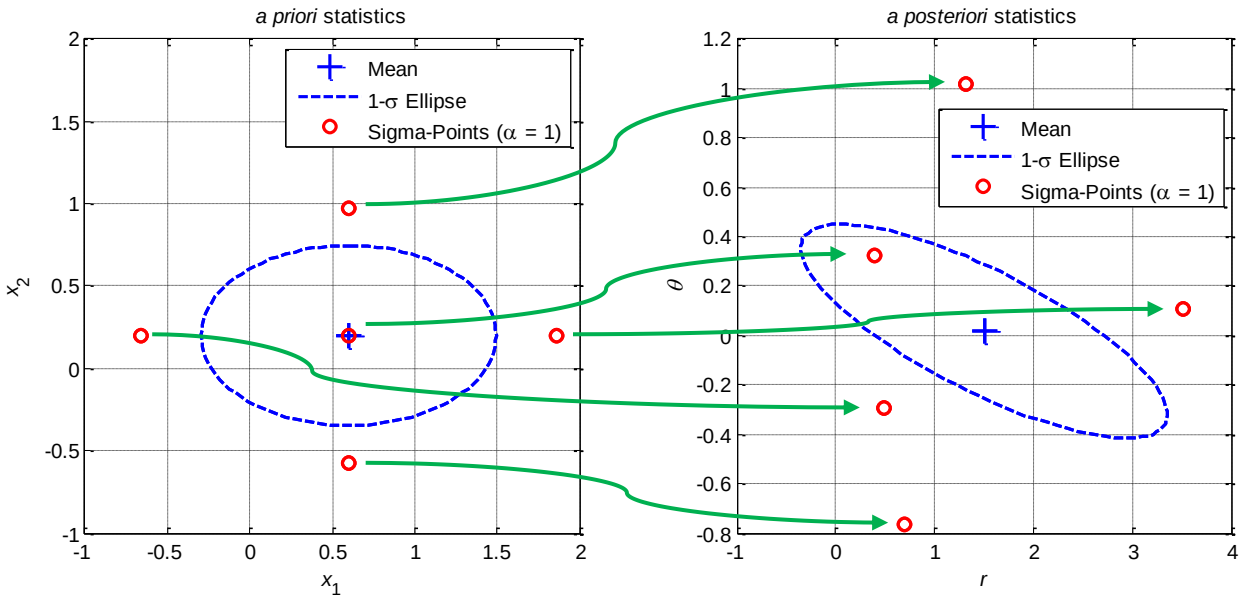

Figure 2. Unscented Transformation Example

In Figure 2, the left shows the *a priori* statistics, i.e. the statistics of **x**, while the right shows the *a posteriori* statistics, i.e. the statistics of **y**. The sigma-points are generated and shown for the *a priori* statistics on the left, and then each is transformed through the nonlinear function resulting in the sigma-points shown on the right. The individual mapping of each sigma-point is shown with arrows. From these transformed sigma-points, the mean and covariance of **y** were calculated using (12). There are a few interesting things to note about Figure 2. First, the transformed sigma-points do *not* fit nicely to a Gaussian distribution, i.e. the sigma-points are not arranged along an elliptical path. In general, when Gaussian random variables are transformed through a nonlinear equation, the resulting distribution will no longer be Gaussian. This represents the primary weakness of this method for example when compared to a particle filter (Rhudy et al., 2013b). Another interesting observation is that even though the errors in **x** were uncorrelated (diagonal $\mathbf{P}_x$), the errors in **y** are correlated (non-diagonal $\mathbf{P}_y$ resulting in a rotated ellipse). This example represented a single instance of the UT. For nonlinear filtering applications, a linearization technique is often required at each time step in the filter. First the nonlinear filtering problem is described, followed by the definitions of the UKF.

B. Standard (Unaugmented) UKF

A common assumption used in stochastic estimation problems is that the process and measurement noise terms are additive, as in

$$\begin{aligned}\mathbf{x}_k &= \mathbf{f}\left(\mathbf{x}_{k-1},\mathbf{u}_{k-1}\right)+\mathbf{w}_{k-1}\\ \mathbf{y}_k &= \mathbf{h}\left(\mathbf{x}_k,\mathbf{u}_k\right)+\mathbf{v}_k\end{aligned} \tag{18}$$

Using this noise assumption, the standard UKF equations can be applied without any need for augmentation of the state vector, which is discussed in the next section, see also (Wu et al., 2005). For this case, the dimension of the sigma-points is the same as the state vector, i.e. $L = n_x$. Starting with the assumed initial conditions $\hat{\mathbf{x}}_0$ and $\mathbf{P}_0$, the UKF is executed recursively. The UKF uses the Unscented Transformation (UT) as described in Section IIA. Note that the scaling parameters are typically assumed constant throughout the UKF, i.e. the same scaling of UT is used for each time step in the filter. These scaling parameters as well as the weight vectors can be defined once prior to executing the filter. At each discrete-time step *k*, first a set of sigma-points are generated from the prior state estimate $\hat{\mathbf{x}}_{k-1}$ and covariance $\mathbf{P}_{k-1}$, as in

$$\boldsymbol{\chi}_{k-1} = \left[\hat{\mathbf{x}}_{k-1} \quad \hat{\mathbf{x}}_{k-1}+\sqrt{L+\lambda}\sqrt{\mathbf{P}_{k-1}} \quad \hat{\mathbf{x}}_{k-1}-\sqrt{L+\lambda}\sqrt{\mathbf{P}_{k-1}}\right] \tag{19}$$

Next, each sigma point is passed through the nonlinear state prediction function, **f**. Since the process noise is assumed to be additive and zero mean, it can be omitted from the prediction function, as in

$$\boldsymbol{\chi}_{k|k-1}^{(i)} = \mathbf{f}\left(\boldsymbol{\chi}_{k-1}^{(i)}, \mathbf{u}_{k-1}\right), \quad i = 0,1,...,2L \tag{20}$$

Note, that these transformed sigma-points use the subscript $k|k$-1, or "k given k-1," which means that this is the predicted value of the sigma-point based on the information from the previous time step. An alternative representation that is sometime used instead of $k|k$-1 is a subscript of k with a superscript of – (minus). Now that these sigma-points have been transformed, the post transformation mean and covariance are calculated using weighted averages of the transformed sigma-points

$$\begin{aligned}\hat{\mathbf{x}}_{k|k-1} &= \sum_{i=0}^{2L} \boldsymbol{\eta}_i^m \boldsymbol{\chi}_{k|k-1}^{(i)} \\ \mathbf{P}_{k|k-1} &= \mathbf{Q}_{k-1} + \sum_{i=0}^{2L} \boldsymbol{\eta}_i^c \left(\boldsymbol{\chi}_{k|k-1}^{(i)} - \hat{\mathbf{x}}_{k|k-1}\right)\left(\boldsymbol{\chi}_{k|k-1}^{(i)} - \hat{\mathbf{x}}_{k|k-1}\right)^T\end{aligned} \tag{21}$$

