2010-12

Analysis and comparison of extended and unscented Kalman filtering methods for spacecraft attitude determination

Diaz, Orlando X.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/5010

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

ANALYSIS AND COMPARISON OF EXTENDED AND UNSCENTED KALMAN FILTERING METHODS FOR SPACECRAFT ATTITUDE DETERMINATION

by

Orlando X. Diaz

December 2010

Thesis Advisor:	Marcello Romano
Second Reader:	Hyun-wook Woo

Approved for public release; distribution unlimited

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2010	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: Analysis and Comparison of Extended and Unscented Kalman Filtering Methods for Spacecraft Attitude Determination	**5. FUNDING NUMBERS**
6. AUTHOR(S) Orlando X. Diaz	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number: N/A..

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Two methods of estimating the attitude position of a spacecraft are examined in this thesis: the extended Kalman filter (EKF) and the unscented Kalman filter (UKF). In particular, the UnScented QUaternion Estimator (USQUE) derived from [4] is implemented into a spacecraft model. For generalizations about the each of the filters, a simple problem is initially solved. These solutions display typical characteristics of each filter type. The UKF is very attractive in spacecraft attitude estimation, given that spacecraft dynamics are highly nonlinear. For nonlinear systems, the UKF is of particular interest because it uses a carefully selected set of sample points that more accurately map the probability distribution than the linearization of the standard extended Kalman filter. This leads to faster convergence of the attitude solution from largely inaccurate initial conditions. The filter created in this thesis is formulated based on Markley and Crassidis's work on standard attitude-vector measurements using a gyro-based model for attitude propagation. From the standard attitude vector measurements, the global attitude parameterization is found and given by a quaternion, while a generalized three-dimensional attitude representation is used to define the local attitude error. The multiplicative quaternion-error is then found from the local error. The simulation results indicate that the unscented filter is more robust than the extended Kalman filter.

14. SUBJECT TERMS Kalman Filter, Attitude Determination, Nano-Satellite, IMU, Gyroscope, Magnetometer, Extended Kalman Filter, Unscented Kalman Filter, UnScented QUaternion Estimator, USQUE, MEKF, EKF, UKF, ADCS	**15. NUMBER OF PAGES** 135
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

ANALYSIS AND COMPARISON OF EXTENDED AND UNSCENTED KALMAN FILTERING METHODS FOR SPACECRAFT ATTITUDE DETERMINATION

Orlando X. Diaz
Civilian, National Aeronautics and Space Administration
B.S. Aerospace Engineering, The University of Texas at Austin, 2004

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ASTRONAUTICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2010

Author:	Orlando X. Diaz
Approved by:	Marcello Romano Thesis Advisor
	Hyun-wook Woo Second Reader
	Knox Millsaps Chairman, Department of Mechanical and Aerospace Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Two methods of estimating the attitude position of a spacecraft are examined in this thesis: the extended Kalman filter (EKF) and the unscented Kalman filter (UKF). In particular, the UnScented QUaternion Estimator (USQUE) derived from [4] is implemented into a spacecraft model. For generalizations about the each of the filters, a simple problem is initially solved. These solutions display typical characteristics of each filter type. The UKF is very attractive in spacecraft attitude estimation, given that spacecraft dynamics are highly nonlinear. For nonlinear systems, the UKF is of particular interest because it uses a carefully selected set of sample points that more accurately map the probability distribution than the linearization of the standard extended Kalman filter. This leads to faster convergence of the attitude solution from largely inaccurate initial conditions. The filter created in this thesis is formulated based on Markley and Crassidis's work on standard attitude-vector measurements using a gyro-based model for attitude propagation. From the standard attitude vector measurements, the global attitude parameterization is found and given by a quaternion, while a generalized three-dimensional attitude representation is used to define the local attitude error. The multiplicative quaternion-error is then found from the local error. The simulation results indicate that the unscented filter is more robust than the extended Kalman filter.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

1-U	Refers the 1-CubeSat standard of 10 cm x 10 cm x 10 cm
3-U	Refers to the 3-CubeSat standard of 10 cm x 10 cm x 30 cm
ADCS	Attitude Determination and Control System
ADS	Attitude Determination System
CanX	The Canadian Advanced Nanospace eXperiment
EKF	Extended Kalman Filter
GPS	Global Positioning System
IMU	Intertial Measurement Unit
NACL	Nanosatellite Advanced Concepts Laboratory
NASA	National Aeronautics and Space Administration
NPS	The Naval Postgraduate School
ORS	Office of Responsive Space
QUEST	QUaternion EStimator
RAX	Radio Aurora Explorer
UKF	Unscented Kalman Filter
USQUE	UnScented QUaternion Estimator

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

The completion of this program would not have been possible without the help of others. I would like to acknowledge the National Aeronautics and Space Administration, particularly NASA Ames Research Center Director Dr. S. Pete Worden and Engineering Director Pete Klupar, for supporting me financially and ensuring that my education gets put to good use when I return.

I would also like to thank the following individuals for their invaluable assistance in the completion of this thesis:

Professor Dr. Marcello Romano for allowing me to be part of his research.

Dr. Hyun-wook Woo, without your help and support I would have progressed endlessly without direction. Your help was greatly appreciated.

Ms. Karen Andersen for helping me find this wonderful opportunity of a free education and navigating the possibly endless road of government politics. This simply would not have been possible without your dedication and perseverance.

Ms. Trang Luong for her support and patience during these two years while commuting between San Jose and Monterey, always reminding me what's most important in life, each other. The future always seems brighter when you can see yourself in it with someone special.

My classmates, Space Systems Engineering and Operations Students 2010, for allowing me to be a part of the military community for these short years. The long nights of studying are not so terrible when you are in the company of heroes. A bit cliché, but true. I hope all the best for your future endeavors.

Lastly, to Avilio Peña and Juan Garces Diaz who both passed away during my time here, for teaching me the meaning of service and the importance of education at an early age.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

In August of 1960, the United States Air Force and the Central Intelligence Agency successfully launched the world's first reconnaissance satellite, Corona. The imaging resolution was 8 meters and taken on film. The program lasted for 12 years, and ushered in the era of space-based reconnaissance and intelligence gathering that would be iconic of the Cold War. Since the beginning of spacecraft building, organizations have prized themselves on pushing the envelopes of technology. Programs such as NASA's Explorer, TIROS, and Pioneer later proved that the U.S. was investing heavily on space technologies.

I

Figure 1. Dr. William Pickering, Dr. James Van Allen, and Dr. Wernher Von Braun hold a model of the Explorer 1 vehicle above their heads. Credit: NASA

Historically, the building of spacecraft has been a lengthy process, often taking many years or even decades. Recently, however, a new methodology for building spacecraft has transpired. Organizations such as The Office of Responsive Space (ORS) have been created to change this expensive and lengthy process into one that focuses on providing a "good enough" service in a timely manner [1]. This push for faster programs has also led areas of the industry to build smaller systems in attempts to utilize the leftover-over payload mass of launch vehicles. This is more commonly referred to as a "secondary payload." The industry push combined with the emerging university nanosatellite community has created an influx of new commercialism for space-based hardware.

One of the limiting technologies in the small spacecraft arena is attitude determination and control systems (ADCS). While currently there is an increased interest in this area, a limited number of complete solutions in a 3U or 1U-class nanosatellite have been demonstrated on-orbit. Many proposed solutions are also not affordable to this community. While companies like Boeing, Honeywell, and Sinclair are working on hardware solutions, the problem of attitude determination and control can be attacked from both sides. That is to say, as the hardware is being developed, both academic and commercial institutions can focus their resources on the optimal estimation and control theory problems. The lack of an affordable hardware should not inhibit willing parties to develop solutions and methods for the small spacecraft ADCS problem.

B. SHORT OVERVIEW OF ATTITUDE ESTIMATION

One of the most common estimation techniques that has been widely used for various dynamics systems is the Kalman filter. While the filter was initially designed for linear systems, variations of this filter have been developed in particular for nonlinear systems. The extended Kalman filter can be used on nonlinear systems and is based on linearizing the system dynamics. While this is potentially attractive for the nonlinear spacecraft attitude control problem there are several associated with the nonlinearization [2].

While the EKF has proven to be a popular tool for nonlinear estimation, it continues to endure some fundamental issues inherent in the linearization process, which can be the potential cause of divergence. A later development for nonlinear estimation was developed by Julier and Uhlmann and is called the “unscented” Kalman filter [3].

The UKF is “founded on the intuition that it is easier to approximate a probability distribution than it is to approximate an arbitrary nonlinear function or transformation [3]." The UKF successfully avoids the EKF linearization step by introducing a set of sample points that capture the higher order statics of the system. Finally, the UKF method has been developed to estimate the quaternions associated with the attitude of a spacecraft [4]. The numerical simulations presented in these studies have illustrated the superior performance of the UKF in this context.

The primary goal of this thesis is to develop and verify estimation algorithms and simulation code for a spacecraft attitude determination system (ADS). In particular, the two estimation methods that are compared for determining the attitude are the extended Kalman filter (EKF) and the unscented Kalman filter (UKF). Each filter is evaluated based on error computation time. The inherent linearity and nonlinearity of each type of filter is examined by choosing related problems that highlight issues in trying to use a linear filter (EKF) to solve a nonlinear problem. To do this, two separate simulations codes were designed. These simulation codes include an accurate spacecraft model where torque disturbances, Earth physics, and orbital mechanics are accounted for, as well as sensor models of an inertial measurement unit and magnetometer.

A simplified problem was used to verify the behavior of both estimation methods on linear and nonlinear dynamics. For this, the simple pendulum was used as a way to show how each filter can be used to estimate the states of a given dynamic problem. After this problem was worked, these filters were used as analogs against a simulated spacecraft model. Primarily focusing on the UKF, this thesis discusses the differences between the two filters and focuses on the benefits of using nonlinear estimation. It is widely known that there are many benefits to nonlinear estimation. The UKF is very attractive in spacecraft attitude estimation, given that spacecraft dynamics are highly nonlinear. This thesis highlights these benefits while solving both the EKF and UKF spacecraft attitude estimation problem. While previous theses discussed the nuances of characterizing these types of sensors for inclusion in the simulation [4], this paper will focus on the estimation methods as they apply to attitude determination.

C. RECENT CUBESAT ADCS SYSTEMS

Several spacecraft that have implemented ADCS systems into the small CubeSat standard. This section discusses three of these, which include the following:

- Canadian Advanced Nanospace eXperiment (CanX) –The University of Toronto Institute for Aerospace Studies Space Flight Laboratory (UTIAS SLF)
- AISSat-1 – The University of Toronto Institute for Aerospace Studies Space Flight Laboratory (UTIAS SLF)
- Radio Aurora eXperiment (RAX) – The University of Michigan

1. Canadian Advanced Nanospace eXperiment (CanX) ADS

The Canadian Advanced Nanospace eXperiment (CanX) program is run by the University of Toronto Institute for Aerospace Studies (UTIAS) Space Flight Laboratory. CanX-1 launched on June 2003 from Plesetsk, Russia, and was a 1-U CubeSat that consisted of several ADS hardware components. The primary mission of CanX-1 was to demonstrate the several experimental ADS components. The CanX-1 ADS package consisted of a CMOS Imager for ground-controlled horizon sensing and star tracking, active three-axis magnetic stabilization and a Global Positioning System (GPS) receiver that was modified to work in low Earth orbit. Figure 2 shows a picture of the CanX CMOS imager used for star tracking [6].

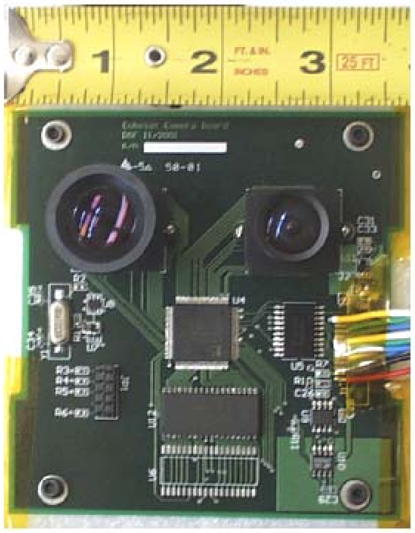


Figure 2. CanX-1 Agilent Technologies CMOS Imager

CanX-2, which launched in April 2008, uses many of the same types of ADS systems. The CanX-2 ADS uses a suite of sun sensors and a three-axis magnetometer. Both CanX-1 and CanX-2 use a standard extended Kalman filter to estimate the attitude of the spacecraft.

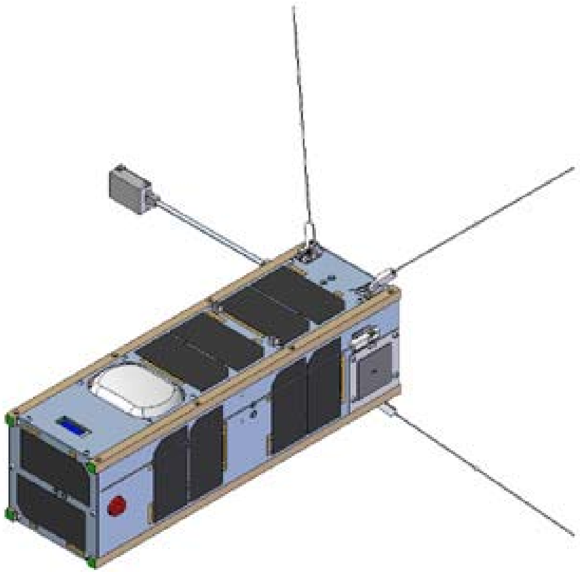


Figure 3. Computer Rendering of CanX-2

2. AISSat-1 ADS

AISSat-1 is a 6-kg Norwegian nanosatellite, being constructed on behalf of government of Norway by UTIAS/SFL, whose primary mission is to investigate the feasibility and performance of a spacecraft-based Automatic Identification System (AIS) sensor in low-Earth orbit as a means of tracking maritime assets. AISSat-1 is intended as both a research and development platform, and a demonstration mission for a larger operational capability.

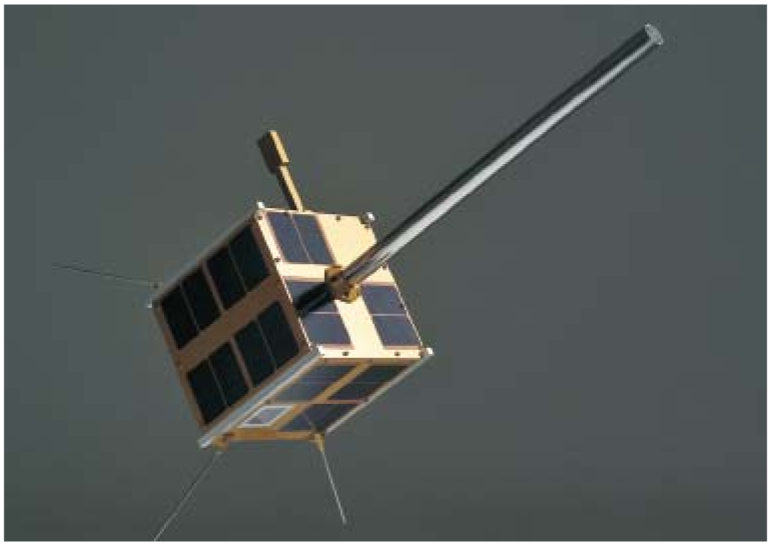


Figure 4. Computer Rendering of AISSat-1, from [7]

A full 3-axis attitude determination and control system provides attitude stabilization and fine pointing for AISSat-1. The satellite is able to point in either and inertial orientations, or an orbit-frame-fixed orientation, including on nadir. Attitude sensors consist of six sun sensors, a magnetometer and rate gyros. Three orthogonally mounted reaction wheels and three magnetorquer coils controls the actuation of the satellite. The magnetorquer is used for de-tumbling and momentum dumping while the reaction wheels provide fine attitude pointing capability. The attitude control system is able to maintain several degree level pointing accuracy and stability over the course of the entire orbit, including eclipse. For attitude estimation, this spacecraft also implemented an extended Kalman filter [7].

3. Radio Aurora Explorer (RAX) ADS

The Radio Aurora Explorer (RAX) spacecraft, currently being developed by The University of Michigan is a 3U CubeSat, which will also implement and attitude determination system. The primary scientific objective of the Radio Aurora Explorer (RAX) mission is to understand the microphysics of plasma instabilities that lead to field-aligned irregularities (FAI) of electron density in the polar lower (80–400 km) ionosphere. For attitude control, an inertial measurement unit in conjunction with sun sensors and magnetometers will observe the time it takes the passive magnetic attitude control system to de-tumble the spacecraft after deployment. This system will implement a continuous-discrete extended Kalman filter. They will implement a 13 state filter, which will consist of 3 position, 3 velocity, 4 quaternions, and 3 angular rates. The team will implement the QUaternion ESTimator (QUEST) method developed by Shuster and Oh [8]. Literature describes the QUEST method as computationally expensive; however, the information will be gathered on orbit and processed on the ground to eliminate computational constraints on the filtering process. Some of the ADS hardware will include six 3-axis magnetometers, nine sun sensors and an inertial measurement unit, which will consist of a 3-axis gyroscope.

THIS PAGE INTENTIONALLY LEFT BLANK

II. GENERAL EXTENDED AND UNSCENTED KALMAN FILTERING METHODS FOR THE ESTIMATION OF DYNAMIC SYSTEMS

A. BACKGROUND

Accurate attitude knowledge is essential for many spacecraft missions. Kalman filtering has been widely known since the 1960s as a method for filtering out noise in a given measurement. Theoretically, the Kalman filter is a sequential optimal estimator for what is called the *linear-quadratic problem*, which is the problem of estimating the instantaneous "state" of a linear dynamic system including its uncertainty--by using measurements linearly related to the state corrupted by white noise [10]. For attitude determination, several types of Kalman filters have been developed over the years. This section describes two basic types of Kalman filters, the extended Kalman filter (EKF) and the unscented Kalman filter (UKF). There have been many technical papers written on Kalman filtering for state estimation [2][3]. This chapter will start the discussion with the continuous-time Kalman filter as a base line. Several textbooks use a variety of nomenclature to describe this estimation process. As a standard, the following tables define the notation used in this thesis. These tables are also consistent with [9] and [10].

Table 1. Standard Symbols of Kalman Filtering, from [10]

Symbols	Symbol Definition
$\boldsymbol{F}$	Dynamic coefficient matrix (state matrix) of a continuous linear differential equation defining a dynamic system
$\boldsymbol{G}$	Coupling matrix between random process noise and the state of a dynamic system
$\boldsymbol{H}$	Measurement sensitivity matrix defining the linear relationship between the state of the dynamic system and measurements that can be made, (also known as a coefficient matrix [9])
$\boldsymbol{K}$	Kalman gain matrix
$\boldsymbol{P}$	Covariance matrix of state estimation uncertainty
$\boldsymbol{Q}$	Covariance matrix of process noise in the system state dynamics also called the process noise covariance
$\boldsymbol{R}$	Covariance matrix of observational (measurement) uncertainty also called the measurement noise covariance
$\boldsymbol{x}$	State vector
$\boldsymbol{y}$	Vector (or scalar) of measured values.
Φ	State transition matrix of a discrete dynamic system

Table 2. Special State Space Notation, from [10]

Symbols	**Symbol Definition**
$x_k(i)$	The *i*-th component of the vector *x*, or the *i*-th element of the sequence. The sub-index *k* refers to the sequence of propagation as it occurs in the filtering process. i.e. *k+1* can be referred to as the "update" term that is determined from the same term calculated previously. $x_{k+1}(i) = x_k(i) + noise$
$\hat{x}$	An estimate of the value of *x*.
$\hat{x}_k^-$	A *priori* estimate of the x_k, conditioned on all prior measurements except the one at time t_k
$\hat{x}_k^+$	A *posteriori* estimate of the x, conditioned on all available measurements at time t_k
$\tilde{y}$	A measurement of some quantity we can estimate to the state vector from.
$\dot{x}$	Derivative of x with respect to time

The Kalman filter uses a parametric characterization of the probability distribution of its estimation errors in determining the optimal filtering gains, and it is the probability distribution that can be used for assessing its performance as a function of the "design parameters" of an estimation system [10]. Some of these can include:

- the types of sensors used,
- the locations and orientations of the various sensor types with respect to the system to be estimated,
- the allowable noise characteristics of the sensors,
- the data sampling rates for the various sensor types, and most importantly,
- the level of model simplification to reduce implementation requirements.