Note that the assumed process noise covariance matrix, **Q**, is added to the error covariance matrix due to the additive noise assumption. These are the predicted estimates of the mean and covariance for the time-step k, which are sometimes referred to as the *a priori* state and covariance estimates. This is a bit confusing, since *a priori* can also mean before a transformation, and this is after a transformation. In this case, the *a priori* is referring to before the measurement update stage of the filter. Next, the transformed sigma-points are sent through the observation function. As for the prediction, since the measurement noise is additive and zero-mean, the measurement noise is omitted from the observation function, as in

$$\boldsymbol{\psi}_{k|k-1}^{(i)} = \mathbf{h}\left(\boldsymbol{\chi}_{k|k-1}^{(i)}, \mathbf{u}_k\right), \quad i = 0,1,...,2L \tag{22}$$

where $\boldsymbol{\psi}$ is a matrix of output sigma-points, i.e. each column of this matrix is an $n_y \times 1$ point in the output space. These output sigma-points are then used to calculate the predicted output, output covariance matrix, and cross-covariance between the state and output, using

$$\begin{aligned}\hat{\mathbf{y}}_{k|k-1} &= \sum_{i=0}^{2L} \boldsymbol{\eta}_i^m \boldsymbol{\psi}_{k|k-1}^{(i)} \\ \mathbf{P}_k^{yy} &= \mathbf{R}_k + \sum_{i=0}^{2L} \boldsymbol{\eta}_i^c \left(\boldsymbol{\psi}_{k|k-1}^{(i)} - \hat{\mathbf{y}}_{k|k-1}\right)\left(\boldsymbol{\psi}_{k|k-1}^{(i)} - \hat{\mathbf{y}}_{k|k-1}\right)^T \\ \mathbf{P}_k^{xy} &= \sum_{i=0}^{2L} \boldsymbol{\eta}_i^c \left(\boldsymbol{\chi}_{k|k-1}^{(i)} - \hat{\mathbf{x}}_{k|k-1}\right)\left(\boldsymbol{\psi}_{k|k-1}^{(i)} - \hat{\mathbf{y}}_{k|k-1}\right)^T\end{aligned} \tag{23}$$

Note that the assumed measurement noise covariance matrix, **R**, is added to the output covariance matrix due to the additive noise assumption. These covariance matrices are used to calculate the Kalman gain matrix, **K**, using

$$\mathbf{K}_k = \mathbf{P}_k^{xy}\left(\mathbf{P}_k^{yy}\right)^{-1} \tag{24}$$

This Kalman gain is then used to update both the state and covariance estimates, as in

$$\begin{aligned}\hat{\mathbf{x}}_k &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\left(\mathbf{y}_k - \hat{\mathbf{y}}_{k|k-1}\right) \\ \mathbf{P}_k &= \mathbf{P}_{k|k-1} - \mathbf{K}_k \mathbf{P}_k^{yy} \mathbf{K}_k^T\end{aligned} \tag{25}$$

where $\mathbf{y}_k$ (sometimes denoted $\mathbf{z}_k$) is the measurement vector provided from an independent source (not used elsewhere in the filter), and $\hat{\mathbf{x}}_k$ and $\mathbf{P}_k$ are the *a posteriori* (after the measurement update) state and covariance estimates. These estimates are then used as the previous (k-1) estimates for the next time step of the UKF. Repeat this procedure at each time step of the UKF for the desired number of time steps. While it is useful to have a good understanding of the equations behind the UKF, they can still be a bit cryptic in terms of their implementation. A more practical outline of the steps of the additive-noise UKF is provided in Appendix A. Additionally, a side-by-side comparison of the additive noise EKF and UKF equations is given in Appendix B. Next, the method for non-additive noise assumptions is provided.

C. *Augmented UKF*

For non-additive noise assumptions as in (5), the state vector of the UKF can be augmented with additional "states" to model the non-additive process and/or measurement noise terms, as in

$$\mathbf{x}_k^a = \begin{bmatrix} \mathbf{x}_k \\ \mathbf{w}_k \\ \mathbf{v}_k \end{bmatrix} \tag{26}$$

where the superscript '*a*' denotes augmentation. The noise states are initialized to zero and remain zero throughout the filter (due to the zero mean noise assumptions). However, the elements of the sigma-points which correspond to these augmented "states" have an effect in the filter. For this case, the dimension of the augmented state is the sum of the length of the state vector, process noise vector, and measurement noise vector, i.e. $L = n_x + n_w + n_v$. The error covariance matrix must also be augmented accordingly, taking the form of a block diagonal matrix

$$\mathbf{P}_k^a = \begin{bmatrix} \mathbf{P}_k & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_k & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{R}_k \end{bmatrix} \tag{27}$$

where the superscript '*a*' denotes augmentation. Now, the augmented state vector and covariance matrix are used to generate the augmented sigma-points

$$\boldsymbol{\chi}_{k-1}^a = \begin{bmatrix} \boldsymbol{\chi}_{k-1}^x \\ \boldsymbol{\chi}_{k-1}^w \\ \boldsymbol{\chi}_{k-1}^v \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{x}}_{k-1}^a & \hat{\mathbf{x}}_{k-1}^a + \sqrt{L+\lambda}\sqrt{\mathbf{P}_{k-1}^a} & \hat{\mathbf{x}}_{k-1}^a - \sqrt{L+\lambda}\sqrt{\mathbf{P}_{k-1}^a} \end{bmatrix} \tag{28}$$

Note that the matrix square root of the augmented covariance matrix can be done in blocks, as in

$$\sqrt{\mathbf{P}_k^a} = \begin{bmatrix} \sqrt{\mathbf{P}_k} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \sqrt{\mathbf{Q}_k} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \sqrt{\mathbf{R}_k} \end{bmatrix} \tag{29}$$

If constant process and/or measurement noise matrices are used, these matrix square root blocks can be calculated once prior to filtering, which would improve the computational efficiency of the algorithm. The sigma-points can be separated into three sections: state, **x**, process noise, **w**, and measurement noise, **v**, as noted by the superscripts. Now, only the state sigma-points need to be predicted using prior information from both the state and process noise sigma-points

$$\boldsymbol{\chi}_{k|k-1}^{x,(i)} = \mathbf{f}\left(\boldsymbol{\chi}_{k-1}^{x,(i)}, \mathbf{u}_{k-1}, \boldsymbol{\chi}_{k-1}^{w,(i)}\right), \quad i = 0,1,\ldots,2L \tag{30}$$