B. CONTINUOUS-TIME EXTENDED KALMAN FILTER

For nonlinear systems, such as spacecraft dynamics, the extended Kalman filter (EKF) has been previously proposed in literature and used on-board many spacecraft [2]. In the EKF, the state transition and observation models do not need to be linear functions representing the state, granted they are differentiable. Given that a vast majority of nonlinear problems can be described with differentiable nonlinear functions, the Continuous-Time EKF can often be used. The Continuous-Time EKF is very similar to the Continuous-Time Linear Kalman filter [9]. The derivation of the Continuous-Time EKF starts with the continuous non-linear system model below:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t) + \mathbf{G}(t)\mathbf{w}(t) \tag{2.1}$$

$$\tilde{\mathbf{y}}(t) = \mathbf{h}(\mathbf{x}(t), t) + \mathbf{v}(t) \tag{2.2}$$

where it is important to note that **f**(**x**(t), **u**(t),t) represents nonlinear continuous function or the state transition model, while **G**(t) and **w**(t) represent the coupling matrix and continuous-time covariance respectively. For Equation 2.2, $\tilde{\mathbf{y}}(t)$ represents the measured nonlinear observed model using a continuous function **h**(x(t),t) plus the continuous-time covariance, **v**(t).

The inherent linearization process can cause the filter to diverge, as the Gaussian input does not necessarily produce a Gaussian output [9]. To continue, we must assume that, for our purposes, a linear representation of our non-linear system will suffice. For example, this method can certainly be used for functions where small angle approximation is valid. Examples of the limitations are discussed in detail in Section F. For the EKF, we must also assume that the true state of the system is sufficiently close to the estimated state. Therefore, the error dynamics can be reasonably approximated by a linearized first order Taylor series expansion. The first order expansion of $\mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t)$ about a nominal state $\bar{\mathbf{x}}(t)$ becomes:

$$\mathbf{f}(\mathbf{x}(t),\mathbf{u}(t),t) \cong \mathbf{f}(\overline{\mathbf{x}}(t),\mathbf{u}(t),t) + \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\overline{\mathbf{x}}(t)} [\mathbf{x}(t) - \overline{\mathbf{x}}(t)] \qquad 2.3$$

where $\mathbf{x}(t)$ is close to $\overline{\mathbf{x}}(t)$. Similarly, the output in Equation 2.3 becomes [9]:

$$\mathbf{h}(\mathbf{x}(t),t) \cong \mathbf{h}(\overline{\mathbf{x}}(t),t) + \left.\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right|_{\overline{\mathbf{x}}(t)} [\mathbf{x}(t) - \overline{\mathbf{x}}(t)] \qquad 2.4$$

Here the EKF solves this problem by calculating the Jacobians of *f* and *h* around the estimated state, which in turn yields a trajectory model function centered around this state. Figure 5 shows this graphically [11].

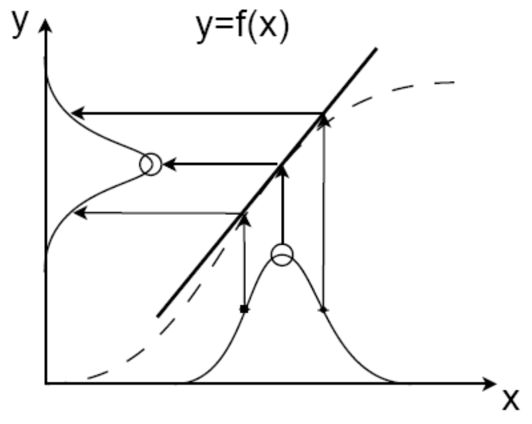


Figure 5. Illustration of Extended Kalman filter linearization of nonlinear function and the related Gaussian distribution.

To find the estimate of the state, the extended Kalman filter continues with assumption made earlier, that $\overline{\mathbf{x}}(t) = \hat{\mathbf{x}}(t)$. Thus, the expectation of both Equations 2.3 and 2.4 gives the following equation, where E represents the conditional mean or expectation [9].

$$E\{\mathbf{f}(\mathbf{x}(t),\mathbf{u}(t),t)\} = \mathbf{f}(\hat{\mathbf{x}}(t),\mathbf{u}(t),t) \qquad 2.5$$

$$E\{\mathbf{h}(\mathbf{x}(t),t)\} = \mathbf{h}(\hat{\mathbf{x}}(t),t) \qquad 2.6$$

Therefore, the extended Kalman filter for the state and output estimate is given by the following two equations [9].

$$\dot{\hat{\mathbf{x}}}(t) = \mathbf{f}(\hat{\mathbf{x}}(t),\mathbf{u}(t),t) + K(t)\left[\tilde{\mathbf{y}}(t) - \mathbf{h}(\hat{\mathbf{x}}(t),t)\right] \qquad 2.7$$

$$\hat{\mathbf{y}}(t) = \mathbf{h}(\hat{\mathbf{x}}(t),t) \qquad 2.8$$

Because the equation of the measurement of the state vector has the same structure as the linear Kalman filter, we can use the covariance expression shown in Table 3. The following table summarizes the equations for the continuous-time extended Kalman filter.

Table 3. Continuous-Time Extended Kalman Filter, from [9]

Model	$\dot{\mathbf{x}}(t)=\mathbf{f}(\mathbf{x}(t),\mathbf{u}(t),t)+\mathbf{G}(t)\mathbf{w}(t),$ $\tilde{\mathbf{y}}(t)=\mathbf{h}(\mathbf{x}(t),t)+\mathbf{v}(t)$
Initialize	$\hat{\mathbf{x}}(t_0)=\hat{\mathbf{x}}_0$ $P_0=E\left\{\tilde{\mathbf{x}}(t_0)\tilde{\mathbf{x}}^T(t_0)\right\}$
Gain	$K(t)=P(t)H^T(\hat{\mathbf{x}}(t),t)R^{-1}(t)$
Covariance	$\dot{P}(t)=F(\hat{\mathbf{x}}(t),t)P(t)+P(t)F^T(\hat{\mathbf{x}}(t),t)$ $-P(t)H^T(\mathbf{x}(t),t)R^{-1}(t)H(\hat{\mathbf{x}}(t),t)P(t)+G(t)Q(t)G^T(t)$ $F(\hat{\mathbf{x}},t)\equiv\left.\frac{\partial\mathbf{f}}{\partial\mathbf{x}}\right\|_{\hat{\mathbf{x}}(t)}\quad,\quad H(\hat{\mathbf{x}},t)\equiv\left.\frac{\partial\mathbf{h}}{\partial\mathbf{x}}\right\|_{\hat{\mathbf{x}}(t)}$
Estimate	$\dot{\hat{\mathbf{x}}}(t)=\mathbf{f}(\hat{\mathbf{x}}(t),\mathbf{u}(t),t)+K(t)\left[\tilde{\mathbf{y}}(t)-\mathbf{h}(\hat{\mathbf{x}}(t),t)\right]$

C. DISCRETE-TIME LINEAR AND EXTENDED KALMAN FILTERS

While understanding the basics of the continuous-time extended Kalman filter is valuable in the sense that it can often be used to solve entire solutions analytically, implementation of this is not practical. In most cases, the control system is responding to different given inputs. The use of real-time processing is inevitable in the practical implementation of estimating dynamic systems. Thus, the continuous-time Kalman filter must be discretized so that it may be applied to iterative methods. This section describes how the Kalman filter is derived.

Derivation of the discrete-time filter and the extended Kalman filter are very similar. To derive the discrete-time Kalman filter, an assumption must be made that both the model and measurement are available in discrete form. Here, we can start with the non-linear "truth" model shown below [9]:

$$\mathbf{x}_{k+1}=\Phi_k\mathbf{x}_k+\Gamma_k\mathbf{u}_k+\Upsilon_k\mathbf{w}_k \qquad 2.9$$

$$\tilde{\mathbf{y}}_k = H_k(\mathbf{x}_k) + \mathbf{v}_k \qquad 2.10$$

where Φ is the state transition matrix, Γ is the control-input matrix that is applied to the control vector u_k, and Υ is the noise matrix. The definition of Φ, Γ, and Υ are shown below.

$$\Phi \equiv e^{F\Delta t} \qquad 2.11$$

$$\Gamma \equiv \left[\int_0^{\Delta t} e^{Ft} dt\right] B \qquad 2.12$$

$$\Upsilon \equiv \left[\int_0^{\Delta t} e^{Ft} dt\right] G \qquad 2.13$$

where *B* and G are the coefficient matrices taken from the continuous system. Again, in Equations 2.9 and 2.10 $w_k(t)$ and $v_k(t)$ are assumed to be zero-mean Gaussian white-noise processes and their covariance's are given by the expectation equations [9]:

$$E\left\{\mathbf{w}_k \mathbf{w}_j^T\right\} = \begin{cases} 0 & k \neq j \\ Q_k & k = j \end{cases} \qquad 2.14$$

$$E\left\{\mathbf{v}_k \mathbf{v}_j^T\right\} = \begin{cases} 0 & k \neq j \\ R_k & k = j \end{cases} \qquad 2.15$$

The Q_k matrix accounts for the state process noise while the R_k matrix accounts for the expected measurement noise. These equations imply that the errors are not correlated forward or backward in time. We can also assume that v_k and w_k are *uncorrelated* so:

$$E\left\{\mathbf{v}_k \mathbf{w}_k^T\right\} = 0 \qquad 2.16$$

Updating the current estimate of the state $\hat{x}_k$ to obtain $\hat{x}_{k+1}$ based upon all k+1 measurement subsets assumes that the gain (K) can vary in time. This propagation can be done using [9]:

$$\hat{\mathbf{x}}_{k+1}^{-} = \Phi_k \hat{\mathbf{x}}_k^{+} + \Gamma_k \mathbf{u}_k \qquad 2.17$$

Furthermore, the updated state is given by:

$$\hat{\mathbf{x}}_k^{+} = \hat{\mathbf{x}}_k^{-} + K_k \left[\tilde{\mathbf{y}}_k - H_k \hat{\mathbf{x}}_k^{-} \right] \qquad 2.18$$

where $\tilde{\mathbf{y}}_k$ is the measurement vector. The gain K_k changes with time properly weighting the relative confidence of the accuracy of the propagated state verses the measured state. To find K_k, first the state error and error covariance matrixes must be defined [9]:

$$P_k \equiv E\left\{ \tilde{\mathbf{x}}_k^{-} \tilde{\mathbf{x}}_k^{-T} \right\} \qquad 2.19$$

where

$$\tilde{\mathbf{x}}_k \equiv \hat{\mathbf{x}}_k^{-} - \mathbf{x}_k \qquad 2.20$$

Substituting Equations 2.9 and 2.17 into Equation 2.20 and substituting the resulting equation into Equation 2.19 leads to:

$$P_{k+1}^{-} = \Phi_k P_k^{+} \Phi_k^T + \Upsilon_k Q_k \Upsilon_k^T \qquad 2.21$$

Because $\mathbf{w}_k$ and $\tilde{\mathbf{x}}_k^{+}$ are uncorrelated the terms $E\left\{ \mathbf{w}_k \tilde{\mathbf{x}}_k^{+T} \right\} = E\left\{ \tilde{\mathbf{x}}_k^{+} \mathbf{w}_k^T \right\} = 0$. To find the updated error covariance matrix, we can use Equations 2.10 and 2.18. Then substitution of the resulting equation into Equation 2.20 leads to:

$$P_k^{+} = \left[I - K_k H_k \left(\hat{\mathbf{x}}_k^{-} \right) \right] P_k^{-} \qquad 2.22$$

To find the gain K, the trace of error covariance matrix P_k^{+} is minimized. Solving gives:

$$K_k = P_k^{-} H_k^T \left(\hat{\mathbf{x}}_k^{-} \right) \left[H_k \left(\hat{\mathbf{x}}_k^{-} \right) P_k^{-} H_k^T \left(\hat{\mathbf{x}}_k^{-} \right) + R_k \right]^{-1} \qquad 2.23$$

As mentioned previously, the extended Kalman filter and discrete-time Kalman filter are nearly identical. The only difference between these two are the initial model equations and the propagation equations. The extended Kalman filter assumes that the model is a continuous function and thus be differentiable. This is clearly evident in Table 4.

Table 4. Discrete-Time Linear Kalman Filter, from [9]

Model	$\mathbf{x}_{k+1} = \Phi_k \mathbf{x}_k + \Gamma_k \mathbf{u}_k + \Upsilon \mathbf{w}_k, \quad \mathbf{w}_k \sim N(\mathbf{0}, Q_k)$ $\tilde{\mathbf{y}}_k = H_k \mathbf{x}_k + \mathbf{v}_k, \quad \mathbf{v}_k \sim N(\mathbf{0}, R_k)$
Initialize	$\hat{\mathbf{x}}(t_0) = \hat{\mathbf{x}}_0$ $P_0 = E\left\{\tilde{\mathbf{x}}(t_0)\tilde{\mathbf{x}}^T(t_0)\right\}$
Gain	$K_k = P_k^- H_k^T(\hat{\mathbf{x}}_k^-)\left[H_k(\hat{\mathbf{x}}_k^-)P_k^- H_k^T(\hat{\mathbf{x}}_k^-) + R_k\right]^{-1}$
Update	$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + K_k\left[\tilde{\mathbf{y}}_k - H_k(\hat{\mathbf{x}}_k^-)\right]$ $P_k^+ = \left[I - K_k H_k\right]P_k^-$
Propagation	$\hat{\mathbf{x}}_{k+1}^- = \Phi_k \hat{\mathbf{x}}_k^+ + \Gamma_k \mathbf{u}_k$ $P_{k+1}^- = \Phi_k P_k^+ \Phi_k^T + \Upsilon_k Q_k \Upsilon_k^T$

D. UNSCENTED KALMAN FILTER

The inherent issue with propagating Gaussian random variables through a nonlinear function can also be approached using a technique described as the *unscented transform*. While the extended Kalman filter has many applications, and is the most popular method for nonlinear estimation to date, the unscented Kalman filter (UKF) was proposed by Julier, Uhlmann, and Durrant-Whyte [12] to overcome the instabilities associated with the EKF. While the EKF typically works well in the regions where the first-order Taylor series linearization adequately approximates the nonlinear probably distribution, a primary area of concern is during the initialization stage, where the estimated initial state can be far from the true state [9]. The UKF typically involves more complex computations than the EKF, but has the following advantages:

1. the expected error is lower than the EKF
2. it can be applied to non-differentiable function
3. it avoids the derivation of Jacobian matrices
4. it is valid to higher-order expansions than the standard EKF [4]

The UKF can be thought of as an extension of the traditional Kalman filter for the estimation of nonlinear systems that implements the unscented transformation. The unscented transformation uses a set of sample, or sigma, points that are determined from the *a priori* mean and covariance of the state. The sigma points undergo the nonlinear transformation. Then the *a posteriori* mean and covariance of the state are determined from the transformed sigma points. This approach gives the UKF better convergence characteristics and greater accuracy than the EKF for nonlinear systems [13]. The ability of the UKF to accurately estimate nonlinearities make it attractive for implementation on spacecraft as the state and observations are inherently nonlinear. This section describes the basic derivation of the unscented Kalman filter, while the subsequent sections describe the implementation of the UKF for attitude determination.

The derivation of the unscented Kalman filter starts by selecting a nonlinear system defined by [9]:

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, k) + G_k \mathbf{w}_k \qquad 2.24$$

$$\tilde{\mathbf{y}}_k = \mathbf{h}(\mathbf{x}_k, k) + \mathbf{v}_k \qquad 2.25$$

where x_k is the n × 1 state vector and y_k is the m × 1 measurement vector. It is interesting to note that a continuous-time model can also be expressed in the form of Equation 2.24. Similar to the previous derivations, v_k represents the measurement-error noise while w_k describes the white Gaussian process noise with covariances given by

$$E\left\{\mathbf{w}(t)\mathbf{w}^T(\tau)\right\} = Q(t)\delta(t-\tau) \qquad 2.26$$

$$E\left\{\mathbf{w}(t)\mathbf{v}^T(\tau)\right\} = R(t)\delta(t-\tau) \qquad 2.27$$

$$E\left\{\mathbf{v}(t)\mathbf{w}^T(\tau)\right\} = 0 \qquad 2.28$$

The covariance matrices of each of these are given by Q_k and R_k respectively [4]. The Kalman filter update equations are rewritten from Table 4 as [3]:

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + K_k\upsilon_k \qquad 2.29$$

$$P_k^+ = P_k^- - K_k P_k^{\upsilon\upsilon} K_k^T \qquad 2.30$$

where υ_k is the *innovations process*, given by

$$\upsilon_k \equiv \tilde{\mathbf{y}}_k - \hat{\mathbf{y}}_k^- = \tilde{\mathbf{y}}_k - \mathbf{h}(\hat{\mathbf{x}}_k^-, \mathbf{u}_k k) \qquad 2.31$$

The covariance of the *innovations process*, υ_k is given by $P_k^{\upsilon\upsilon}$ [4].

$$P_{k+1}^{\upsilon\upsilon} = P_{k+1}^{yy} + R_{k+1} \qquad 2.32$$

The Kalman gain is computed by the following equation [4].

$$K_k = P_k^{xy}(P_k^{\upsilon\upsilon})^{-1} \qquad 2.33$$

where P_k^{xy} is the cross-correlation matrix between $\hat{\mathbf{x}}_k^-$, and $\hat{\mathbf{y}}_k^-$. The cross-correlation is defined later in the discussion below. To define the propagation equations, the following sigma points must be computed [4]. The filter starts by augmenting the state vector to *L* dimensions in the original state-vector, model noise, and measurement noise where *L* is the size of the vector $\mathbf{x}_k^a$, or the *augmented state* defined by Equation 2.37 [9]. The covariance matrix is similarly augmented and this forms the augmented state estimate vector shown below.

$$\sigma \leftarrow 2L \ \ columns \ \ from \ \ \pm\gamma\sqrt{P_k^a} \qquad 2.34$$

$$\chi_k^a(0) = \hat{\mathbf{x}}_k^a \qquad 2.35$$

$$\chi_k^a(i) = \sigma_k(i) + \hat{\mathbf{x}}_k^a \qquad 2.36$$

where $\hat{\mathbf{x}}_k^a$ is an augmented state defined by [4]

$$\mathbf{x}_k^a = \begin{bmatrix} x_k \\ w_k \\ v_k \end{bmatrix}, \quad \hat{\mathbf{x}}_k^a = \begin{bmatrix} \hat{x}_k \\ 0_{q\times 1} \\ 0_{m\times 1} \end{bmatrix} \qquad 2.37$$

Augmenting the covariance requires the computation of *2(q+l)* additional sigma points. It is important to mention here that *q* is the dimension of w_k, *l* is the dimension of v_k, and m is the output dimension. While L is the size of the vector $\hat{\mathbf{x}}_k^a$, the parameter γ is given by the following [4].

$$\gamma = \sqrt{L+\lambda} \qquad 2.38$$

and the composite scaling parameter , λ, is given by

$$\lambda = \alpha^2(L+\kappa) - L \qquad 2.39$$

The constant α, represents the spread of sigma points and is usually set to a small positive value (e.g., $1\times 10^{-4} \le \alpha \le 1$). There are 2L values for σ_k, each representing the positive and negative values of the square root. The Cholesky method is often used to find the square root of a matrix. Similar to the EKF, the UKF now propagates these sigma-points from a Gaussian distribution through a nonlinear function, and recreates a Gaussian distribution by calculating the mean and covariance of these results [11].

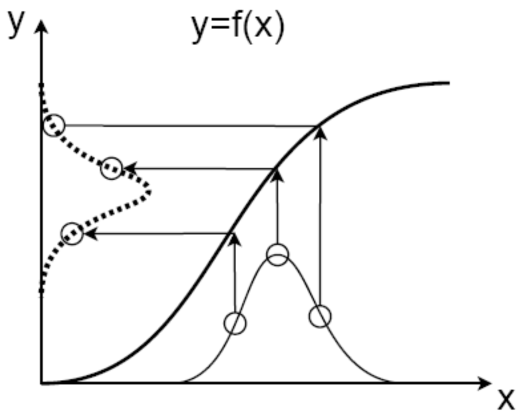


Figure 6. Illustration of the unscented Kalman filter sigma-points propagation

These sigma points are evaluated by:

$$\chi_{k+1}(i) = \mathbf{f}(\chi_k^x(i), \chi_k^w(i), \mathbf{u}_k, k) \qquad 2.40$$

where $\chi_k^x(i)$ is a vector of the first *n* elements of $\chi_k^a(i)$, and $\chi_k^w(i)$ is a vector of the next *q* elements of $\chi_k^a(i)$, with

$$\chi_k^a = \begin{bmatrix} \chi_k^x(i) \\ \chi_k^w(i) \\ \chi_k^\upsilon(i) \end{bmatrix} \qquad 2.41$$

The predicted mean for the state estimate is calculated using a weighted sum of points $\chi_{k+1}^x(i)$, given by:

$$\hat{\mathbf{x}}_{k+1}^- = \sum_{i=0}^{2L} W_i^{mean} \chi_{k+1}^x(i) \qquad 2.42$$

where the weight terms W_i^{mean} is given by:

$$W_0^{mean} = \frac{\lambda}{L+\lambda} \tag{2.43}$$

and

$$W_i^{mean} = W_i^{\text{cov}} = \frac{\lambda}{2(L+\lambda)}, \quad i = 1, 2, \ldots, 2L \tag{2.44}$$

Similarly, the predicted covariance term is given by:

$$P_{k+1}^{-} = \sum_{i=0}^{2L} W_i^{conv} \left[\chi_{k+1}^{x}(i) - \hat{\mathbf{x}}_{k+1}^{-} \right] \left[\chi_{k+1}^{x}(i) - \hat{\mathbf{x}}_{k+1}^{-} \right]^T \tag{2.45}$$

where the weight terms are given by 2.44, and the following equation.

$$W_0^{\text{cov}} = \frac{\lambda}{L+\lambda} + (1 - \alpha^2 + \beta) \tag{2.46}$$

The mean observation is given by

$$\hat{\mathbf{y}}_{k+1}^{-} = \sum_{i=0}^{2L} W_i^{mean} \gamma_{k+1}(i) \tag{2.47}$$

where

$$\boldsymbol{\gamma}_{k+1}(i) = \mathbf{h}\left(\chi_{k+1}^{x}(i), \mathbf{u}_{k+1}, \chi_{k+1}^{\upsilon}(i), k+1 \right) \tag{2.48}$$

The output covariance matrix is given by:

$$P_{k+1}^{yy} = \sum_{i=0}^{2L} W_i^{conv} \left[\gamma_{k+1}(i) - \hat{\mathbf{y}}_{k+1}^{-} \right] \left[\gamma_{k+1}(i) - \hat{\mathbf{y}}_{k+1}^{-} \right]^T \tag{2.49}$$

The innovations covariance is given by Equation 2.32. The cross correlation matrix is finally described as

$$P_{k+1}^{xy} = \sum_{i=0}^{2L} W_i^{conv} \left[\chi_{k+1}^{x}(i) - \hat{\mathbf{x}}_{k+1}^{-} \right] \left[\gamma_{k+1}(i) - \hat{\mathbf{y}}_{k+1}^{-} \right]^T \tag{2.50}$$

Finally, the Kalman gain and states are updated using the following equations.

$$K_k = P_k^{xy}(P_k^{\upsilon\upsilon})^{-1} \qquad 2.51$$

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + K_k\upsilon_k \qquad 2.52$$

$$P_k^+ = P_k^- - K_k P_k^{\upsilon\upsilon} K_k^T \qquad 2.53$$

$$\upsilon_k \equiv \tilde{\mathbf{y}}_k - \hat{\mathbf{y}}_k^- = \tilde{\mathbf{y}}_k - \mathbf{h}(\hat{\mathbf{x}}_k^-, \mathbf{u}_k k) \qquad 2.54$$

A summary of these equations are listed in Table 5 and will be referred to in subsequent sections that describe the implementation of these filters.

Table 5. Unscented Kalman Filter, from [9]

Model	$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{w}_k, \mathbf{u}_k, k)$ $\tilde{\mathbf{y}}_k = \mathbf{h}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{v}_k, k)$
Initialize	$\hat{\mathbf{q}}(k_0) = \hat{\mathbf{q}}_0, \quad \beta(k_0) = \beta_0$ $P(k_0) = P_0$
Gain	$K_k = P_k^{xy}(P_k^{\upsilon\upsilon})^{-1}$
Update	$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + K_k \upsilon_k$ $P_k^+ = P_k^- - K_k P_k^{\upsilon\upsilon} K_k^T$ $\upsilon_k \equiv \tilde{\mathbf{y}}_k - \hat{\mathbf{y}}_k^- = \tilde{\mathbf{y}}_k - \mathbf{h}(\hat{\mathbf{x}}_k^-, \mathbf{u}_k k)$
Propagation	$\hat{\mathbf{x}}_{k+1}^- = \sum_{i=0}^{2L} W_i^{mean} \chi_{k+1}^x(i)$ $P_{k+1}^- = \sum_{i=0}^{2L} W_i^{conv} \left[\chi_{k+1}^x(i) - \hat{\mathbf{x}}_{k+1}^-\right]\left[\chi_{k+1}^x(i) - \hat{\mathbf{x}}_{k+1}^-\right]^T$ $\hat{\mathbf{y}}_{k+1}^- = \sum_{i=0}^{2L} W_i^{mean} \gamma_{k+1}(i)$ $P_{k+1}^{yy} = \sum_{i=0}^{2L} W_i^{conv} \left[\gamma_{k+1}(i) - \hat{\mathbf{y}}_{k+1}^-\right]\left[\gamma_{k+1}(i) - \hat{\mathbf{y}}_{k+1}^-\right]^T$ $P_{k+1}^{\upsilon\upsilon} = P_{k+1}^{yy}$ $P_{k+1}^{xy} = \sum_{i=0}^{2L} W_i^{conv} \left[\chi_{k+1}^x(i) - \hat{\mathbf{x}}_{k+1}^-\right]\left[\gamma_{k+1}(i) - \hat{\mathbf{y}}_{k+1}^-\right]^T$

E. IMPLEMENTATION OF EKF AND UKF METHODS USING THE SIMPLE PENDULUM PROBLEM

Prior to implementing the EKF and UKF on the spacecraft model, an easier problem was solved. For this, the simple pendulum was used. Figure 7 shows a diagram of the simple pendulum problem.

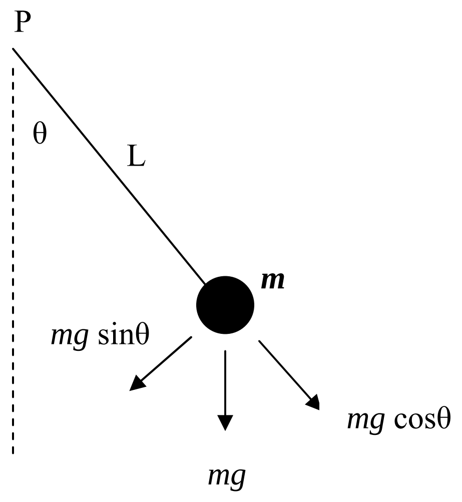


Figure 7. Simple Pendulum Problem

The dynamic equation is commonly known and listed below.

$$\ddot{\theta} = \frac{-mgl\sin(\theta)}{I_y + ml^2} \tag{2.55}$$

Translated into the state space model, this becomes:

$$x = \begin{Bmatrix} \theta \\ \dot{\theta} \\ I_y \end{Bmatrix} = \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} \tag{2.56}$$

$$\dot{x} = \begin{Bmatrix} \dot{\theta} \\ \ddot{\theta} \\ \dot{I}_y \end{Bmatrix} = \begin{Bmatrix} x_2 \\ x_3 \\ x_4 \end{Bmatrix} = \begin{Bmatrix} x_2 \\ \frac{-mgl\sin(x_1)}{x_3+ml^2} \\ 0 \end{Bmatrix} \tag{2.57}$$

We can now implement the state-space model into the simulation block diagram as our dynamics state function. The three states that were estimated were θ, $\dot{\theta}$, and I_y.

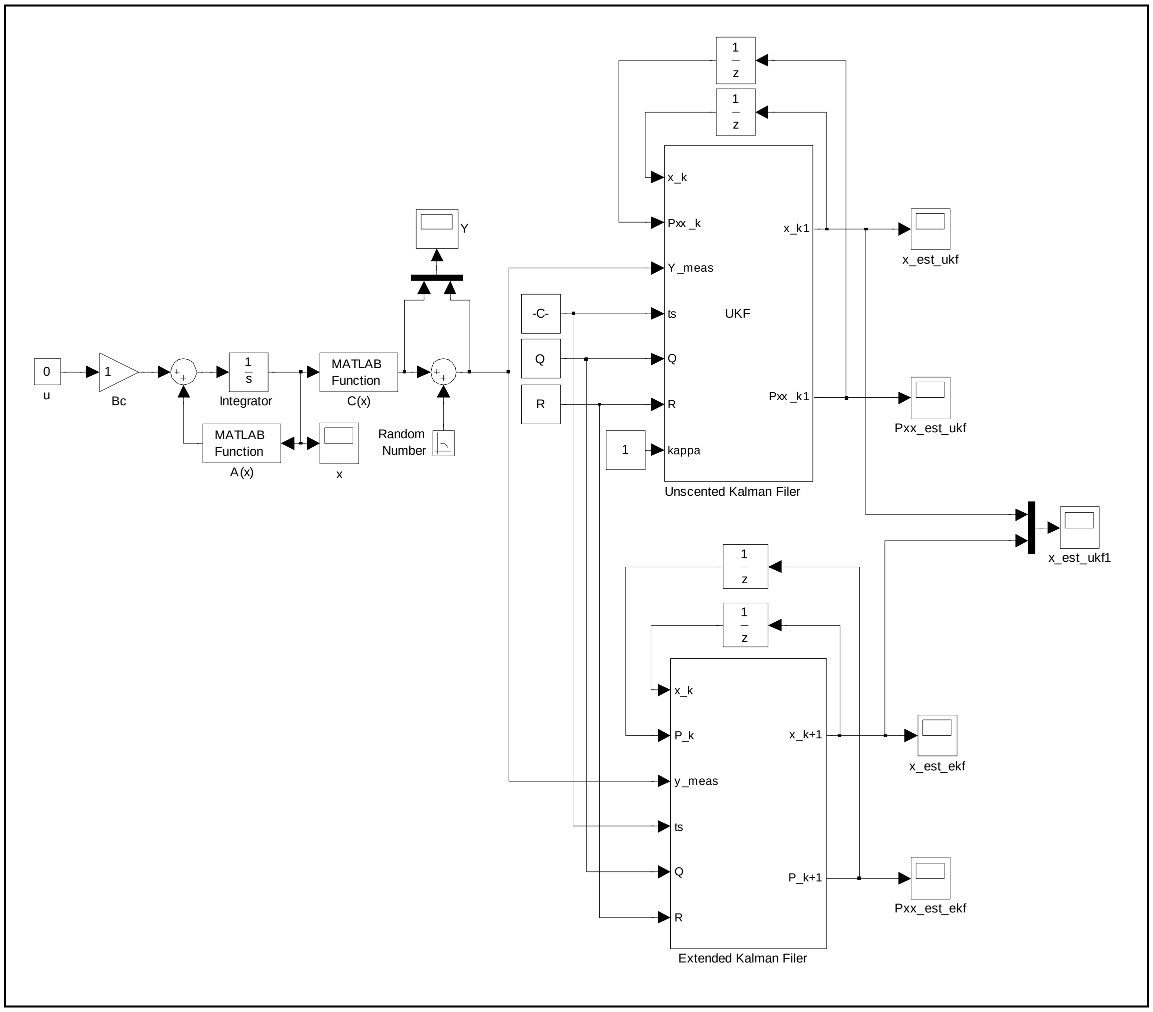


Figure 8. Simulink Block Diagram of Simple Pendulum Model

By solving Equation 2.57, we can then use its solution to determine our measurement equation.

$$y_{meas} = B_0 \sin(\theta + \alpha) \qquad 2.58$$

where y_{meas} represents the "measured" angle θ and α represents some initial angular quantity. The measured values are then perturbed by white Gaussian random numbers to simulating sensor noise and are subsequently fed into both the EKF and UKF. Appendix A – Simple Pendulum Simulation, shows the details of the simulation, including the simulation blocks, and associated Matlab code.

F. EKF AND UKF ESTIMATION RESULTS USING THE SIMPLE PENDULUM PROBLEM

The results of this estimation problem show how the EKF does not estimate accurately for nonlinear problems. For the first simulation the pendulum was set to θ = 30°, $\dot{\theta}$ = 0°/sec, and Iy = 5 kg m^2. Figure 9 shows the 3σ plot for the angle error between the estimated values and the true value. The 3σ plot is typically used to the confidence interval of a given set of data. While the term "3σ" actually refers to three times the variance of the data distribution, mathematically 3σ can be translated to mean that our data falls within approximately 99.73% of the symmetric confidence interval (CI). Conversely, this means that approximately 0.27% of the data falls outside the CI. The calculation for 3σ is shown below where the variance of diagonal values of the covariance matrix P^{xx} are used for n number of states.

$$P_{k+1}^{xx} = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,n} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n,1} & c_{n,2} & \cdots & c_{n,n} \end{bmatrix} \tag{2.59}$$

$$\sigma_i(t_k) = \pm\sqrt{c_{ii}(t_k)}, \quad i = 1,2,\ldots,n \tag{2.60}$$

Here we can see that the EKF cannot accurately estimate the state due to the nonlinearity of the system. As the pendulum swings and the angle increases, the nonlinearity of the dynamics increase and thus the filter becomes inaccurate. Conversely, we see in Figure 10 that the UKF accurately estimates the state well between the 3σ bounds.

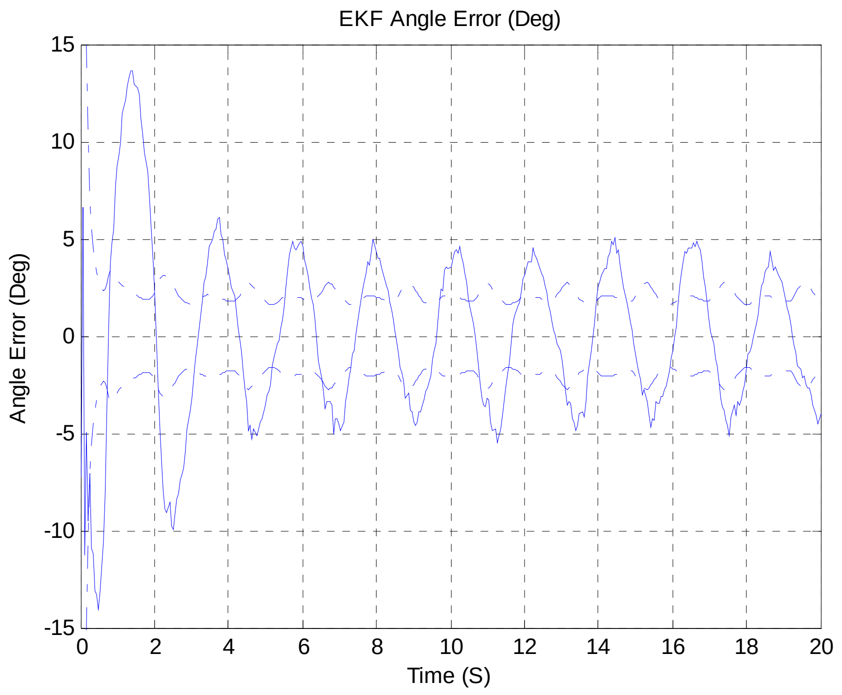


Figure 9. Angular Errors in EKF with 3σ Error Bounds Simulation 1 (large θ)

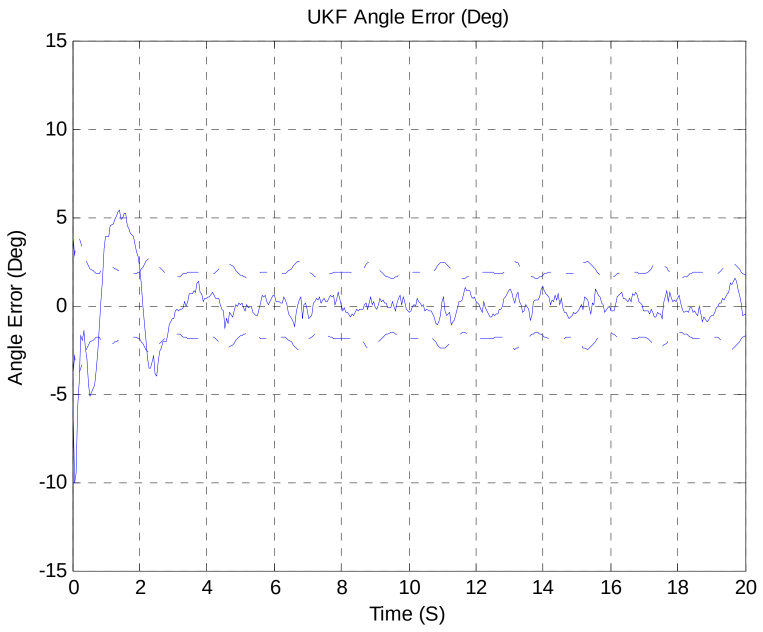


Figure 10. Angular Errors in UKF with 3σ Error Bounds Simulation 1 (large θ)

Figure 11 shows similar results for the estimation of the angular velocity ω.

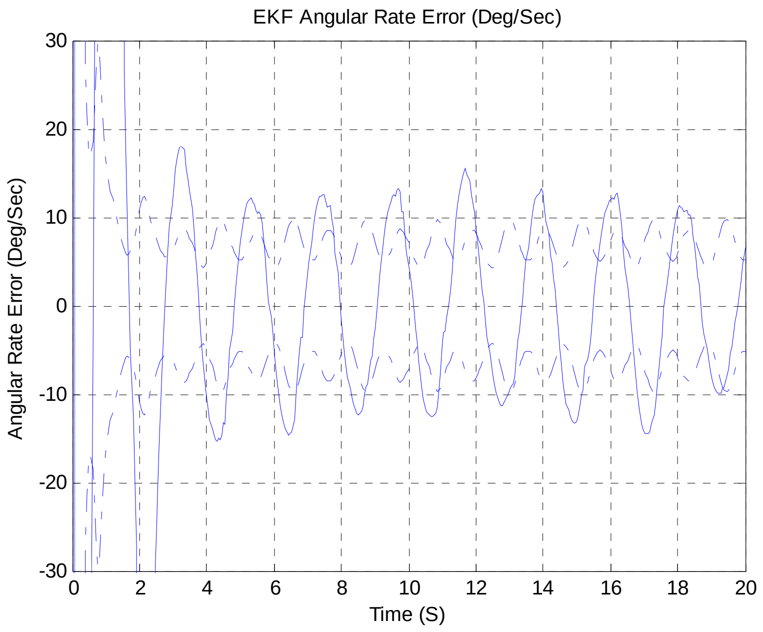


Figure 11. Angular Rate Errors in EKF with 3σ Error Bounds Simulation 1 (large θ)

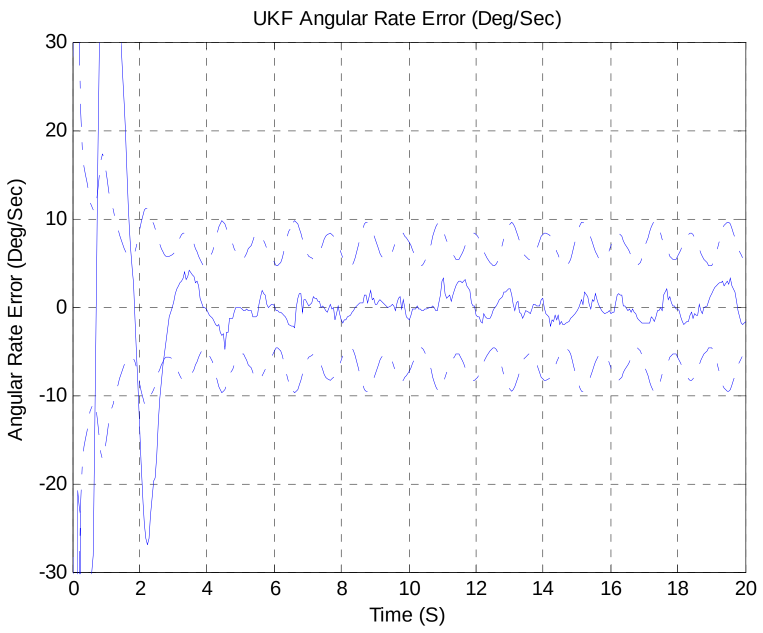


Figure 12. Angular Rate Errors in UKF with 3σ Error Bounds Simulation 1 (large θ)

Furthermore, we can see that the estimation for the moment of inertia, I_y, is accurate for both EKF and UKF. We can conclude that this is largely because I_y is a constant quantity.