The *a priori* statistics are then recovered

$$\begin{aligned} \hat{\mathbf{x}}_{k|k-1} &= \sum_{i=0}^{2L} \boldsymbol{\eta}_i^m \boldsymbol{\chi}_{k|k-1}^{x,(i)} \\ \mathbf{P}_{k|k-1} &= \sum_{i=0}^{2L} \boldsymbol{\eta}_i^c \left(\boldsymbol{\chi}_{k|k-1}^{x,(i)} - \hat{\mathbf{x}}_{k|k-1}\right)\left(\boldsymbol{\chi}_{k|k-1}^{x,(i)} - \hat{\mathbf{x}}_{k|k-1}\right)^T \end{aligned} \tag{31}$$

Note that the process noise covariance matrix is no longer added to the predicted covariance. The output sigma points are calculated using prior information from the measurement noise sigma-points

$$\boldsymbol{\psi}_{k|k-1}^{(i)} = \mathbf{h}\left(\boldsymbol{\chi}_{k|k-1}^{x,(i)}, \mathbf{u}_k, \boldsymbol{\chi}_{k|k-1}^{v,(i)}\right), \quad i = 0,1,\ldots,2L \tag{32}$$

Then, the predicted output, output covariance matrix, and cross-covariance between the state and output are calculated

$$\hat{\mathbf{y}}_{k|k-1} = \sum_{i=0}^{2L} \boldsymbol{\eta}_i^m \boldsymbol{\psi}_{k|k-1}^{(i)}$$
$$\mathbf{P}_k^{yy} = \sum_{i=0}^{2L} \boldsymbol{\eta}_i^c \left(\boldsymbol{\psi}_{k|k-1}^{(i)} - \hat{\mathbf{y}}_{k|k-1} \right) \left(\boldsymbol{\psi}_{k|k-1}^{(i)} - \hat{\mathbf{y}}_{k|k-1} \right)^T \tag{33}$$
$$\mathbf{P}_k^{xy} = \sum_{i=0}^{2L} \boldsymbol{\eta}_i^c \left(\boldsymbol{\chi}_{k|k-1}^{x,(i)} - \hat{\mathbf{x}}_{k|k-1} \right) \left(\boldsymbol{\psi}_{k|k-1}^{(i)} - \hat{\mathbf{y}}_{k|k-1} \right)^T$$

Note that the measurement noise covariance matrix is no longer added to the output covariance. Now, the Kalman gain can be calculated using (24) and then the *a posteriori* statistics can be calculated with (25). These values are then propagated to the next time step, where they need to be re-augmented with noise states (all zeros) and corresponding covariance matrices, as in (27). Repeat this procedure at each time step of the UKF for the desired number of time steps. This concludes the theoretical explanation of the UKF equations. A more practical outline of the steps of the non-additive noise UKF is provided in Appendix C. Next, an example problem using the UKF is presented.

IV. Example Problem Using Unscented Kalman Filter

This section details an example problem which is approached using the UKF. This example problem is "Computer Exercise 13.21" from (Simon, 2006). This problem is summarized as

$$\mathbf{x}_k = \begin{bmatrix} n_k \\ e_k \\ \dot{n}_k \\ \dot{e}_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n_{k-1} \\ e_{k-1} \\ \dot{n}_{k-1} \\ \dot{e}_{k-1} \end{bmatrix} + \mathbf{w}_k$$
$$\mathbf{y}_k = \begin{bmatrix} \sqrt{(n_k - N_1)^2 + (e_k - E_1)^2} \\ \sqrt{(n_k - N_2)^2 + (e_k - E_2)^2} \end{bmatrix} + \mathbf{v}_k \tag{34}$$

where n_k and e_k are the north and east coordinates of an object, T = 0.1 s is the sampling time, $\mathbf{w}$ is the process noise with covariance $\mathbf{Q}$ = diag(0, 0, 4, 4), $\mathbf{v}$ is the measurement noise with covariance $\mathbf{R}$ = diag(1, 1), and (N_1, E_1) = (20, 0), and (N_2, E_2) = (0, 20) are the locations of tracking stations. The assumed initial conditions of the system are given by

$$\hat{\mathbf{x}}_0 = \begin{bmatrix} 0 & 0 & 50 & 50 \end{bmatrix}^T, \quad \mathbf{P}_0 = \mathbf{I} \tag{35}$$

where $\mathbf{I}$ is an identity matrix. Note that this problem has a linear prediction stage and a nonlinear observation stage with additive noise terms. This filtering problem was solved using both the EKF and UKF. See the provided MATLAB code for this example in Appendix D. Additionally, Appendix E offers MATLAB code for this example in the same side-by-side format as the equations in Appendix B. The state estimation results for this example are shown in Figure 3. In this figure, the state estimates from the EKF and UKF are provided with the true state trajectory on the left, while the state estimation error for EKF and UKF is provided on the right. Note that for this example, little difference is shown between the EKF and UKF, which is likely due to the low level of nonlinear in this system (Rhudy and Gu, 2013). For more simulation examples of the UKF see e.g. (Kandepu et al., 2008). For some experimental application examples of the UKF see e.g. (Rhudy et al., 2013a).