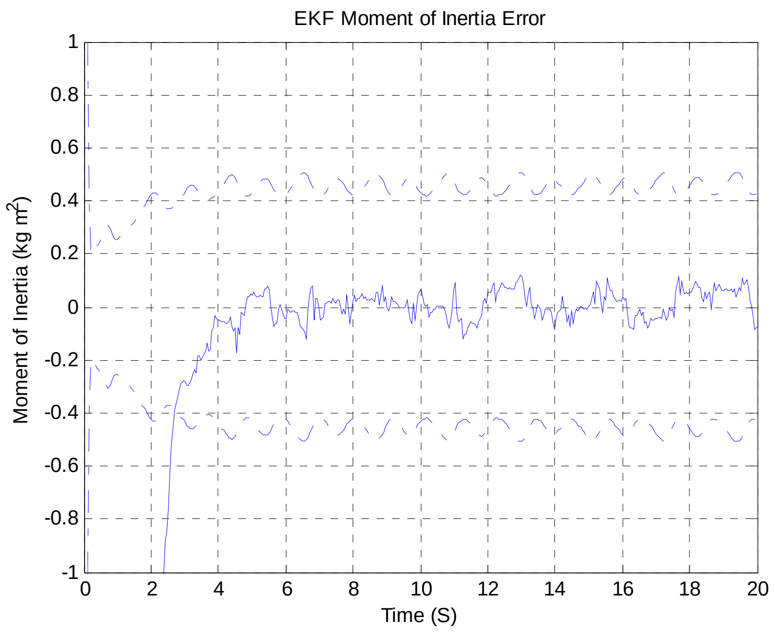


Figure 13. Moment of Inertia Errors in EKF with 3σ Error Bounds Simulation 1 (large θ)

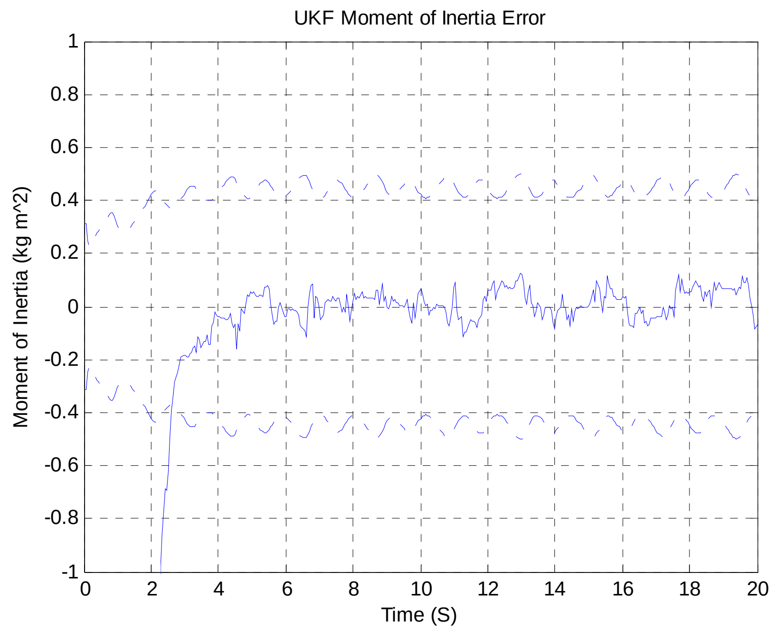


Figure 14. Moment of Inertia Errors in UKF with 3σ Error Bounds Simulation 1 (large θ)

These plots clearly show how the UKF provides a more accurate solution for even simple nonlinear problems. To further verify this, a second simulation was performed using smaller initial conditions. Using $\theta = 1^{\circ},\ \dot{\theta} = 0,\ and\ I_y = 5\ kg\ m^2$, we can see the both filters estimate well within the 3σ bounds. This can directly be associated with the small angle approximation where $\sin\theta \cong \theta$ for sufficiently small angles. These plots are shown below.

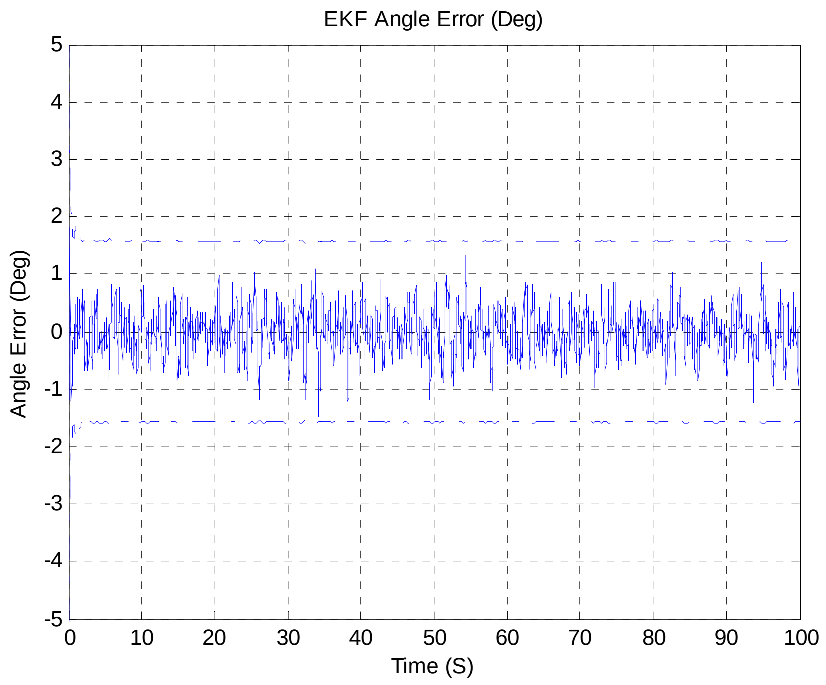


Figure 15. Angular Errors in EKF with 3σ Error Bounds for Simulation 2 (small θ)

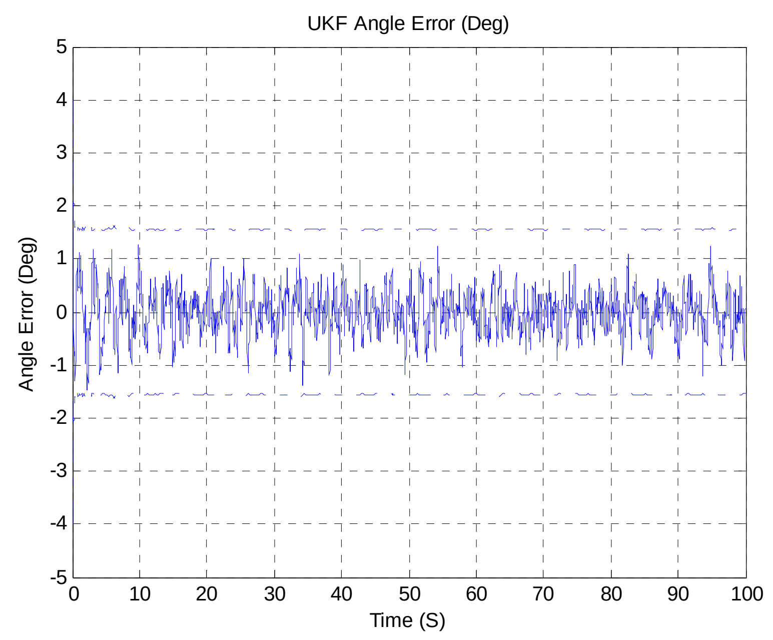


Figure 16. Angular Errors in UKF with 3σ Error Bounds for Simulation 2 (small θ)

Similarly, as shown below, we can see that the angular rates also fall within the bounds.

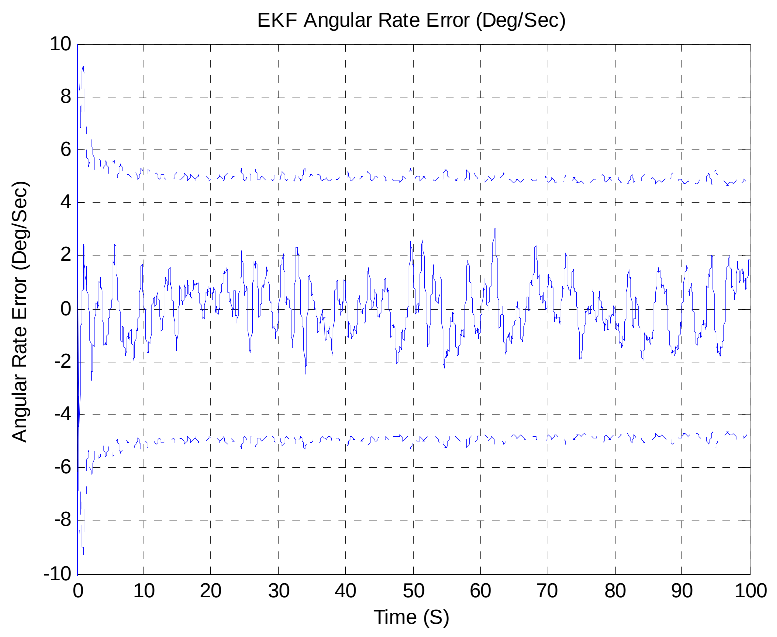


Figure 17. Angular Rate Errors in EKF with 3σ Error Bounds for Simulation 2 (small θ)

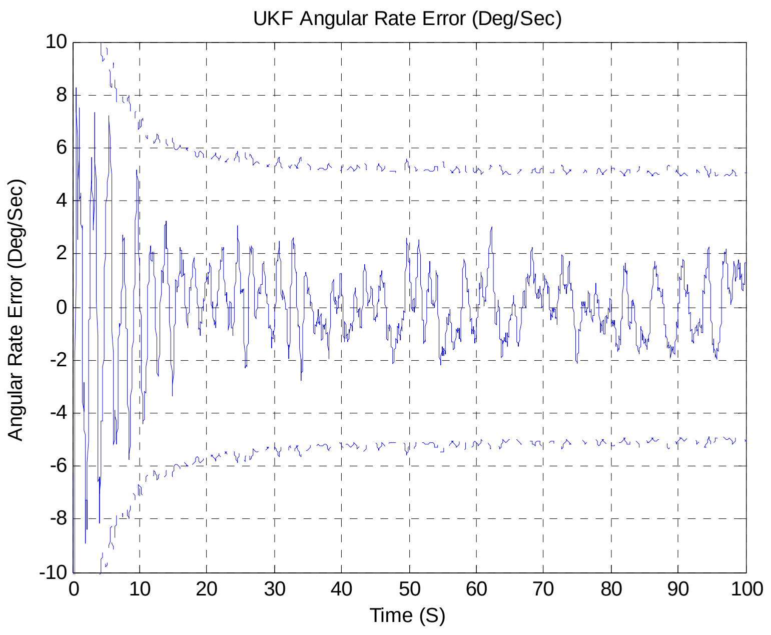


Figure 18. Angular Rate Errors in UKF with 3σ Error Bounds for Simulation 2 (small θ)

We can also see that the both moments of inertia also converge the proper values.

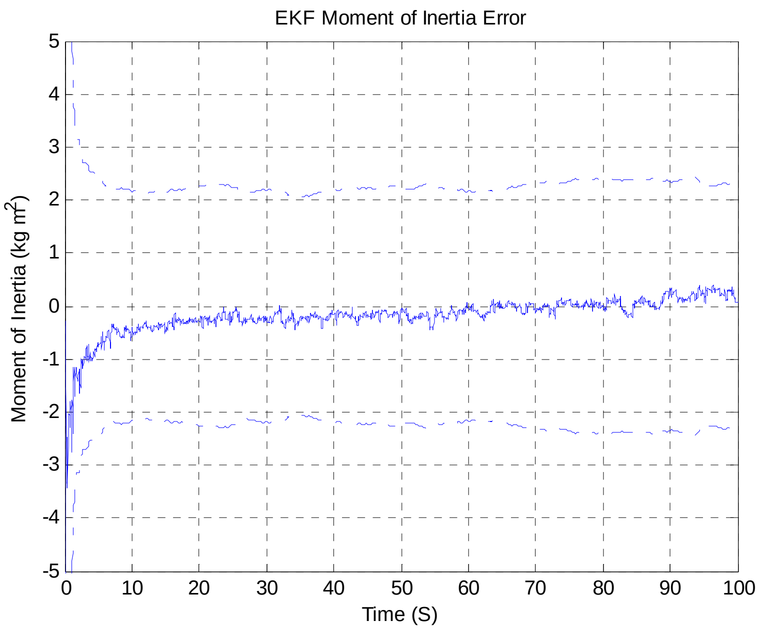


Figure 19. Moment of Inertia Errors in EKF with 3σ Error Bounds for Simulation 2 (small θ)

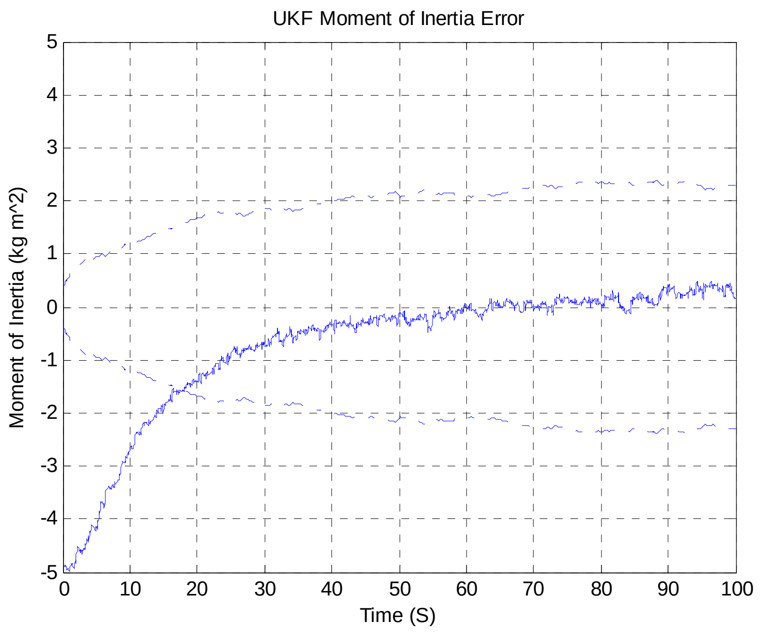


Figure 20. Moment of Inertia Errors in UKF with 3σ Error Bounds for Simulation 2 (small θ)

III. IMPLEMENTATION OF EKF AND UKF FOR SPACECRAFT ATTITUDE DETERMINATION

A. GENERALIZATIONS

While Chapter II discusses the fundamentals of both EKF and UKF, this chapter describes the implementation of both methods for attitude determination. In order to perform Kalman filtering for attitude estimation we must first examine the nature of quaternion estimation. The following discusses the analytical modeling setup, basic quaternion attitude kinematics, and finally, the implementation of both EKF and UKF filters for spacecraft attitude estimation.

B. ANALYTICAL MODELING AND SETUP FOR ATTITUDE DETERMINATION SIMULATIONS

1. Background

To implement the Kalman filters, a spacecraft simulation was created in MATLAB Simulink. Much of the initial foundation for this simulation was built previously, and is documented in [14] and [4]. For a better understanding of how the simulation works, the following sections will briefly discuss the several of the general Simulink Blocks. For our purposes, we will define the general simulation blocks as the following

- Orbit Propagator
- Environmental Effects
- Dynamics and Kinematics
- Disturbance Torques
- Sensors and Noise Modeling

Particularly, the blocks that will be discussed are Spacecraft Kinematics, Attitude Disturbance Torques, and Spacecraft Sensor/Noise Modeling. These blocks, and the entire simulation are shown in Appendix B – Spacecraft Attitude Determination Simulation.

2. Dynamics and Kinematics

Shown in Figure 21, the Dynamics block calculates the spacecraft angular body rates along each body axis by integrating applied forces, including control torques, based on Euler's equations [15].

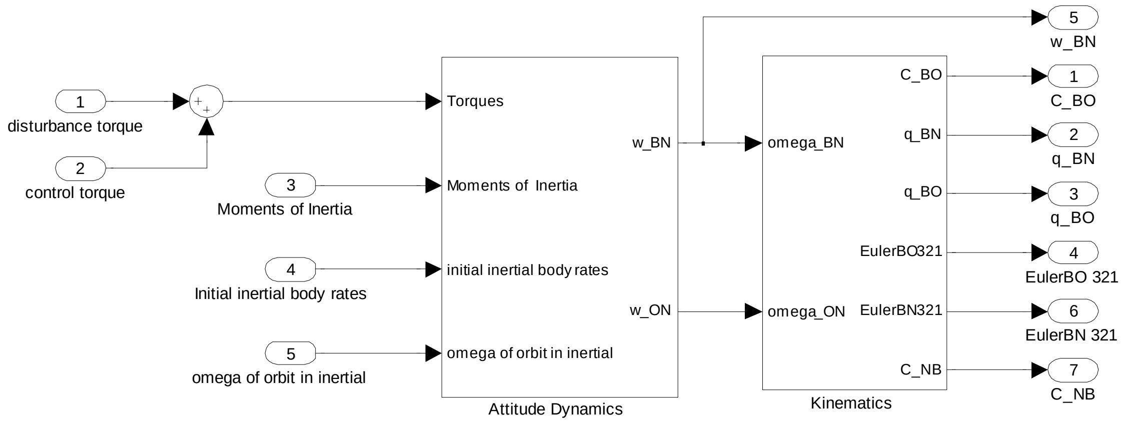


Figure 21. Attitude Dynamics and Kinematics Simulink Block

The Euler equations are listed below [15].

$$\begin{aligned}
\dot{\omega}_x &= \frac{T_x - (J_z - J_y)\omega_z\omega_y}{J_x} \\
\dot{\omega}_y &= \frac{T_y - (J_x - J_z)\omega_x\omega_z}{J_y} \\
\dot{\omega}_z &= \frac{T_z - (J_y - J_x)\omega_y\omega_x}{J_z}
\end{aligned} \qquad 2.61$$

These angular rates were then integrated in to the Kinematics block to determine the spacecraft orientation. For these simulations, the orientation is described in quaternions. The quaternion kinematic differential equation is:

$$\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & \omega_z & -\omega_y & \omega_x \\ -\omega_z & 0 & \omega_x & \omega_y \\ \omega_y & -\omega_x & 0 & \omega_z \\ -\omega_x & -\omega_y & -\omega_z & 0 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} \tag{2.62}$$

3. Spacecraft Attitude Disturbance Torques

Three major torque disturbances were taken into consideration for this simulation, gravity gradient, aerodynamic torque, and solar torque. For gravity gradient torque, the following equations were used.

$$T_{GG} = \frac{3\mu}{R^3} \begin{bmatrix} \left(J_z - J_y\right) c_2 c_3 \\ \left(J_x - J_y\right) c_1 c_3 \\ \left(J_y - J_x\right) c_1 c_2 \end{bmatrix} \tag{2.63}$$

where

$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = C_{BO} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{2.64}$$

The aerodynamic torque was calculated using the simple drag equation.

$$F_{aero} = \frac{1}{2} \rho V^2 C_D A \tag{2.65}$$

Here we should note that the velocity was determined from the orbit parameters, the coefficient of drag (C_D) was assumed to be 2, and effective area (A) was determined using spacecraft component areas and centers of pressure. Finally, solar torque was determined using a Simulink block diagram shown in Appendix B.

4. Spacecraft Sensor and Noise Modeling

All of sensor modeling done for this simulation was completed in Reference [4], where the author accurately modeled sensor sampling rates and noise sources based on manufacturer specifications. We can see this in the "Sensors Block" of the simulation. The most important information from the previous work is shown in Table 6, which shows the example of noise coefficients for the gyro. These numbers are implemented into the spacecraft simulation gyro random noise modeler.

Table 6. Summary of Gyro Noise Coefficients, from [4]

	$E\left[\eta_v^2\right]$ **(°/√sec)**	$E\left[\eta_u^2\right]$ **(°/√sec^3)**
Gyro Data 1	7.840e-04	1.440e-07
Gyro Data 2	7.840e-04	3.240e-08
Gyro Data 3	7.840e-04	3.240e-08

The equation for modeling the internal measurement unit (IMU) is listed below.

$$\tilde{\omega}(t) = \omega(t) + \beta(t) + \eta_v(t) \qquad 2.66$$

$$\dot{\beta}(t) = \eta_u(t) \qquad 2.67$$

where $\tilde{\omega}(t)$ is the continuous-time measured angular rate, and $\eta_v(t)$ and $\eta_u(t)$ are independent zero-mean Gaussian white-noise processes.

C. ATTITUDE KINEMATICS FOR QUATERNION ESTIMATION

This section describes the Kalman filter as it applies to attitude estimation. It is important to note that the equations found in this section apply to both the extended and the unscented Kalman filters.

The quaternion is defined in Equations 2.68, 2.69, and 2.70.

$$\mathbf{q} \equiv \begin{bmatrix} \varsigma^T & q_4 \end{bmatrix}^T \tag{2.68}$$

$$\varsigma \equiv \begin{bmatrix} q_1 & q_2 & q_3 \end{bmatrix}^T \equiv \hat{e}\sin(\tfrac{\theta}{2}) \tag{2.69}$$

$$q_4 = \cos(\tfrac{\theta}{2}) \tag{2.70}$$

where q is the quaternion, $\hat{e}$ is the Euler's axis, θ is the Euler's angle, and the quaternion follows the normalization of $\mathbf{q}^T\mathbf{q} = 1$. The attitude matrix can be related to the quaternion by the equation below [2].

$$A(\mathbf{q}) = \Xi^{\mathbf{T}}(\mathbf{q})\Psi(\mathbf{q}) \tag{2.71}$$

where $\Xi^T(\mathbf{q})$ and $\Psi^T(\mathbf{q})$ are defined by Equations 2.72 and 2.73.

$$\Xi^T(\mathbf{q}) \equiv \begin{bmatrix} q_4 I_{3\times3} + [\varsigma\times] \\ -\varsigma^T \end{bmatrix} \tag{2.72}$$

$$\Psi^T(\mathbf{q}) \equiv \begin{bmatrix} q_4 I_{3\times3} - [\varsigma\times] \\ -\varsigma^T \end{bmatrix} \tag{2.73}$$

Here $I_{3\times3}$ is a 3×3 identity matrix and $[\varsigma\times]$ is the cross product matrix described below.

$$[\varsigma\times] \equiv \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix} \qquad 2.74$$

Notably, the quaternion error cannot accurately found by subtraction, as the result would not satisfy the unit norm constraint, and a renormalization would be needed. The multiplicative error is defined as [16]:

$$\delta \mathbf{q} = \mathbf{q} \otimes \hat{\mathbf{q}}^{-1} \qquad 2.75$$

Here we use the symbol $\otimes$ to indicate the quaternion multiplication [2]. This relationship is described in Equation 2.76.

$$A(\mathbf{q}')A(\mathbf{q}) = A(\mathbf{q}' \otimes \mathbf{q}) \qquad 2.76$$

For implementation, the function XI was used in Matlab. This can be seen in Appendix A – Matlab Code and Simulink Diagrams. The time derivative of the quaternion error becomes

$$\delta \dot{\mathbf{q}} = \dot{\mathbf{q}} \otimes \hat{\mathbf{q}}^{-1} + \mathbf{q} \otimes \dot{\hat{\mathbf{q}}}^{-1} \qquad 2.77$$

As derived in [16], the estimated quaternion kinematics equation is given by

$$\dot{\mathbf{q}}(t) = \tfrac{1}{2}\Xi[\mathbf{q}(t)]\boldsymbol{\omega}(t) \qquad 2.78$$

Where $\boldsymbol{\omega}(t)$ is the 3×1 angular velocity vector. The local error-quaternion, $\delta q \equiv \begin{bmatrix} \delta\varsigma^T & \delta q_4 \end{bmatrix}^T$, can now be used to find the generalized Rodriguez parameter which will be useful later in the implementation of the UKF [4].

$$\delta\mathbf{p} \equiv f\,\frac{\delta\varsigma}{a+\delta q_4} \qquad 2.79$$

where a is a parameter from 0 to 1, and f is a scale factor. Suggested values for, *f*, is *2(a+1)* so that $\|\delta\mathbf{p}\|$ is equal to ϑ for small errors, where ϑ is the angle of rotation [4]. In the simulations presented in the thesis a was set to 1 to reproduce the results shown in [4]. While the propagation of the state and covariance can be accomplished by using numerical integration techniques, the measurement observations are typically sampled at higher rates than they are updated. This proves useful, as we can use a discretized version of the propagation equations. Using the power series, we can derive the new discretized propagation equation from 2.78 [9].

$$e^{\frac{\Omega(\hat{\omega})t}{2}} = \sum_{k=0}^{\infty}\left\{\frac{\left[\frac{1}{2}\Omega(\hat{\omega})t\right]^{2k}}{(2k)!} + \frac{\left[\frac{1}{2}\Omega(\hat{\omega})t\right]^{2k+1}}{(2k+1)!}\right\} \qquad 2.80$$

Using the identities described in Equations 2.81 and 2.82, we can substitute them into Equation 2.80.

$$\Omega^{2k}(\hat{\omega}) = (-1)^k \|\hat{\omega}\|^{2k} I_{4x4} \qquad 2.81$$

$$\Omega^{2k+1}(\hat{\omega}) = (-1)^k \|\hat{\omega}\|^{2k} \Omega(\hat{\omega}) \qquad 2.82$$

$$e^{\frac{\Omega(\hat{\omega})t}{2}} = I_{4x4}\sum_{k=0}^{\infty}\frac{(-1)^k\left[\frac{1}{2}\|\hat{\omega}\|t\right]^{2k}}{(2k)!} + \|\hat{\omega}\|^{-1}\Omega(\hat{\omega})\sum_{k=0}^{\infty}\frac{(-1)^k\left[\frac{1}{2}\|\hat{\omega}\|t\right]^{2k+1}}{(2k+1)!} \qquad 2.83$$

This equation then simplifies using trigonometric identities to the following

$$e^{\frac{\Omega(\hat{\omega})t}{2}} = I_{4x4}\cos\left(\frac{1}{2}\|\hat{\omega}\|t\right) + \Omega(\hat{\omega})\frac{\sin\left(\frac{1}{2}\|\hat{\omega}\|t\right)}{\|\hat{\omega}\|} \tag{2.84}$$

Finally, the quaternion propagation is found to be [9]:

$$\hat{\mathbf{q}}_{k+1}^{-} = \bar{\Omega}\left(\hat{\omega}_k^{+}\right)\hat{\mathbf{q}}_k^{+} \tag{2.85}$$

Where $\hat{\omega}_k^{+}$ and $\hat{\mathbf{q}}_k^{+}$ are the post-update estimates and $\mathbf{\Omega}\left(\hat{\omega}_k^{+}\right)$ are given by Equations 2.86 and 2.87.

$$\mathbf{\Omega}\left(\hat{\omega}_k^{+}\right) \equiv \begin{bmatrix} I_{3x3}\cos\left(\frac{1}{2}\|\hat{\omega}_k^{+}\|\Delta t\right) - \left[\hat{\Psi}_k^{+}\times\right] & \hat{\Psi}_k^{+} \\ -\hat{\Psi}_k^{+T} & \cos\left(\frac{1}{2}\|\hat{\omega}_k^{+}\|\Delta t\right) \end{bmatrix} \tag{2.86}$$

$$\hat{\Psi}_k^{+} \equiv \frac{\sin\left(\frac{1}{2}\|\hat{\boldsymbol{\omega}}_k^{+}\|\Delta t\right)\hat{\boldsymbol{\omega}}_k^{+}}{\|\hat{\boldsymbol{\omega}}_k^{+}\|} \tag{2.87}$$

For both the EKF and the UKF, we will use a rate gyro and a magnetometer. Given a post-update estimate for the bias $\hat{\beta}_k^{+}$, we will use the following equation to find the post-update angular velocity and propagated bias.

$$\begin{aligned} \hat{\omega}_k^{+} &= \tilde{\omega}_k - \hat{\beta}_k^{+} \\ \hat{\beta}_{k+1}^{-} &= \hat{\beta}_k^{+} \end{aligned} \tag{2.88}$$

We will use these equations in both the EKF and UKF to solve the attitude determination problem in the following sections.

D. CRASSIDIS AND MARKLEY'S UNSCENTED QUATERNION ESTIMATOR (USQUE)

In this section, the unscented Kalman filter described in Crassidis and Markley's paper on spacecraft attitude estimation, Reference [4], is implemented. This filter is called the UnScented QUaternion Estimator (USQUE). More specifically, the following describes how the USQUE is implemented in spacecraft attitude-determination simulations using MATLAB.

First, however, we must take step back and look at implementation of the UKF process as a sequential series of steps. Figure 22 shows the UKF graphically in the form of a flow chart. Similarly, we will refer to this flow chart throughout this section as it follows Crassidis and Markley's USQUE closely. The Matlab Code generated for this simulation also follows this flowchart and is listed in Appendix A.

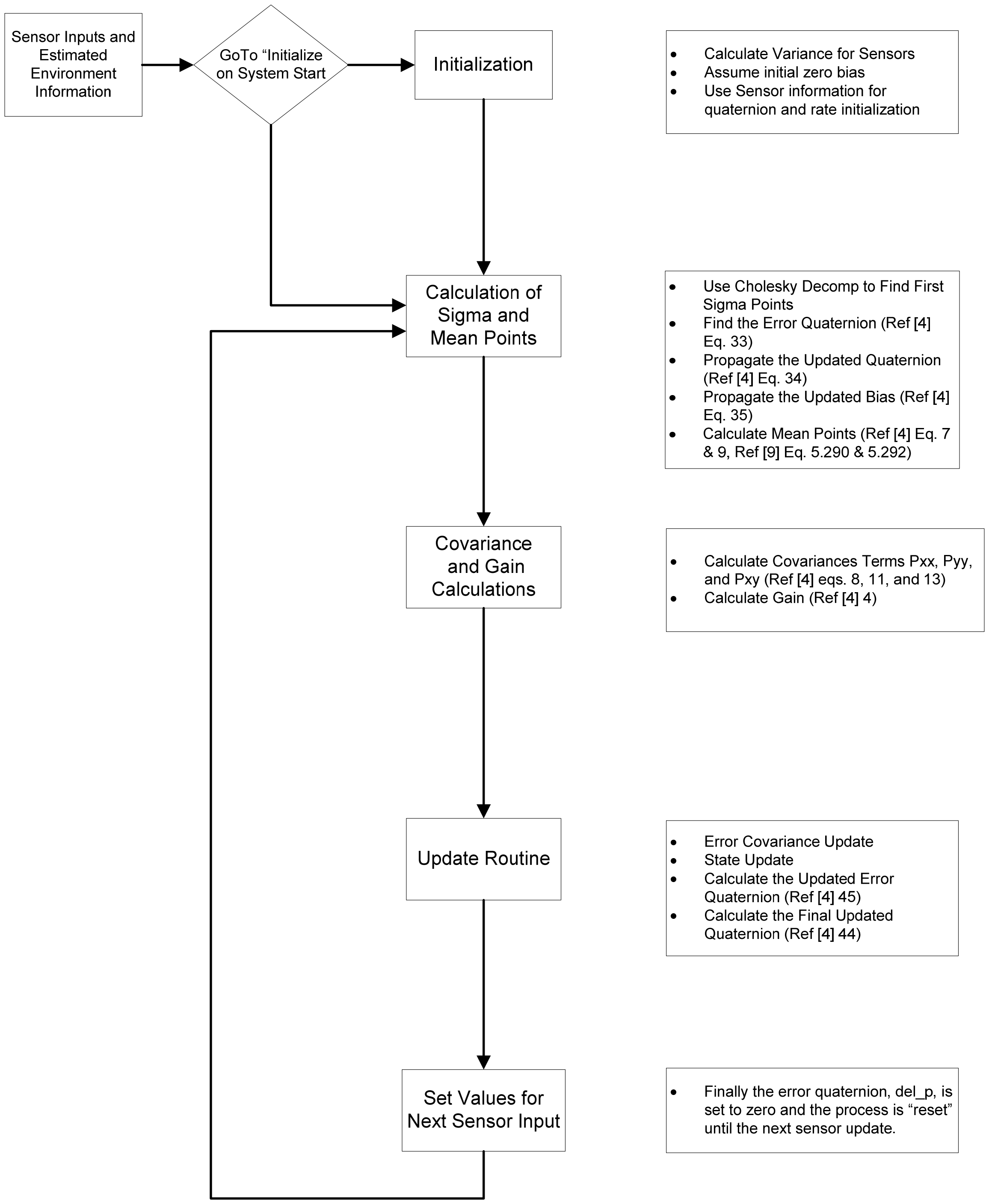

Sensor Inputs and Estimated Environment Information
GoTo "Initialize on System Start
Initialization
Calculate Variance for Sensors
Assume initial zero bias
Use Sensor information for quaternion and rate initialization
Calculation of Sigma and Mean Points
Use Cholesky Decomp to Find First Sigma Points
Find the Error Quaternion (Ref [4] Eq. 33)
Propagate the Updated Quaternion (Ref [4] Eq. 34)
Propagate the Updated Bias (Ref [4] Eq. 35)
Calculate Mean Points (Ref [4] Eq. 7 & 9, Ref [9] Eq. 5.290 & 5.292)
Covariance and Gain Calculations
Calculate Covariances Terms Pxx, Pyy, and Pxy (Ref [4] eqs. 8, 11, and 13)
Calculate Gain (Ref [4] 4)
Update Routine
Error Covariance Update
State Update
Calculate the Updated Error Quaternion (Ref [4] 45)
Calculate the Final Updated Quaternion (Ref [4] 44)
Set Values for Next Sensor Input
Finally the error quaternion, del_p, is set to zero and the process is "reset" until the next sensor update.

Figure 22. Unscented Quaternion Estimator Flow Chart

To begin, Figure 23 shows the UKF block of the attitude determination simulation shown in Appendix A.

Table 7 describes the inputs and outputs of this block.

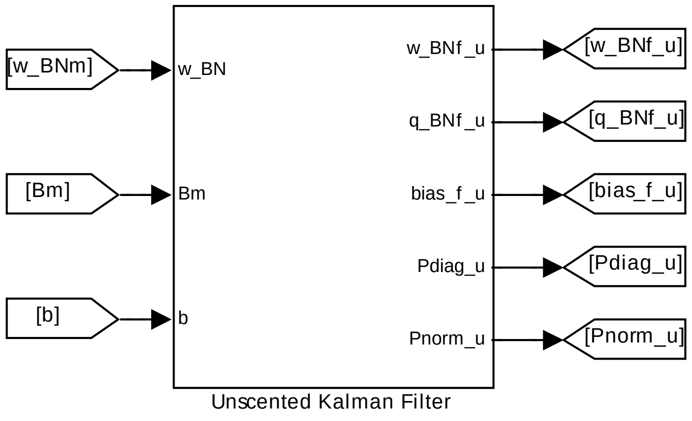


Figure 23. Unscented Kalman Filter Block - Level 1

Table 7. Description of Inputs and Outputs for UKF Block - Level 1

Input		**Output**	
Variable Name	**Description**	**Variable Name**	**Description**
w_BNm, ($\tilde{\omega}_k$)	Sensor measured angular rate (from gyro)	w_BNf_u, ($\hat{\omega}$)	Estimated angular rate
Bm, ($\tilde{\beta}_k$)	Sensor measured magnetic field (from magnetometer)	q_BNf_u, ($\hat{q}$)	Estimated quaternion
b, ($\hat{\beta}_k$)	Estimated magnetic field from environment model.	bias_f_u	Estimated magnetometer bias
		Pdiag_u	Diagonal terms of the covariance matrix
		Pnorm_u	Norm of the covariance matrix

As Figure 23 shows the top level of the UKF, Figure 24 shows the Level 2 block showing a few more inputs. These will be discussed further in the section. As a side note, the following sections are also well documented in the embedded Matlab code associated with this Simulink block. This code can be found in Appendix B.

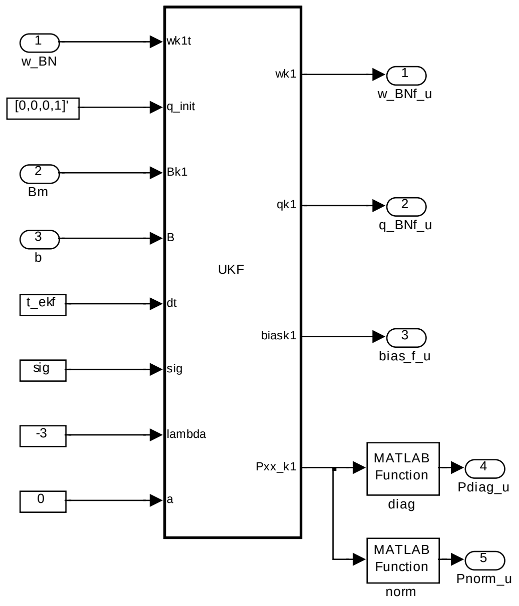


Figure 24. Unscented Kalman Filter Block - Level 2

1. Initialization

Referring to Figure 22, we can see that the USQUE process begins with the initialization portion of the estimation. Here we must choose the initial values for our states, which include the quaternion and the bias. For the initial simulations, the initial quaternion was set to [0,0,0,1] and the bias to [0,0,0]. For later simulations, as discussed in the results section, initial conditions were changed to highlight major differences between the UKF and EKF.