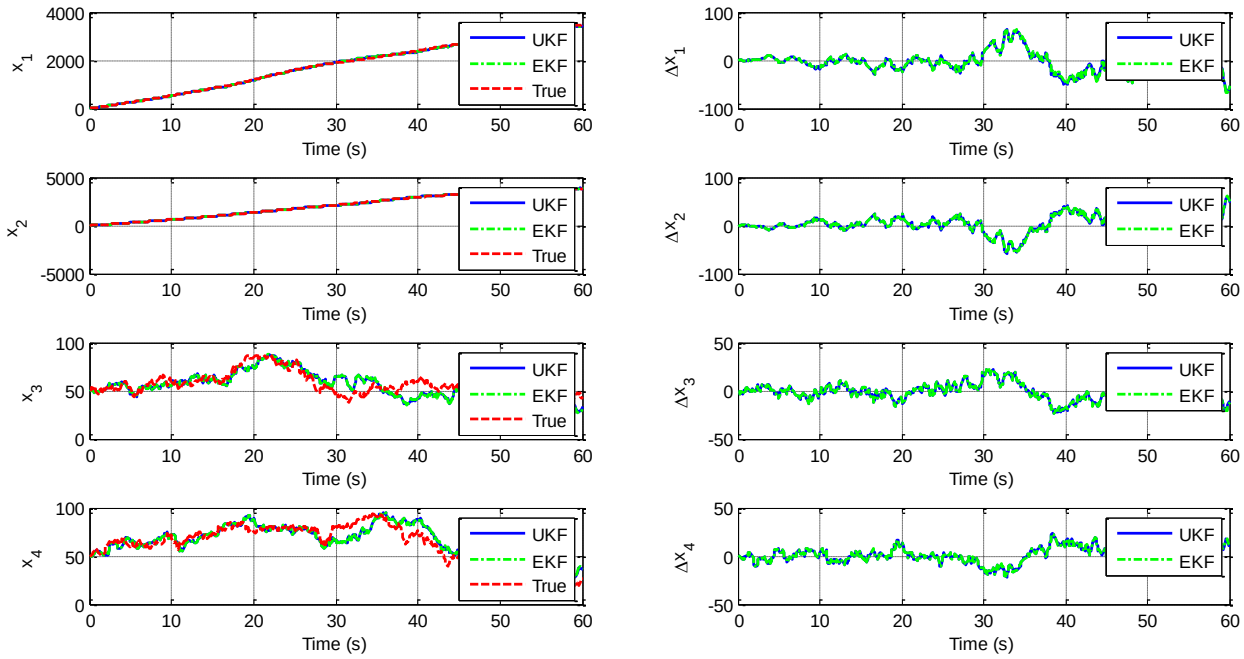

Figure 3. UKF Example 1 State Estimation Results

ACKNOWLEDGEMENT

The authors would like to thank Dr. Jason N. Gross for his help in deriving this work.

REFERENCES

Anderson, B. D. O., and Moore, J. B. (1979), *Optimal Filtering*, Prentice-Hall, NJ, 1979.

Gross, J., Gu, Y., Rhudy, M., Gururajan, S., and Napolitano, M. (2012), "Flight Test Evaluation of GPS/INS Sensor Fusion Algorithms for Attitude Estimation," *IEEE Trans. on Aerospace Electronic Systems*, Vol. 48, No. 3, July 2012, pp. 2128-2139.

Gururajan, S., Fravolini, M. L., Chao, H., Rhudy, M., and Napolitano, M. R. (2013a), "Performance Evaluation of NN based Approached for Airspeed Sensor Failure Accommodation on a Small UAV," accepted to *Mediterranean Conference on Control and Automation*, Platanias-Chania, Crete, Greece, June 25-28, 2013.

Gururajan, S., Rhudy, M., Fravolini, M. L., Chao, H., and Napolitano, M. R. (2013b), "Evaluation of Approaches to Sensor Failure Detection and Accommodation for the Airspeed Sensor on a Small UAV," submitted to *AIAA Guidance Navigation and Control Conference*, Boston, MA, 2013.

Julier, S., and Uhlmann, J. (1997), "A New Extension of the Kalman Filter to Nonlinear Systems (1997)," *Proceedings of SPIE: The International Society for Optical Engineering*, Vol. 3068, April 1997, pp. 182–193.

Kalman, R. E., and Bucy, R. S. (1961), "New Results in Linear Filtering and Prediction Theory," *Journal of Basic Engineering*, Vol. 83, No. 1, 1961, pp. 95–108.

Kandepu, R., Foss, B., and Imsland, L. (2008), "Applying the Unscented Kalman Filter for Nonlinear State Estimation," *Journal of Process Control*, Vol. 18, Nos. 7–8, Aug. 2008, pp. 753–768.

Rhudy, M., Gu, Y., Gross, J., and Napolitano, M. R. (2011), "Evaluation of Matrix Square Root Operations for UKF within a UAV GPS/INS Sensor Fusion Application," *International Journal of Navigation and Observation*, Vol. 2011, Article ID 416828, 11 pages, Dec. 2011. doi:10.1155/2011/416828

Rhudy, M., Gu, Y., Gross, J., Gururajan, S., and Napolitano, M. R. (2013a), "Sensitivity and Robustness Analysis of EKF and UKF Design Parameters for GPS/INS Sensor Fusion," *AIAA Journal of Aerospace Information Systems*, Vol. 10, No. 3, March 2013, pp. 131-143.

Rhudy, M., Gu, Y., and Napolitano, M. R. (2013b), "An Analytical Approach for Comparing Linearization Methods in EKF and UKF," *International Journal of Advanced Robotic Systems*, Vol. 10, No. 208, 2013.

Rhudy, M., Larrabee, T., Chao, H., Gu, Y., and Napolitano, M. R. (2013c), "UAV Attitude, Heading, and Wind Estimation Using GPS/INS and an Air Data System," submitted to *AIAA Guidance Navigation and Control Conference*, Boston, MA, 2013.

Rhudy, M., and Gu, Y. (2013), "EKF versus UKF: Selection of a Nonlinear Kalman Filter," *Interactive Robotics Letters*, West Virginia University, April 2013. Link: http://www2.statler.wvu.edu/~irl/index_files/OnlineJournal.htm

Simon, D. (2006), *Optimal State Estimation*, Wiley, Hoboken, NJ, 2006.

van der Merwe, R. and Wan, E. A. (2001), "The square-root unscented Kalman filter for state and parameter-estimation," *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pp. 3461–3464, Salt Lake City, Utah, USA, May 2001.

van der Merwe, R.,Wan, E., and Julier, S. (2004), "Sigma-Point Kalman Filters for Nonlinear Estimation and Sensor Fusion-Applications to Integrated Navigation," *AIAA Guidance, Navigation and Control Conference*, Providence, RI, 2004.

Wan and van der Merwe (2002), "The Unscented Kalman Filter," Chap. 7 in *Kalman Filtering and Neural Networks*, Wiley, New York, March 2002, pp. 221–282.

Wu, Y., Hu, D., Wu, M., and Hu, X. (2005), "Unscented Kalman Filtering for Additive Noise Case: Augmented versus Nonaugmented," *IEEE Signal Processing Letters*, Vol. 12, No. 5, May 2005.

APPENDIX A: IMPLEMENTATION OF THE ADDITIVE NOISE UKF

Before Executing Filter (complete each step once before filtering)

Step 1: Determine Scaling and Weights

Define the primary, secondary, and tertiary scaling parameters	α, β, κ	(default $\alpha = 1$, $\beta = 2$, $\kappa = 0$)
Define size of state vector	$L = n_x$	(length of state vector)
Calculate scaling parameter	$\lambda = \alpha^2 \left(L + \kappa \right) - L$	
Calculate weight vectors	$\boldsymbol{\eta}_0^m = \lambda / \left(L + \lambda \right)$	
	$\boldsymbol{\eta}_0^c = \lambda / \left(L + \lambda \right) + 1 - \alpha^2 + \beta$	
	$\boldsymbol{\eta}_i^m = \boldsymbol{\eta}_i^c = 1 / \left[2 \left(L + \lambda \right) \right], \quad i = 1, \ldots, 2L$	