2. Calculation of Sigma Points

Calculations of Sigma Points begin with defining the following state vector:

$$\chi_k(0) = \hat{x}_k^+ \equiv \begin{bmatrix} \delta\hat{\mathbf{p}}_k^+ \\ \hat{\beta}_k^+ \end{bmatrix} \tag{2.89}$$

Here we can use Equation 2.79 for $\delta\hat{\mathbf{p}}_k^+$, which is the 4 x 1 error quaternion, and the 3 state bias term, $\hat{\beta}_k^+$. These values will be propagated and used to update the final nominal state. This resulting covariance matrix is a 6 x 6. It is important to note here that for propagating these values forward we can now use Equations 2.42 through 2.50. However, before we use these equations, we must partition the sigma points $\chi_k(i)$ so that we can work only with the quaternion portion.

$$\boldsymbol{\chi}_k(i) \equiv \begin{bmatrix} \boldsymbol{\chi}_k^{\delta p}(i) \\ \boldsymbol{\chi}_k^{\beta}(i) \end{bmatrix} \quad i = 0,1,\ldots,12 \tag{2.90}$$

where $\boldsymbol{\chi}_k^{\delta p}$ is the attitude error part, and $\boldsymbol{\chi}_k^{\beta}(i)$ is the gyro bias part. Now that we have parsed out these terms, we must determine the new quaternion generated by multiplying the error quaternion by its current estimate.

$$\hat{\mathbf{q}}_k^+(0) = \hat{\mathbf{q}}_k^+ \tag{2.91}$$

$$\hat{\mathbf{q}}_k^+(i) = \delta\hat{\mathbf{q}}_k^+(i) \otimes \hat{\mathbf{q}}_k^+ \quad i = 1,2,\ldots,12 \tag{2.92}$$

where $\hat{\mathbf{q}}_k^+$ is the current quaternion estimate, and $\delta\hat{\mathbf{q}}_k^+$ is the error quaternion. The error quaternion is broken up into the 3 state quaternion vector $\delta\boldsymbol{\varsigma}_k^+$, and the forth quaternion, $\delta q_{4_k}^+$, shown in Equations 2.93, 2.94, and 2.95.

$$\delta\hat{\mathbf{q}}_k^+(i) \equiv \left[\delta\boldsymbol{\varsigma}_k^{+T}(i) \quad \delta q_{4_k}^+(i) \right]^T, \quad i = 1,2,...,12 \tag{2.93}$$

$$\delta q_{4_k}^{+}(i) = \frac{-a\left\|\boldsymbol{\chi}_k^{\delta\mathbf{p}}(i)\right\|^2 + f\sqrt{f^2+(1-a^2)\left\|\boldsymbol{\chi}_k^{\delta\mathbf{p}}(i)\right\|^2}}{f^2+\left\|\boldsymbol{\chi}_k^{\delta\mathbf{p}}(i)\right\|^2}, \quad i = 1,2,\ldots,12 \qquad 2.94$$

$$\delta\boldsymbol{\varsigma}_k^{+}(i) = f^{-1}\left[a+\delta q_{4_k}^{+}(i)\right]\boldsymbol{\chi}_k^{\delta\mathbf{p}}(i), \quad i = 1,2,\ldots,12 \qquad 2.95$$

We chose $f = 2(a+1)$, where a values were selected using Table 1 from [4]. Next, these updated quaternions are propagated forward using Equation 2.85 for each *i*, or step, shown below.

$$\hat{\mathbf{q}}_{k+1}^{-}(i) = \bar{\Omega}\left(\hat{\omega}_k^{+}(i)\right)\hat{\mathbf{q}}_k^{+}(i) \quad i = 0,1,\ldots,12 \qquad 2.96$$

where again the angular velocities are given by Equation 2.97 similar to Equation 2.88 in the previous section. Here, we can see that for $\hat{\omega}_k^{+}(0) = \tilde{\omega}_k - \boldsymbol{\chi}_k^{\beta}(0)$, $\boldsymbol{\chi}_k^{\beta}(0) = \hat{\beta}_k^{+}$.

$$\hat{\omega}_k^{+}(i) = \tilde{\omega}_k - \boldsymbol{\chi}_k^{\beta}(i), \quad i = 0,1,\ldots,12 \qquad 2.97$$

The propagated error quaternions are now calculated using Equation 2.98.

$$\delta\mathbf{q}_{k+1}^{-}(i) = \hat{\mathbf{q}}_{k+1}^{-}(i)\otimes\left[\hat{\mathbf{q}}_{k+1}^{-}(0)\right]^{-1}, \quad i = 0,1,\ldots,12 \qquad 2.98$$

it is interesting to note that where $\delta\hat{\mathbf{q}}_{k+1}^{-}(0)$ here should be the identity quaternion [0, 0, 0, 1]. Finally, the propagated sigma points can be calculated using the following equations.

$$\chi_{k+1}^{\delta\mathbf{p}}(0) = 0 \qquad 2.99$$

$$\boldsymbol{\chi}_{k+1}^{\delta\mathbf{p}}(i) = f\,\frac{\delta\boldsymbol{\varsigma}_{k+1}^{-}(i)}{a+\delta q_{4_{k+1}}^{-}(i)}, \quad i = 1,2,\ldots,12 \qquad 2.100$$

where $\delta\boldsymbol{\varsigma}_{k+1}^{-}(i)$ and $\delta q_{4_{k+1}}^{-}(i)$ are found from the following equation.

$$\delta\mathbf{q}_{k+1}^{-}(i) = \left[\delta\boldsymbol{\varsigma}_{k+1}^{-T}(i) \quad \delta q_{4_{k+1}}^{-T}(i)\right] \tag{2.101}$$

We also know that from Equation 2.88, we can expect the following.

$$\boldsymbol{\chi}_{k+1}^{\beta}(i) = \boldsymbol{\chi}_{k}^{\beta}(i), \quad i = 1,2,...,12 \tag{2.102}$$

3. Covariance and Gain Calculations

The next step in the UKF process is to calculate the covariances and gains which is the most notable difference between the EKF and UKF. Now that we have calculated our sigma points, we can determine these values. As previously mentioned in the derivation of the UKF, we can determine the predicted covariance matrix P_{k+1}^{-}, shown as *Pxx* in the Matlab code, the output covariance P_{k+1}^{yy}, and the cross correlation covariance P_{k+1}^{xy}. These equations are found as 2.39, 2.43, and 2.45, respectively. These equations are utilized in the "Covariance and Gain Calculations" section of the embedded Matlab code for the UKF. For initial conditions, P_{xx} is the set to $\bar{Q}_k$, where the variations for the sensors are used. The following equation is used for $\bar{Q}_k$.

$$\bar{Q}_k = \frac{\Delta t}{2}\begin{bmatrix} (\sigma_{\upsilon}^2 - \frac{1}{6}\sigma_u^2\Delta t^2)I_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & \sigma_u^2 I_{3\times3}\end{bmatrix} \tag{2.103}$$

The mean observation is also needed to calculate the covariance terms.

$$\gamma_{k+1}(i) = \begin{bmatrix} A\left[\hat{\mathbf{q}}^{-}(i)\right]r_1 \\ A\left[\hat{\mathbf{q}}^{-}(i)\right]r_2 \\ \vdots \\ A\left[\hat{\mathbf{q}}^{-}(i)\right]r_N \end{bmatrix}_{k+1}, \quad i = 0,1,...,12 \tag{2.104}$$

With these covariance matrices calculated, we can now determine the Kalman gain K from the equation in Table 5. Unscented Kalman Filter, from [9]. This equation is also shown below.

$$K_k = P_k^{xy}(P_k^{\upsilon\upsilon})^{-1} \tag{2.105}$$

4. Update Routine for States and Error Covariance

After the gains are calculated, states and error covariances must be updated. First, the error covariance is updated using the following equation.

$$P_k^+ = P_k^- - K_k P_k^{\upsilon\upsilon} K_k^T \tag{2.106}$$

The state update is found using

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + K_k \upsilon_k \tag{2.107}$$

Finally we can update the quaternions using the following set of equations where $\hat{\mathbf{x}}_{k+1}^+ = \left[\delta\hat{\mathbf{p}}_{k+1}^+, \quad \hat{\beta}_{k+1}^+\right]$.

$$\delta\mathbf{q}_{k+1}^+ = \hat{\mathbf{q}}_{k+1}^+ \otimes \hat{\mathbf{q}}_{k+1}^-(0) \tag{2.108}$$

$$\delta\hat{\mathbf{q}}_{k+1}^+ \equiv \left[\delta\boldsymbol{\varsigma}_{k+1}^{+T} \quad \delta q_{4_{k+1}}^+\right]^T \tag{2.109}$$

$$\delta q_{4_{k+1}}^+ = \frac{-a\left\|\boldsymbol{\delta}\hat{\mathbf{p}}_{k+1}^+\right\|^2 + f\sqrt{f^2 + (1-a^2)\left\|\boldsymbol{\delta}\mathbf{p}_{k+1}^+\right\|^2}}{f^2 + \left\|\boldsymbol{\delta}\mathbf{p}_{k+1}^+\right\|^2} \tag{2.110}$$

$$\delta\boldsymbol{\varsigma}_{k+1}^+ = f^{-1}\left[a + \delta q_{4_{k+1}}^+\right]\delta\hat{\mathbf{p}}_{k+1}^+ \tag{2.111}$$

These equations are very similar to those used earlier to find the initial error quaternion. The final step here is to update the bias using Equation 2.88. For further clarification, the Matlab code references the equations used with respect to [4]. This unscented Kalman filter was built to be compared with the extended Kalman filter. The EKF and UKF models for attitude are based on the model presented in [9] and [4] where the state vector is represented as the error in the quaternion and generalized Rodriquez parameter respectively.

E. IMPLEMENTATION OF THE EXTENDED KALMAN FILTER

The EKF implemented in this simulation uses many of the equations used in previous sections. Similar to the previous section, Figure 25 shows a flow chart of the EKF. By comparison, we can see very clearly that the major difference in the EKF is the calculation of the sensitivity matrix, which is the inherent linearization processes associated with this filter. Much of the information on the derivation of the EKF is discussed in [4] and Table 8 shows a summary of the EKF equations. A complete listing of both the Matlab Simulation block diagram and embedded Matlab code are shown in Appendix B.

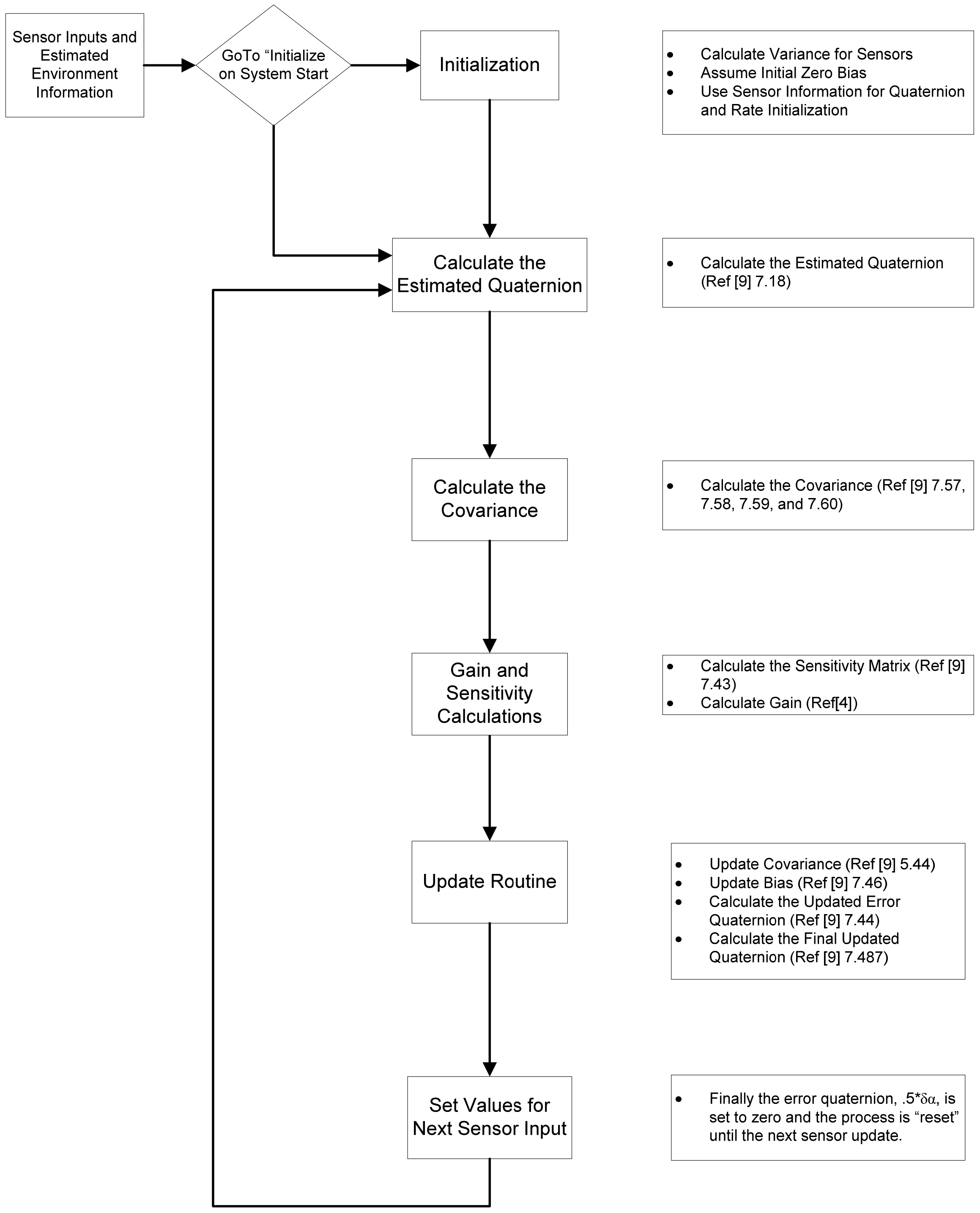

Sensor Inputs and Estimated Environment Information
GoTo "Initialize on System Start
Initialization
• Calculate Variance for Sensors
• Assume Initial Zero Bias
• Use Sensor Information for Quaternion and Rate Initialization
Calculate the Estimated Quaternion
• Calculate the Estimated Quaternion (Ref [9] 7.18)
Calculate the Covariance
• Calculate the Covariance (Ref [9] 7.57, 7.58, 7.59, and 7.60)
Gain and Sensitivity Calculations
• Calculate the Sensitivity Matrix (Ref [9] 7.43)
• Calculate Gain (Ref[4])
Update Routine
• Update Covariance (Ref [9] 5.44)
• Update Bias (Ref [9] 7.46)
• Calculate the Updated Error Quaternion (Ref [9] 7.44)
• Calculate the Final Updated Quaternion (Ref [9] 7.487)
Set Values for Next Sensor Input
• Finally the error quaternion, .5*δα, is set to zero and the process is "reset" until the next sensor update.

Figure 25. Extended Kalman Filter Flow Chart

Table 8. Summary of EKF Equations, from [9]

Initialize	$\hat{\mathbf{q}}(k_0)=\hat{\mathbf{q}}_0, \quad \beta(k_0)=\beta_0$ $P(k_0)=P_0$
Propagate	$\hat{\omega}_k^+ = \tilde{\omega}_k - \hat{\beta}_k^+$ $\hat{\beta}_k^- = \hat{\beta}_k^+$ $\hat{\mathbf{q}}_{k+1}^- = \bar{\Omega}\left(\hat{\omega}_k^+\right)\hat{\mathbf{q}}_k^+$ $P_{k+1}^- = \Phi_k P_k^+ \Phi_k^T + G_k Q_k G_k^T$
Gain	$K_k = P_k^- H_k^T\left(\hat{\mathbf{x}}_k^-\right)\left[H_k\left(\hat{\mathbf{x}}_k^-\right)P_k^- H_k^T\left(\hat{\mathbf{x}}_k^-\right)+R_k\right]^{-1}$ ${}^B H_k = \left[\left[-B_m\times\right] \quad 0_{3x3}\right]$
Update	$\Delta\hat{\tilde{\mathbf{x}}}_k^+ \equiv \left[\Delta\hat{\alpha}_k^{+T} \quad \Delta\hat{\beta}_k^{+T}\right]$ $\Delta\hat{\tilde{\mathbf{x}}}_k^+ = K_k\left[\mathbf{y}_k - \mathbf{h}_k\left(\hat{\mathbf{x}}_k^-\right)\right]$ $\hat{\beta}_k^+ = \hat{\beta}_k^- + \Delta\hat{\beta}_k^+$ $\hat{\mathbf{q}}_k^+ = \hat{\mathbf{q}}_k^- + \frac{\Xi\left(\hat{\mathbf{q}}_k^-\right)}{2}\delta\hat{\alpha}_k^+$ $P_k^+ = \left[I - K_k H_k\left(\hat{\mathbf{x}}_k^-\right)\right]P_k^-$

THIS PAGE INTENTIONALLY LEFT BLANK

IV. COMPARISON OF SIMULATION RESULTS USING EKF AND UKF FILTERING METHODS

A. SIMULATION CONDITIONS

In this section, several performance comparisons between the USQUE and EKF are made through simulations using the previously discussed spacecraft model and the designed EKF and UKF filters. Using a 500-km circular orbit the simulation time was set at 4,000 seconds. The attitude determination hardware in these simulations consisted of a gyroscopic rate sensor and a three-axis magnetometer (TAM). The magnetic field reference model uses a magnetic dipole approximation as previously discussed. Furthermore, these sensors were characterized in previous work [4]. In the first simulation, the initial attitude error was set only to 30°, while the attitude rate error was set to 0°/sec in all axes. A second simulation was run using an initial attitude error of 30° and an attitude rate error of 30°/sec in all axes.

B. SIMULATION 1 RESULTS

The following shows the results of Simulation 1. Figure 26 shows the attitude error of the quaternion for the EKF estimator with 3σ bounds. We can see that the EKF takes approximately 4,000 seconds before the error is bounded. Conversely, we can see that the attitude error of the UKF is bounded in approximately 2,500 seconds. This is shown in Figure 27 where the generalized Rodriguez parameters are shown. It is important to note that we use the generalized Rodriquez parameters instead of the quaternion for the UKF because the 3σ bounds are calculated from the square root of the diagonals of the covariance matrix P. For the UKF the covariance matrix is built from $\delta\mathbf{p}$ which is shown in Equation 2.79, as $\delta\mathbf{p}$ is the error in the vector of the generalized Rodriquez parameter. For comparison purposes, Figure 30 will show both normalized quaternion errors without the 3σ bounds.