Step 2: Determine Noise Assumptions

Define process and measurement noise covariance matrices	$\mathbf{Q}_k = E\left[\mathbf{w}_k \mathbf{w}_k^T \right], \quad \mathbf{R}_k = E\left[\mathbf{v}_k \mathbf{v}_k^T \right]$

Step 3: Initialization

Define initial state and covariance	$\hat{\mathbf{x}}_0 = E\left[\mathbf{x}_0 \right], \quad \mathbf{P}_0 = E\left[\left(\mathbf{x}_0 - \hat{\mathbf{x}}_0 \right) \left(\mathbf{x}_0 - \hat{\mathbf{x}}_0 \right)^T \right]$

Executing the Filter Recursively (perform each step at each discrete-time)

Step 1: Generate the Sigma-Points

Calculate error covariance matrix square root	$\sqrt{\mathbf{P}_{k-1}} = chol\left(\mathbf{P}_{k-1} \right)$ (Lower Cholesky Decomposition)
Calculate the sigma-points	$\boldsymbol{\chi}_{k-1} = \left[\hat{\mathbf{x}}_{k-1} \quad \hat{\mathbf{x}}_{k-1} + \sqrt{L + \lambda} \sqrt{\mathbf{P}_{k-1}} \quad \hat{\mathbf{x}}_{k-1} - \sqrt{L + \lambda} \sqrt{\mathbf{P}_{k-1}} \right]$

Step 2: Prediction Transformation

Propagate each sigma-point through prediction	$\boldsymbol{\chi}_{k\|k-1}^{(i)} = \mathbf{f}\left(\boldsymbol{\chi}_{k-1}^{(i)}, \mathbf{u}_{k-1} \right), \quad i = 0, 1, \ldots, 2L$
Calculate mean of predicted state	$\hat{\mathbf{x}}_{k\|k-1} = \sum_{i=0}^{2L} \boldsymbol{\eta}_i^m \boldsymbol{\chi}_{k\|k-1}^{(i)}$
Calculate covariance of predicted state	$\mathbf{P}_{k\|k-1} = \mathbf{Q}_{k-1} + \sum_{i=0}^{2L} \boldsymbol{\eta}_i^c \left(\boldsymbol{\chi}_{k\|k-1}^{(i)} - \hat{\mathbf{x}}_{k\|k-1} \right) \left(\boldsymbol{\chi}_{k\|k-1}^{(i)} - \hat{\mathbf{x}}_{k\|k-1} \right)^T$

Step 3: Observation Transformation

Propagate each sigma-point through observation	$\boldsymbol{\psi}_{k\|k-1}^{(i)} = \mathbf{h}\left(\boldsymbol{\chi}_{k\|k-1}^{(i)}, \mathbf{u}_k \right), \quad i = 0, 1, \ldots, 2L$
Calculate mean of predicted output	$\hat{\mathbf{y}}_{k\|k-1} = \sum_{i=0}^{2L} \boldsymbol{\eta}_i^m \boldsymbol{\psi}_{k\|k-1}^{(i)}$
Calculate covariance of predicted output	$\mathbf{P}_k^{yy} = \mathbf{R}_k + \sum_{i=0}^{2L} \boldsymbol{\eta}_i^c \left(\boldsymbol{\psi}_{k\|k-1}^{(i)} - \hat{\mathbf{y}}_{k\|k-1} \right) \left(\boldsymbol{\psi}_{k\|k-1}^{(i)} - \hat{\mathbf{y}}_{k\|k-1} \right)^T$
Calculate cross-covariance of state and output	$\mathbf{P}_k^{xy} = \sum_{i=0}^{2L} \boldsymbol{\eta}_i^c \left(\boldsymbol{\chi}_{k\|k-1}^{(i)} - \hat{\mathbf{x}}_{k\|k-1} \right) \left(\boldsymbol{\psi}_{k\|k-1}^{(i)} - \hat{\mathbf{y}}_{k\|k-1} \right)^T$

Step 4: Measurement Update

Calculate Kalman gain	$\mathbf{K}_k = \mathbf{P}_k^{xy} \left(\mathbf{P}_k^{yy} \right)^{-1}$
Update state estimate	$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k\|k-1} + \mathbf{K}_k \left(\mathbf{y}_k - \hat{\mathbf{y}}_{k\|k-1} \right)$
Update error covariance	$\mathbf{P}_k = \mathbf{P}_{k\|k-1} - \mathbf{K}_k \mathbf{P}_k^{yy} \mathbf{K}_k^T$

APPENDIX B: SIDE-BY-SIDE COMPARISON OF EQUATIONS FOR EKF AND UKF

Extended Kalman Filter (EKF)	**Unscented Kalman Filter (UKF)**
Unique Off-Line Calculations (complete each step once before filtering)	
Analytically Calculate Jacobian Matrices $\mathbf{F}_{k-1} = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}_{k-1}}\right\|_{\hat{\mathbf{x}}_{k-1}}, \quad \mathbf{H}_k = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{x}_k}\right\|_{\hat{\mathbf{x}}_{k\|k-1}}$	Define Scaling Parameters and Weight Vectors $\alpha = 1, \quad \beta = 2, \quad \kappa = 0$ $\lambda = \alpha^2 (L + \kappa) - L$ $\boldsymbol{\eta}_0^m = \lambda / (L + \lambda)$ $\boldsymbol{\eta}_0^c = \lambda / (L + \lambda) + 1 - \alpha^2 + \beta$ $\boldsymbol{\eta}_i^m = \boldsymbol{\eta}_i^c = 1 / [2(L + \lambda)], \quad i = 1, \ldots, 2L$
Initialization (complete each step once before filtering)	
$\mathbf{Q}_k = E[\mathbf{w}_k \mathbf{w}_k^T], \quad \mathbf{R}_k = E[\mathbf{v}_k \mathbf{v}_k^T]$ $\hat{\mathbf{x}}_0 = E[\mathbf{x}_0], \quad \mathbf{P}_0 = E[(\mathbf{x}_0 - \hat{\mathbf{x}}_0)(\mathbf{x}_0 - \hat{\mathbf{x}}_0)^T]$	
Prediction (perform each step at each discrete-time)	
	$\sqrt{\mathbf{P}_{k-1}} = chol(\mathbf{P}_{k-1})$
	$\boldsymbol{\chi}_{k-1} = [\hat{\mathbf{x}}_{k-1} \quad \hat{\mathbf{x}}_{k-1} + \sqrt{L+\lambda}\sqrt{\mathbf{P}_{k-1}} \quad \hat{\mathbf{x}}_{k-1} - \sqrt{L+\lambda}\sqrt{\mathbf{P}_{k-1}}]$
	$\boldsymbol{\chi}_{k\|k-1}^{(i)} = \mathbf{f}(\boldsymbol{\chi}_{k-1}^{(i)}, \mathbf{u}_{k-1}), \quad i = 0, 1, \ldots, 2L$
$\hat{\mathbf{x}}_{k\|k-1} = \mathbf{f}(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1})$	$\hat{\mathbf{x}}_{k\|k-1} = \sum_{i=0}^{2L} \boldsymbol{\eta}_i^m \boldsymbol{\chi}_{k\|k-1}^{(i)}$
$\mathbf{P}_{k\|k-1} = \mathbf{F}_{k-1} \mathbf{P}_{k-1} \mathbf{F}_{k-1}^T + \mathbf{Q}_{k-1}$	$\mathbf{P}_{k\|k-1} = \mathbf{Q}_{k-1} + \sum_{i=0}^{2L} \boldsymbol{\eta}_i^c (\boldsymbol{\chi}_{k\|k-1}^{(i)} - \hat{\mathbf{x}}_{k\|k-1})(\boldsymbol{\chi}_{k\|k-1}^{(i)} - \hat{\mathbf{x}}_{k\|k-1})^T$
Observation (perform each step at each discrete-time)	
	$\boldsymbol{\psi}_{k\|k-1}^{(i)} = \mathbf{h}(\boldsymbol{\chi}_{k\|k-1}^{(i)}, \mathbf{u}_k), \quad i = 0, 1, \ldots, 2L$
$\hat{\mathbf{y}}_{k\|k-1} = \mathbf{h}(\hat{\mathbf{x}}_{k\|k-1})$	$\hat{\mathbf{y}}_{k\|k-1} = \sum_{i=0}^{2L} \boldsymbol{\eta}_i^m \boldsymbol{\psi}_{k\|k-1}^{(i)}$
$\mathbf{P}_k^{yy} = \mathbf{H}_k \mathbf{P}_{k\|k-1} \mathbf{H}_k^T + \mathbf{R}_k$	$\mathbf{P}_k^{yy} = \mathbf{R}_k + \sum_{i=0}^{2L} \boldsymbol{\eta}_i^c (\boldsymbol{\psi}_{k\|k-1}^{(i)} - \hat{\mathbf{y}}_{k\|k-1})(\boldsymbol{\psi}_{k\|k-1}^{(i)} - \hat{\mathbf{y}}_{k\|k-1})^T$
$\mathbf{P}_k^{xy} = \mathbf{P}_{k\|k-1} \mathbf{H}_k^T$	$\mathbf{P}_k^{xy} = \sum_{i=0}^{2L} \boldsymbol{\eta}_i^c (\boldsymbol{\chi}_{k\|k-1}^{(i)} - \hat{\mathbf{x}}_{k\|k-1})(\boldsymbol{\psi}_{k\|k-1}^{(i)} - \hat{\mathbf{y}}_{k\|k-1})^T$
Measurement Update (perform each step at each discrete-time)	
$\mathbf{K}_k = \mathbf{P}_k^{xy} (\mathbf{P}_k^{yy})^{-1}$ $\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k\|k-1} + \mathbf{K}_k (\mathbf{y}_k - \hat{\mathbf{y}}_{k\|k-1})$ $\mathbf{P}_k = \mathbf{P}_{k\|k-1} - \mathbf{K}_k \mathbf{P}_k^{yy} \mathbf{K}_k^T$	

APPENDIX C: IMPLEMENTATION OF THE NON-ADDITIVE NOISE UKF

Before Executing Filter (complete each step once before filtering)

Step 1: Determine Scaling and Weights

Define the primary, secondary, and tertiary scaling parameters	α, β, κ (default $\alpha = 1, \beta = 2, \kappa = 0$)
Define size of state vector	$L = n_x + n_w + n_v$
Calculate scaling parameter	$\lambda = \alpha^2 \left(L + \kappa \right) - L$
Calculate weight vectors	$\boldsymbol{\eta}_0^m = \lambda / \left(L + \lambda \right), \quad \boldsymbol{\eta}_0^c = \lambda / \left(L + \lambda \right) + 1 - \alpha^2 + \beta,$ $\boldsymbol{\eta}_i^m = \boldsymbol{\eta}_i^c = 1 / \left[2 \left(L + \lambda \right) \right], \quad i = 1, \ldots, 2L$

Step 2: Determine Noise Assumptions

Define process and measurement noise covariance matrices	$\mathbf{Q}_k = E\left[\mathbf{w}_k \mathbf{w}_k^T \right], \quad \mathbf{R}_k = E\left[\mathbf{v}_k \mathbf{v}_k^T \right]$

Step 3: Initialization

Define initial state and covariance	$\hat{\mathbf{x}}_0 = E\left[\mathbf{x}_0 \right], \quad \mathbf{P}_0 = E\left[\left(\mathbf{x}_0 - \hat{\mathbf{x}}_0 \right) \left(\mathbf{x}_0 - \hat{\mathbf{x}}_0 \right)^T \right]$

Executing the Filter Recursively (perform each step at each discrete-time)

Step 1: Generate the Sigma-Points

Augment state vector with zeros	$\hat{\mathbf{x}}_{k-1}^a = \left[\hat{\mathbf{x}}_{k-1}^T \quad \mathbf{0}_{1 \times n_w} \quad \mathbf{0}_{1 \times n_v} \right]^T$
Calculate error covariance matrix square root	$\sqrt{\mathbf{P}_{k-1}} = chol\left(\mathbf{P}_{k-1} \right)$ (Lower Cholesky Decomposition)
Calculate square root of noise covariance	$\sqrt{\mathbf{Q}_{k-1}} = chol\left(\mathbf{Q}_{k-1} \right), \quad \sqrt{\mathbf{R}_k} = chol\left(\mathbf{R}_k \right)$
Augment square root of error covariance	$\sqrt{\mathbf{P}_{k-1}^a} = blkdiag\left(\sqrt{\mathbf{P}_{k-1}}, \sqrt{\mathbf{Q}_{k-1}}, \sqrt{\mathbf{R}_k} \right)$
Calculate augmented sigma-points	$\boldsymbol{\chi}_{k-1}^a = \left[\hat{\mathbf{x}}_{k-1}^a \quad \hat{\mathbf{x}}_{k-1}^a + \sqrt{L + \lambda} \sqrt{\mathbf{P}_{k-1}^a} \quad \hat{\mathbf{x}}_{k-1}^a - \sqrt{L + \lambda} \sqrt{\mathbf{P}_{k-1}^a} \right]$