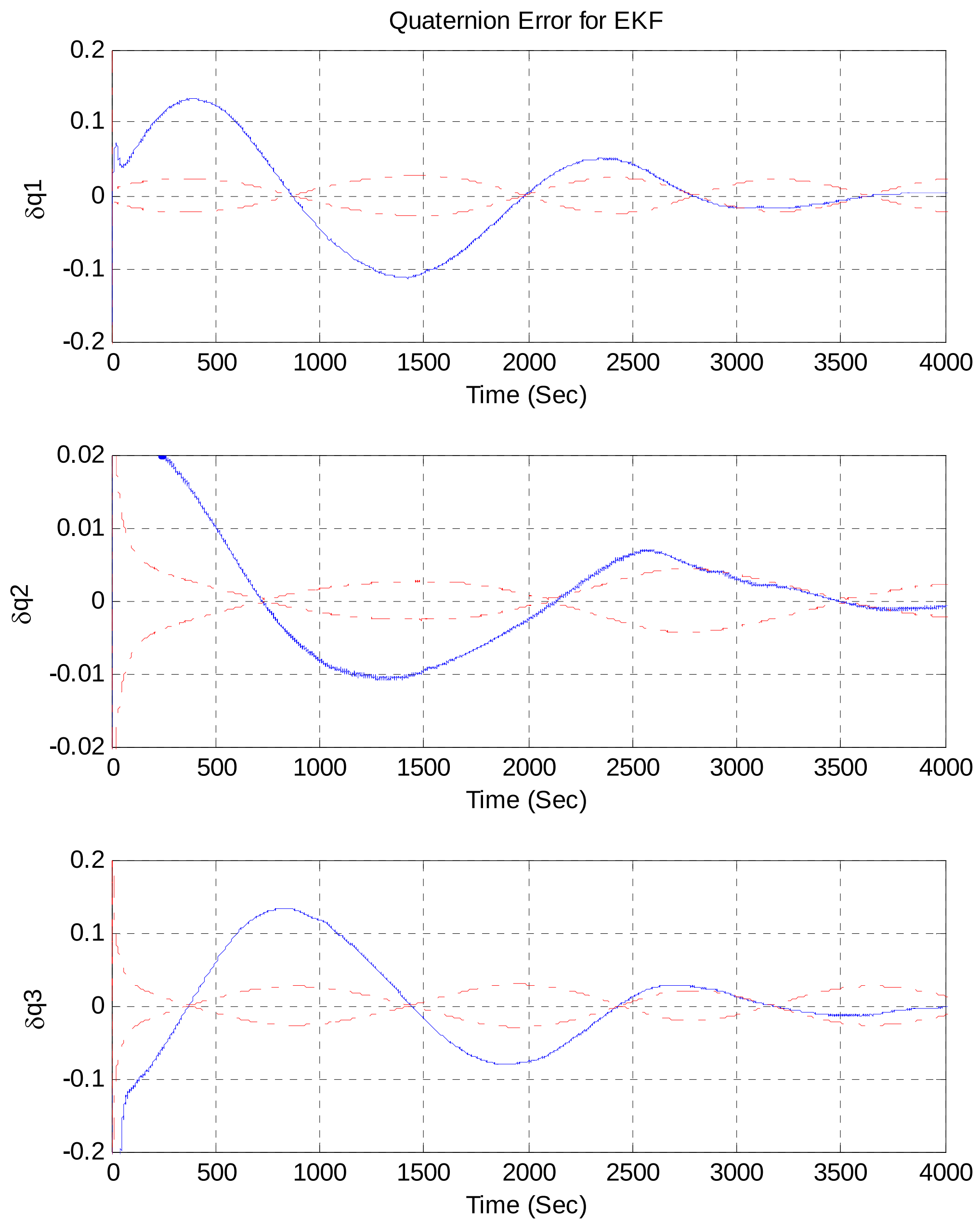


Figure 26. Simulation 1 Quaternion Attitude Error with 3σ Bounds for EKF

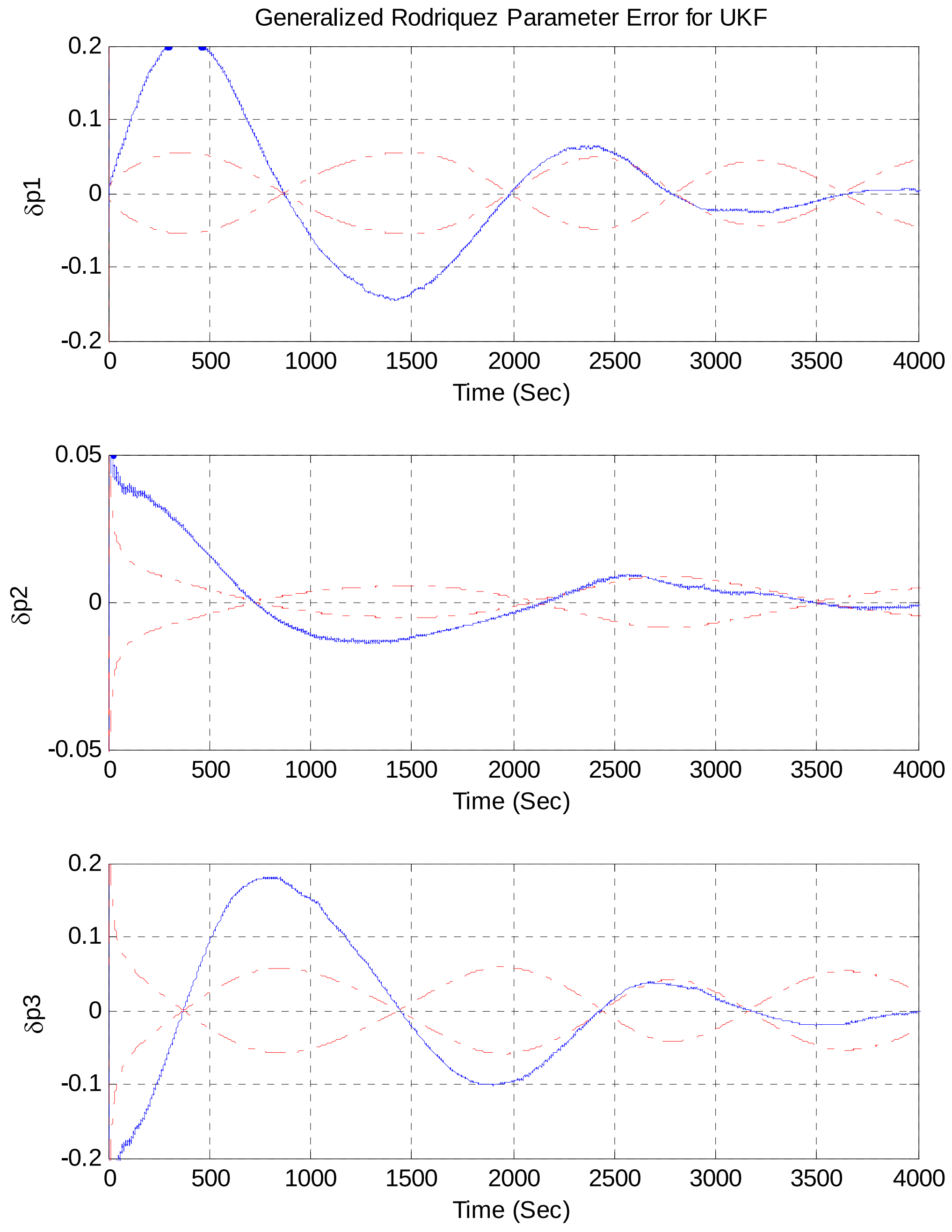


Figure 27. Simulation 1 Generalized Rodriguez Parameter Attitude Error with for 3σ Bounds for UKF

Similarly, we see that both EKF and UKF bias errors converge in a similar way. While Figure 28 shows the bias for the EKF converging within the 3σ bounds at approximately 2,700 seconds, Figure 29 shows convergence at a little after 1,000 seconds. We can also see that the initial estimates of the EKF are more inaccurate than the UKF.

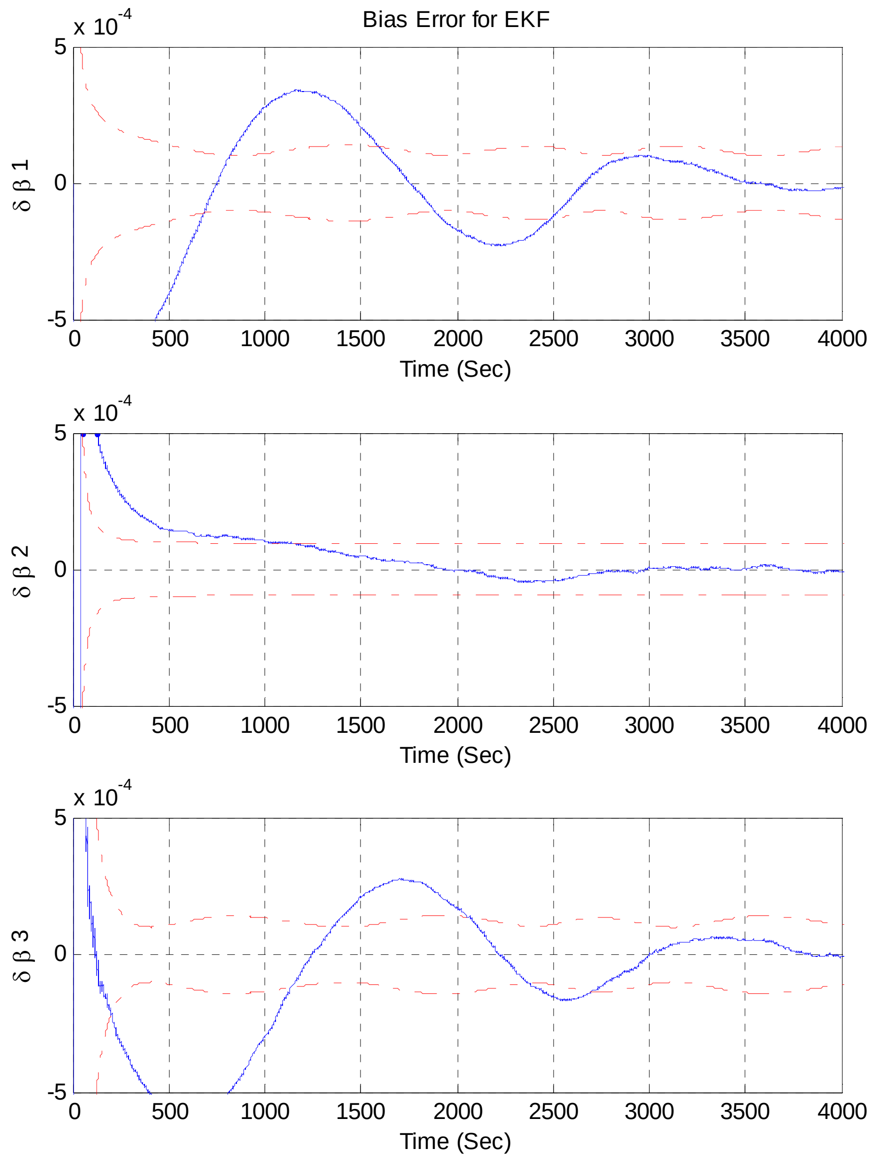


Figure 28. Simulation 1 EKF Bias Errors with 3σ Bounds

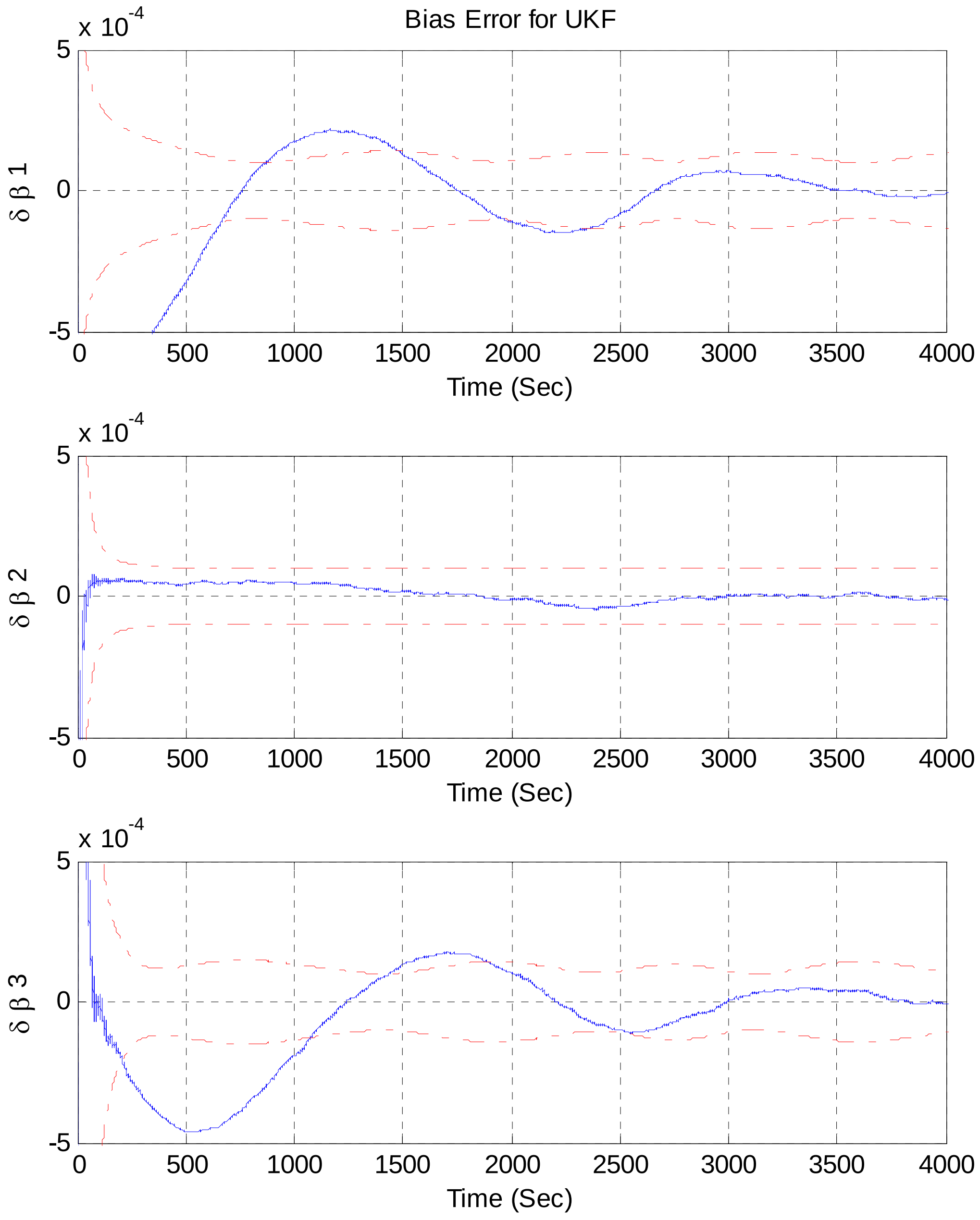


Figure 29. Simulation 1 UKF Bias Errors with 3σ Bounds

Finally, we can see that the normalized EKF and UKF attitude errors converge as originally predicted and demonstrated in the simple pendulum problem. Figure 30 clearly shows the better performance of the UKF. Again, as a nonlinear estimator, the UKF consistently shows better performance on all figures. This is again shown in comparison plot of the normalized bias errors displayed in Figure 31 where we can see that, although both estimators are trending appropriately, that the UKF performs significantly better.

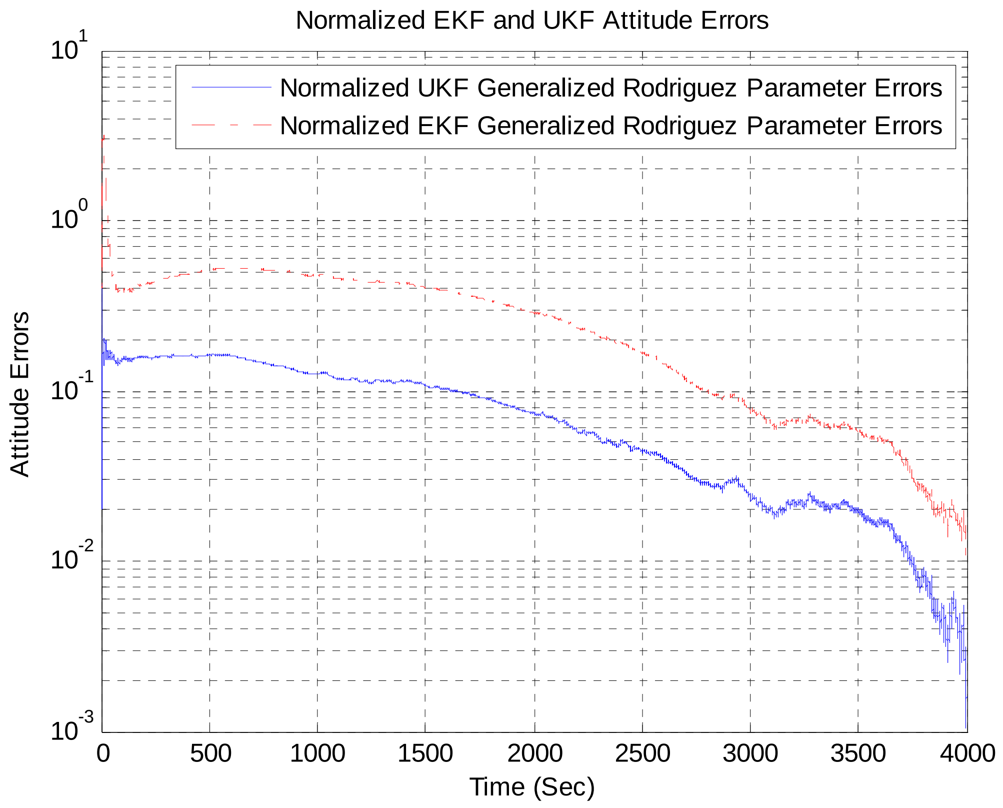


Figure 30. Comparison of EKF and UKF Normalized Attitude Errors for Simulation 1

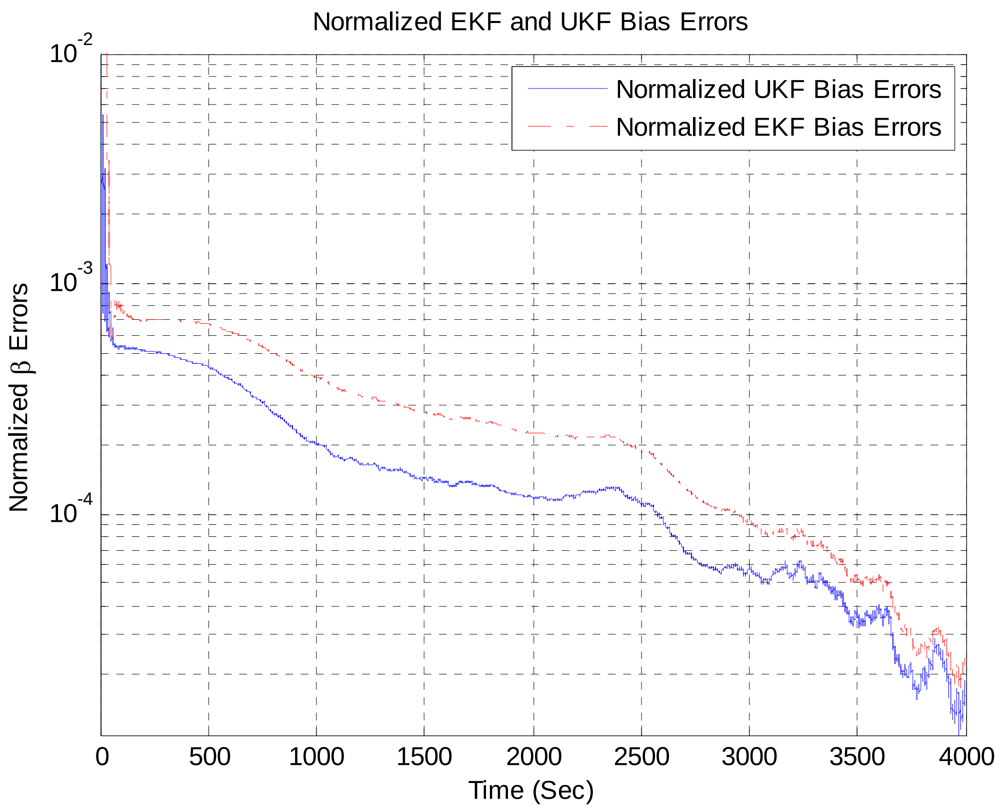


Figure 31. Comparison of EKF and UKF Normalized Bias Errors for Simulation 1

C. SIMULATION 2 RESULTS

The second simulation shows very similar results. Although we can see similar trends in both the EKF and UKF error estimates, we can see that the UKF consistently performs better in every plot. Again, Figure 32 shows the EKF attitude quaternion error, which settles within the 3σ bounds at approximately 3,000 seconds. Figure 33 shows the UKF attitude generalized Rodriquez parameter error settles at 2,500 seconds. The increased performance is shown without fail for all subsequent plots in this section.