Step 2: Prediction Transformation

Propagate each sigma-point through prediction	$\boldsymbol{\chi}_{k|k-1}^{x,(i)} = \mathbf{f}\left(\boldsymbol{\chi}_{k-1}^{x,(i)}, \mathbf{u}_{k-1}, \boldsymbol{\chi}_{k-1}^{w,(i)} \right), \quad i = 0, 1, \ldots, 2L$
Calculate mean of predicted state	$\hat{\mathbf{x}}_{k|k-1} = \sum_{i=0}^{2L} \boldsymbol{\eta}_i^m \boldsymbol{\chi}_{k|k-1}^{x,(i)}$
Calculate covariance of predicted state	$\mathbf{P}_{k|k-1} = \sum_{i=0}^{2L} \boldsymbol{\eta}_i^c \left(\boldsymbol{\chi}_{k|k-1}^{x,(i)} - \hat{\mathbf{x}}_{k|k-1} \right) \left(\boldsymbol{\chi}_{k|k-1}^{x,(i)} - \hat{\mathbf{x}}_{k|k-1} \right)^T$

Step 3: Observation Transformation

Propagate each sigma-point through observation	$\boldsymbol{\psi}_{k|k-1}^{(i)} = \mathbf{h}\left(\boldsymbol{\chi}_{k|k-1}^{x,(i)}, \mathbf{u}_k, \boldsymbol{\chi}_{k|k-1}^{v,(i)} \right), \quad i = 0, 1, \ldots, 2L$
Calculate mean of predicted output	$\hat{\mathbf{y}}_{k|k-1} = \sum_{i=0}^{2L} \boldsymbol{\eta}_i^m \boldsymbol{\psi}_{k|k-1}^{(i)}$
Calculate covariance of predicted output	$\mathbf{P}_k^{yy} = \sum_{i=0}^{2L} \boldsymbol{\eta}_i^c \left(\boldsymbol{\psi}_{k|k-1}^{(i)} - \hat{\mathbf{y}}_{k|k-1} \right) \left(\boldsymbol{\psi}_{k|k-1}^{(i)} - \hat{\mathbf{y}}_{k|k-1} \right)^T$
Calculate cross-covariance of state and output	$\mathbf{P}_k^{xy} = \sum_{i=0}^{2L} \boldsymbol{\eta}_i^c \left(\boldsymbol{\chi}_{k|k-1}^{x,(i)} - \hat{\mathbf{x}}_{k|k-1} \right) \left(\boldsymbol{\psi}_{k|k-1}^{(i)} - \hat{\mathbf{y}}_{k|k-1} \right)^T$

Step 4: Measurement Update

Calculate Kalman gain	$\mathbf{K}_k = \mathbf{P}_k^{xy} \left(\mathbf{P}_k^{yy} \right)^{-1}$
Update state estimate	$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \left(\mathbf{y}_k - \hat{\mathbf{y}}_{k|k-1} \right)$
Update error covariance	$\mathbf{P}_k = \mathbf{P}_{k|k-1} - \mathbf{K}_k \mathbf{P}_k^{yy} \mathbf{K}_k^T$

APPENDIX D: MATLAB CODE FOR EXAMPLE

```
clear all
%% Before filter execution
% System properties
T = 0.1;          % Sampling time
N = 600;          % Number of time steps for filter
N1 = 20;          % Station 1 North coordinate
E1 = 0;           % Station 1 East coordinate
N2 = 0;           % Station 2 North coordinate
E2 = 20;          % Station 2 East coordinate

% Step 1:  Define UT Scaling parameters and weight vectors
L = 4;            % Size of state vector
alpha = 1;        % Primary scaling parameter
beta = 2;         % Secondary scaling parameter (Gaussian assumption)
kappa = 0;        % Tertiary scaling parameter
lambda = alpha^2*(L+kappa) - L;
wm = ones(2*L + 1,1)*1/(2*(L+lambda));
wc = wm;
wm(1) = lambda/(lambda+L);
wc(1) = lambda/(lambda+L) + 1 - alpha^2 + beta;

% Step 2:  Define noise assumptions
Q = diag([0 0 4 4]);
R = diag([1 1]);

% Step 3:  Initialize state and covariance
x = zeros(4, N);             % Initialize size of state estimate for all k
x(:,1) = [0; 0; 50; 50];     % Set initial state estimate
P0 = eye(4,4);            % Set initial error covariance

% Simulation Only:  Calculate true state trajectory for comparison
% Also calculate measurement vector
w = sqrt(Q)*randn(4, N);     % Generate random process noise (from assumed Q)
v = sqrt(R)*randn(2, N);     % Generate random measurement noise (from assumed R)
xt = zeros(4, N);            % Initialize size of true state for all k
xt(:,1) = [0; 0; 50; 50] + sqrt(P0)*randn(4,1);   % Set true initial state
yt = zeros(2, N);            % Initialize size of output vector for all k
for k = 2:N
    xt(:,k) = [1 0 T 0; 0 1 0 T; 0 0 1 0; 0 0 0 1]*xt(:,k-1) + w(:,k-1);
    yt(:,k) = [sqrt((xt(1,k)-N1)^2 + (xt(2,k)-E1)^2); ...
        sqrt((xt(1,k)-N2)^2 + (xt(2,k)-E2)^2)] + v(:,k);
end

%% Initialize and run EKF for comparison
xe = zeros(4,N);
xe(:,1) = x(:,1);
P = P0;
F = [1 0 T 0; 0 1 0 T; 0 0 1 0; 0 0 0 1];   % Linear prediction
for k = 2:N
    % Prediction
    x_m = F*xe(:,k-1);
    P_m = F*P*F' + Q;

    % Observation
    y_m = [sqrt((x_m(1)-N1).^2 + (x_m(2)-E1).^2); ...
        sqrt((x_m(1)-N2).^2 + (x_m(2)-E2).^2)];
    H = [(x_m(1)-N1)/sqrt((x_m(1)-N1)^2 + (x_m(2)-E1)^2), ...
        (x_m(2)-E1)/sqrt((x_m(1)-N1)^2 + (x_m(2)-E1)^2), 0, 0; ...
        (x_m(1)-N2)/sqrt((x_m(1)-N2)^2 + (x_m(2)-E2)^2), ...
        (x_m(2)-E2)/sqrt((x_m(1)-N2)^2 + (x_m(2)-E2)^2), 0, 0];

    % Measurement Update
    K = P_m*H'/(H*P_m*H' + R);          % Calculate Kalman gain
    xe(:,k) = x_m + K*(yt(:,k) - y_m);  % Update state estimate
    P = (eye(4)-K*H)*P_m;               % Update covariance estimate
end
```

(example code continued on next page…)

```
%% Execute Unscented Kalman Filter
P = P0;     % Set first value of P to the initial P0
for k = 2:N
    % Step 1:  Generate the sigma-points
    sP = chol(P,'lower');    % Calculate square root of error covariance
    % chi_p = "chi previous" = chi(k-1)
    chi_p = [x(:,k-1), x(:,k-1)*ones(1,L)+sqrt(L+lambda)*sP, ...
        x(:,k-1)*ones(1,L)-sqrt(L+lambda)*sP];

    % Step 2:  Prediction Transformation
    % Propagate each sigma-point through prediction
    % chi_m = "chi minus" = chi(k|k-1)
    chi_m = [1 0 T 0; 0 1 0 T; 0 0 1 0; 0 0 0 1]*chi_p;
    x_m = chi_m*wm;          % Calculate mean of predicted state
    % Calculate covariance of predicted state
    P_m = Q;
    for i = 1:2*L+1
        P_m = P_m + wc(i)*(chi_m(:,i) - x_m)*(chi_m(:,i) - x_m)';
    end

    % Step 3:  Observation Transformation
    % Propagate each sigma-point through observation
    psi_m = [sqrt((chi_m(1,:)-N1).^2 + (chi_m(2,:)-E1).^2); ...
        sqrt((chi_m(1,:)-N2).^2 + (chi_m(2,:)-E2).^2)];
    y_m = psi_m*wm;          % Calculate mean of predicted output
    % Calculate covariance of predicted output
    % and cross-covariance between state and output
    Pyy = R;
    Pxy = zeros(L,2);
    for i = 1:2*L+1
        Pyy = Pyy + wc(i)*(psi_m(:,i) - y_m)*(psi_m(:,i) - y_m)';
        Pxy = Pxy + wc(i)*(chi_m(:,i) - x_m)*(psi_m(:,i) - y_m)';
    end

    % Step 4:  Measurement Update
    K = Pxy/Pyy;                        % Calculate Kalman gain
    x(:,k) = x_m + K*(yt(:,k) - y_m);   % Update state estimate
    P = P_m - K*Pyy*K';                 % Update covariance estimate
end

%% Display results
figure(1);
t = T*(1:N);
for i = 1:4
    subplot(4,2,2*i-1); plot(t,x(i,:),'b-', t,xe(i,:),'g-.', t,xt(i,:),'r--', 'LineWidth', 2);
    xlabel('Time (s)'); ylabel(['x_',num2str(i)]); grid on; legend('UKF','EKF','True');
    subplot(4,2,2*i); plot(t,x(i,:)-xt(i,:),'b-', t,xe(i,:)-xt(i,:),'g-.', 'LineWidth', 2);
    xlabel('Time (s)'); ylabel(['\Deltax_',num2str(i)]); grid on; legend('UKF','EKF');
end
```

(…example code continued from previous page)

APPENDIX E: SIDE-BY-SIDE COMPARISON OF MATLAB CODE FOR EKF AND UKF

Extended Kalman Filter (EKF)	Unscented Kalman Filter (UKF)
Unique Off-Line Calculations (complete each step once before filtering)	

(Calculate analytical equations for F and H off-line so that they can be hard-coded as below)	`alpha = 1;  % Primary scaling parameter`
	`beta = 2;  % Secondary scaling parameter`
	`kappa = 0;  % Tertiary scaling parameter`
	`lambda = alpha^2*(L+kappa) - L;`
	`wm = ones(2*L + 1,1)*1/(2*(L+lambda));`
	`wc = wm;`
	`wm(1) = lambda/(lambda+L);`
	`wc(1) = lambda/(lambda+L) + 1 - alpha^2 + beta;`

Initialization (complete each step once before filtering)	

`Q = diag([0 0 4 4]);`
`R = diag([1 1]);`
`x = zeros(4, N);  % Initialize size of state estimate for all k`
`x(:,1) = [0; 0; 50; 50];  % Set initial state estimate`
`P0 = eye(4,4);  % Set initial error covariance`	

Prediction (perform each step at each discrete-time)	
`F = [1 0 T 0; 0 1 0 T; 0 0 1 0; 0 0 0 1];`	`sP = chol(P,'lower');`

	`chi_p = [x(:,k-1), ...`
	`  x(:,k-1)*ones(1,L)+sqrt(L+lambda)*sP, ...`
	`  x(:,k-1)*ones(1,L)-sqrt(L+lambda)*sP];`

	`chi_m=[1 0 T 0; 0 1 0 T; 0 0 1 0; 0 0 0 1]` `  *chi_p;`
`x_m = F*x(:,k-1);`	`x_m = chi_m*wm`

`P_m = F*P*F' + Q;`	`P_m = Q;`
	`for i = 1:2*L+1`
	`  P_m = P_m + wc(i)*(chi_m(:,i) - x_m)`
	`  *(chi_m(:,i) - x_m)';`
	`end`

Observation (perform each step at each discrete-time)	

`H = [(x_m(1)-N1)/sqrt((x_m(1)-N1)^2 + ...`
`  (x_m(2)-E1)^2), ...`
`  (x_m(2)-E1)/sqrt((x_m(1)-N1)^2 + ...`
`  (x_m(2)-E1)^2), 0, 0; ...`
`  (x_m(1)-N2)/sqrt((x_m(1)-N2)^2 + ...`
`  (x_m(2)-E2)^2), ...`
`  (x_m(2)-E2)/sqrt((x_m(1)-N2)^2 + ...`
`  (x_m(2)-E2)^2), 0, 0];`
	`psi_m = [sqrt((chi_m(1,:)-N1).^2 + ...`
	`  (chi_m(2,:)-E1).^2); ...`
	`  sqrt((chi_m(1,:)-N2).^2 + ...`
	`  (chi_m(2,:)-E2).^2)];`

`y_m = [sqrt((x_m(1)-N1).^2 + ...`
`  (x_m(2)-E1).^2); ...`
`  sqrt((x_m(1)-N2).^2 + ...`
`  (x_m(2)-E2).^2)];`
	`y_m = psi_m*wm;`

`Pyy = H*P_m*H' + R;`	`Pyy = R;`
	`for i = 1:2*L+1`
	`  Pyy = Pyy + wc(i)*(psi_m(:,i) - y_m)`
	`  *(psi_m(:,i) - y_m)';`
	`end`

`Pyy = P_m*H';`	`Pxy = zeros(L,2);`
	`for i = 1:2*L+1`
	`  Pxy = Pxy + wc(i)*(chi_m(:,i) - x_m)`
	`  *(psi_m(:,i) - y_m)';`
	`end`

Measurement Update (perform each step at each discrete-time)	

`K = Pxy/Pyy;  % Calculate Kalman gain`
`x(:,k) = x_m + K*(yt(:,k) - y_m);  % Update state estimate`
`P = P_m - K*Pyy*K';  % Update covariance estimate`	

DOCUMENT HISTORY

Initial Manuscript Submission: June 07, 2013
Initial Review Completed (V1.0 Upload): June 28, 2013