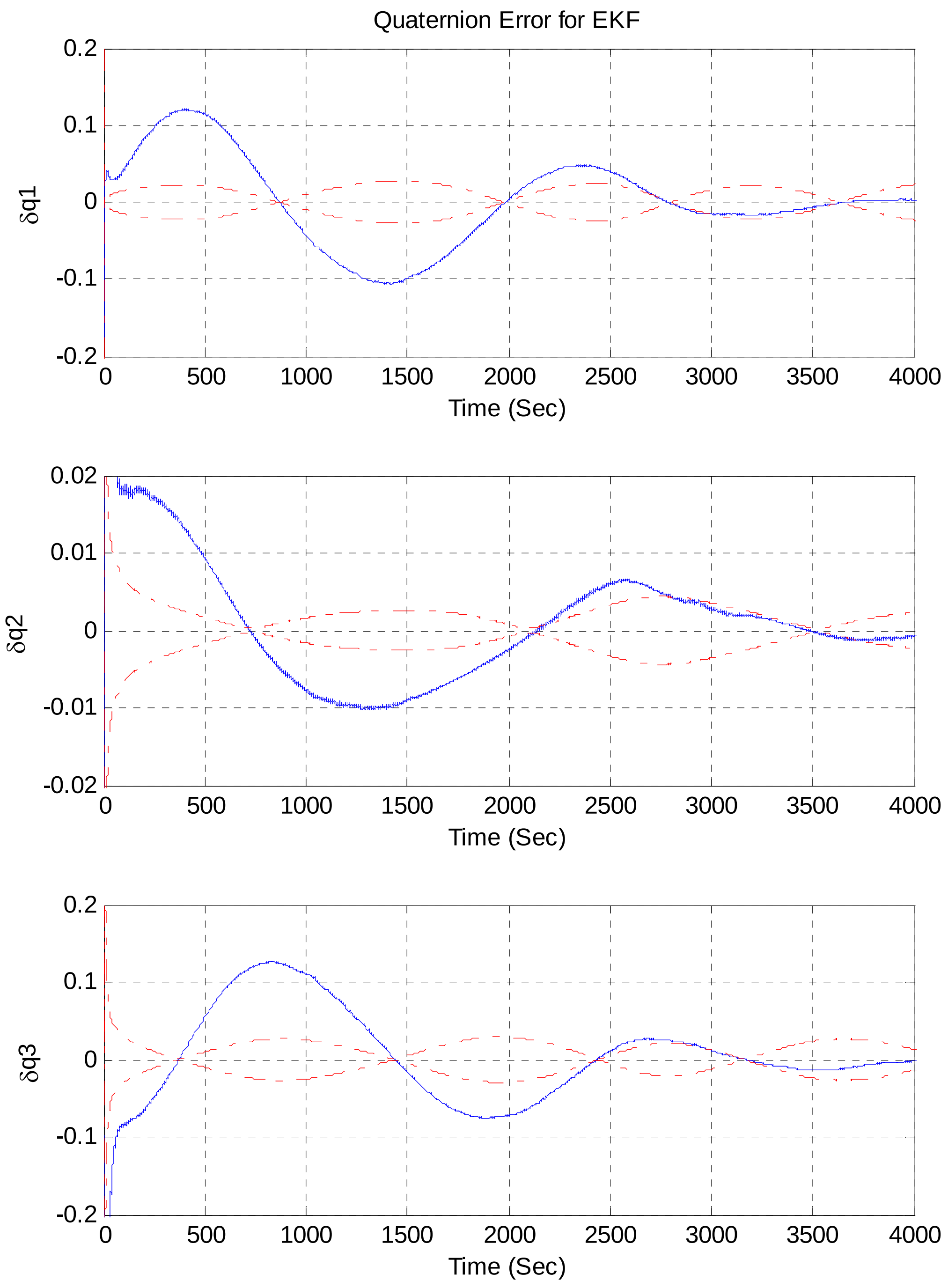


Figure 32. Simulation 2 Quaternion Attitude Error with 3σ Bounds for EKF

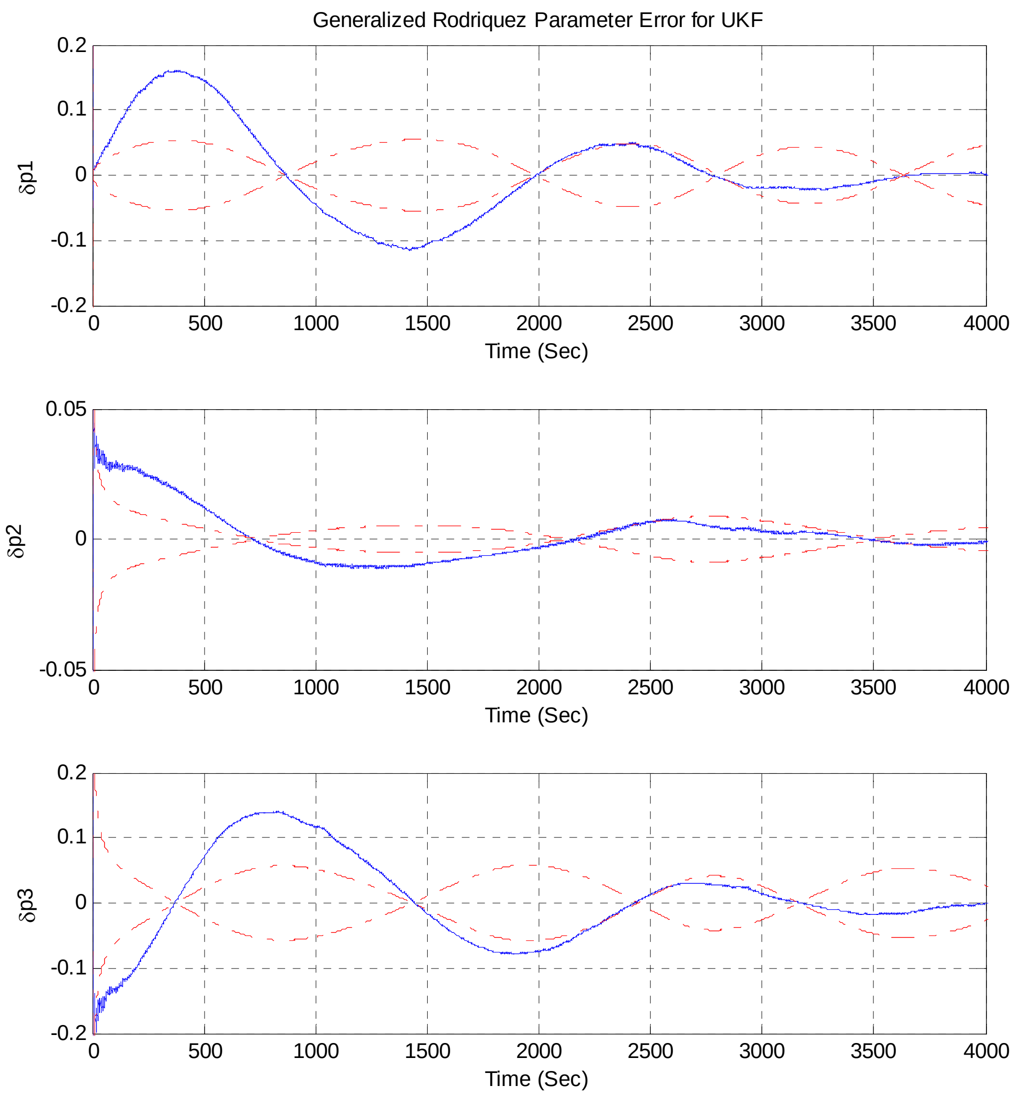


Figure 33. Simulation 2 Generalized Rodriguez Parameter Attitude Error with for 3σ Bounds for UKF

Figures 34 and 35 show that the bias for the EKF settles at approximately 2,700 seconds while the UKF bias settles at 1,700 seconds.

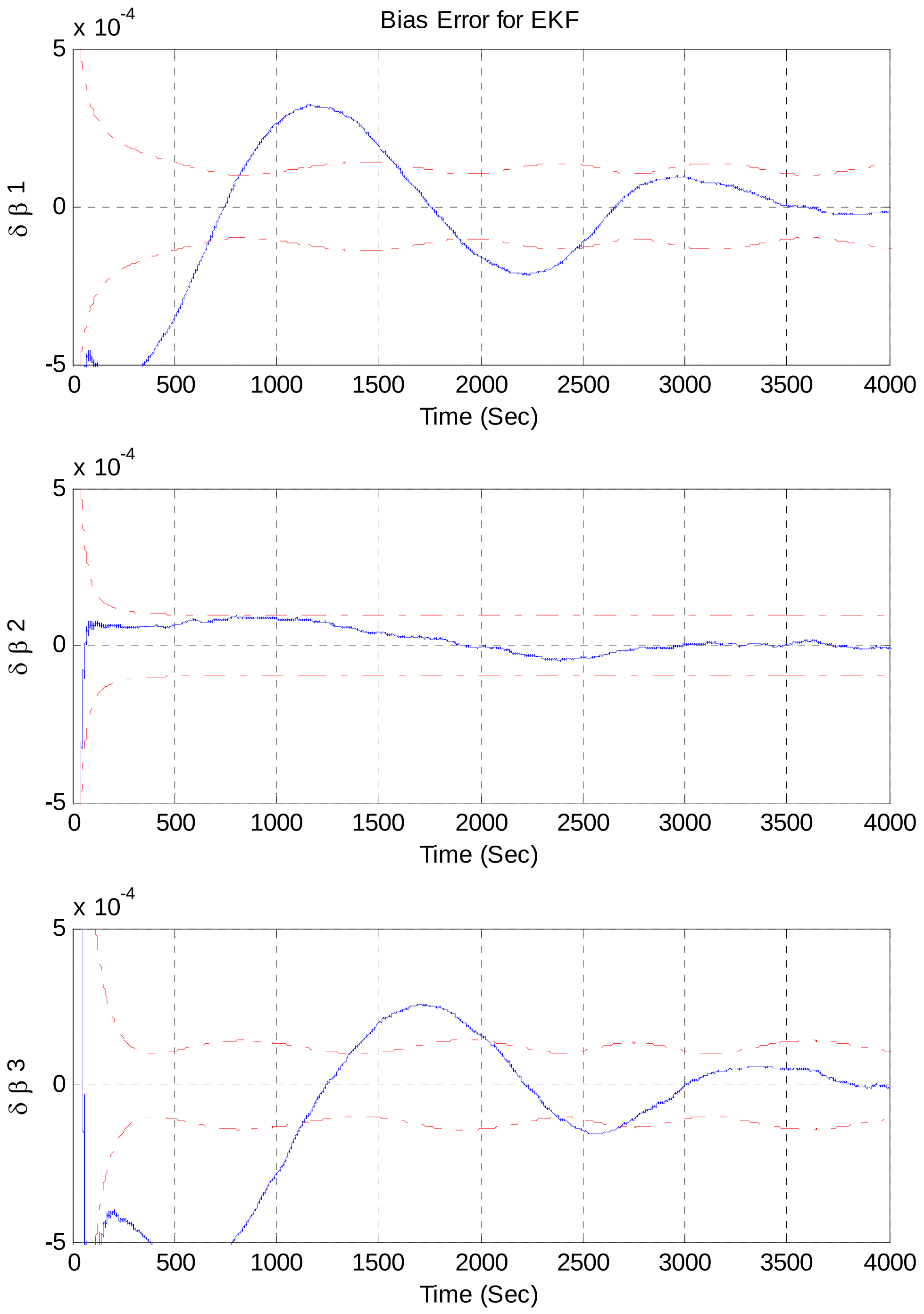


Figure 34. Simulation 2 EKF Bias Errors with 3σ Bounds

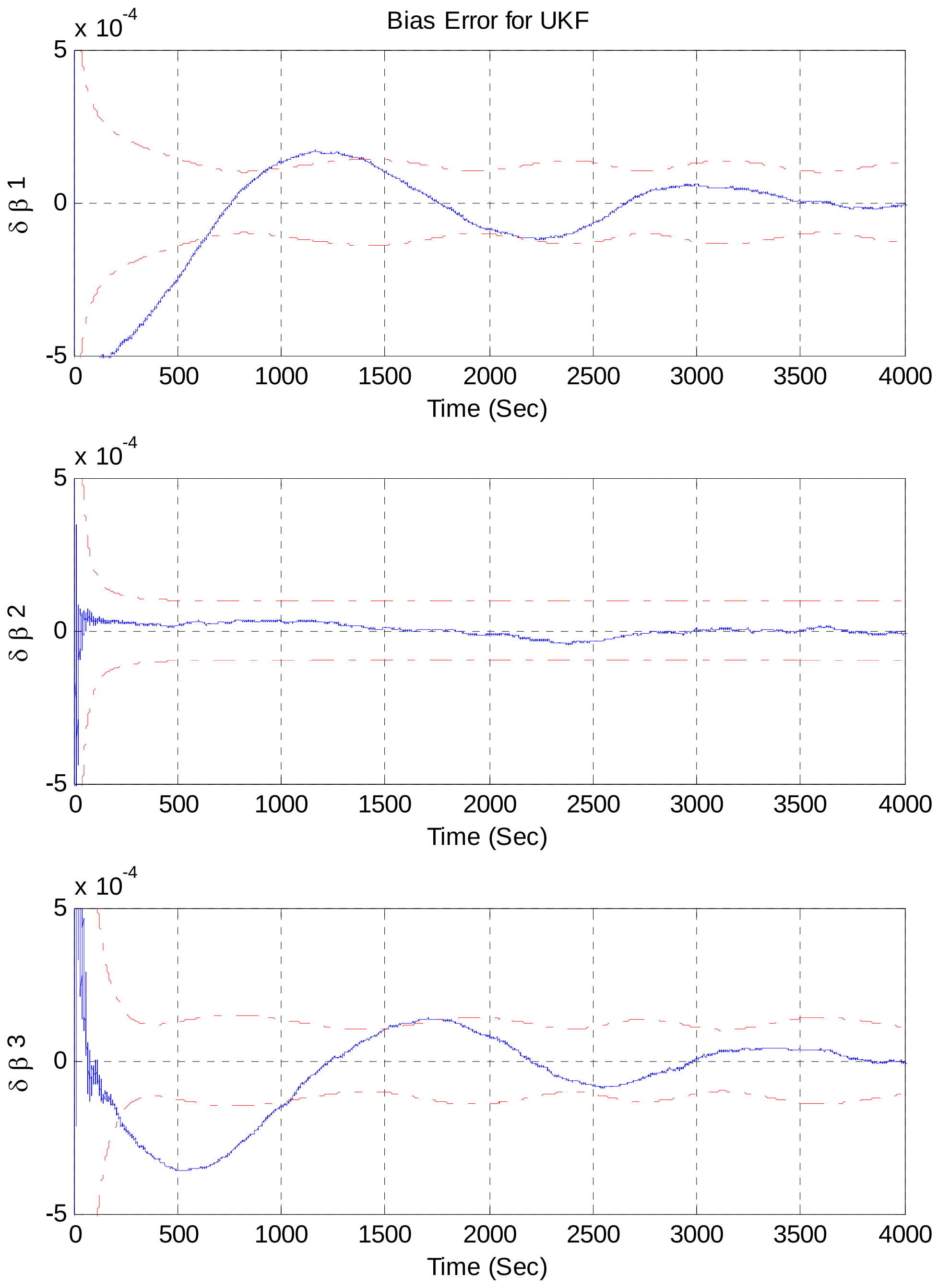


Figure 35. Simulation 2 UKF Bias Errors with 3σ Bounds

Finally, we can see in Figure 36 that the normalized attitude error of the UKF is much better. Similarly, this is shown in Figure 37 with the comparison of the normalized bias errors.

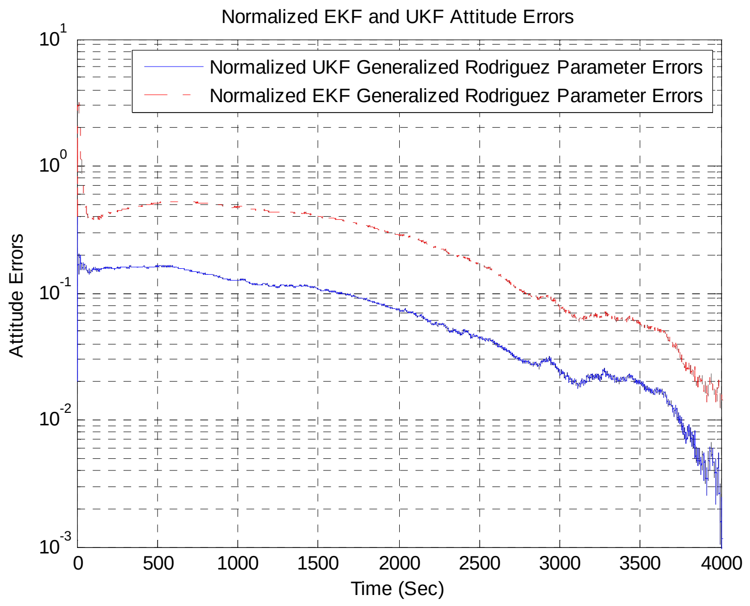


Figure 36. Comparison of EKF and UKF Normalized Attitude Errors for Simulation 2

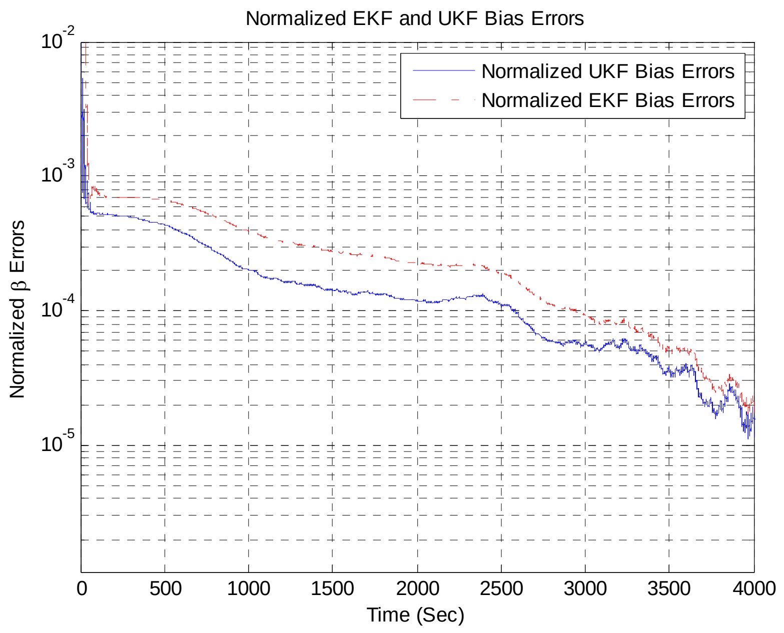


Figure 37. Comparison of EKF and UKF Normalized Bias Errors for Simulation 2

D. DISCUSSION OF RESULTS AGAINST PREVIOUS LITERATURE

Much of the work on the UKF was researched [4]. In this paper, Crassidis and Markley discuss the performance of the UKF as it applies to the spacecraft attitude determination problem. Figure 38 is taken from [4] and shows many similarities to the spacecraft model designed for this thesis. Although the results are not identical, they shown very similar trends and performance characteristics. It should be noted that the initial error conditions used for simulations in [4], were much larger and are used here to highlight the differences in accuracy. The simulations done for this thesis were set at 4,000 seconds.

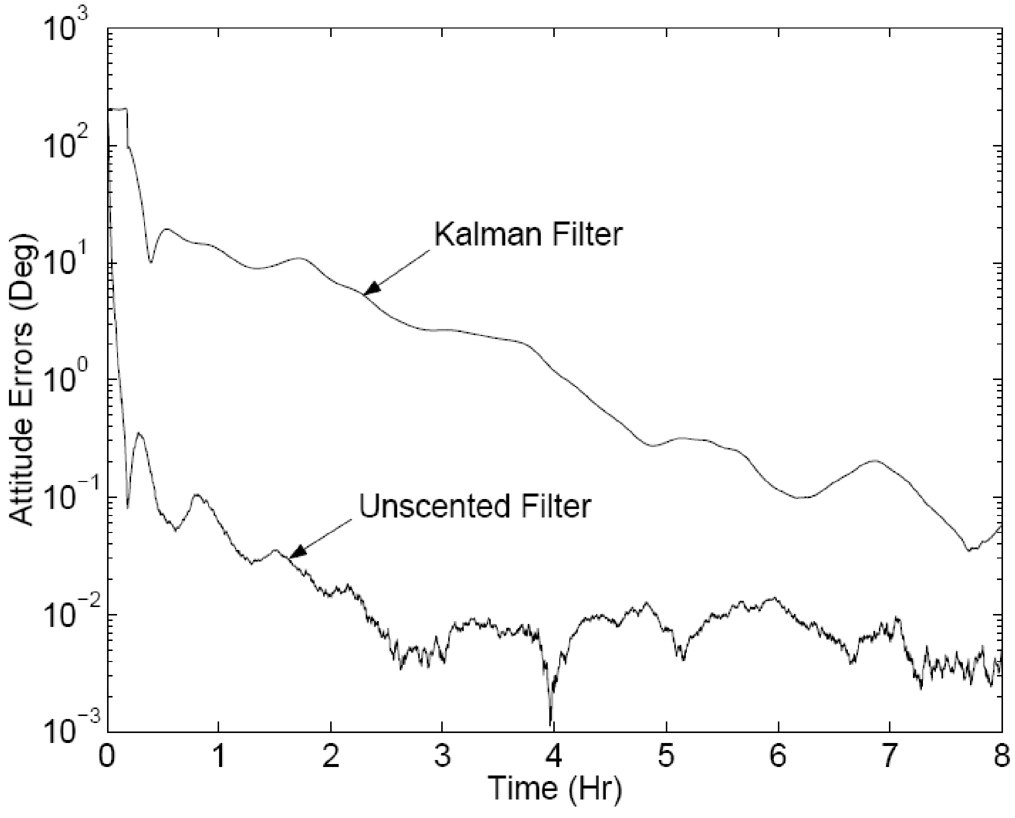


Figure 38. Norm of Attitude Errors, from [4]

V. CONCLUSION

A. SUMMARY

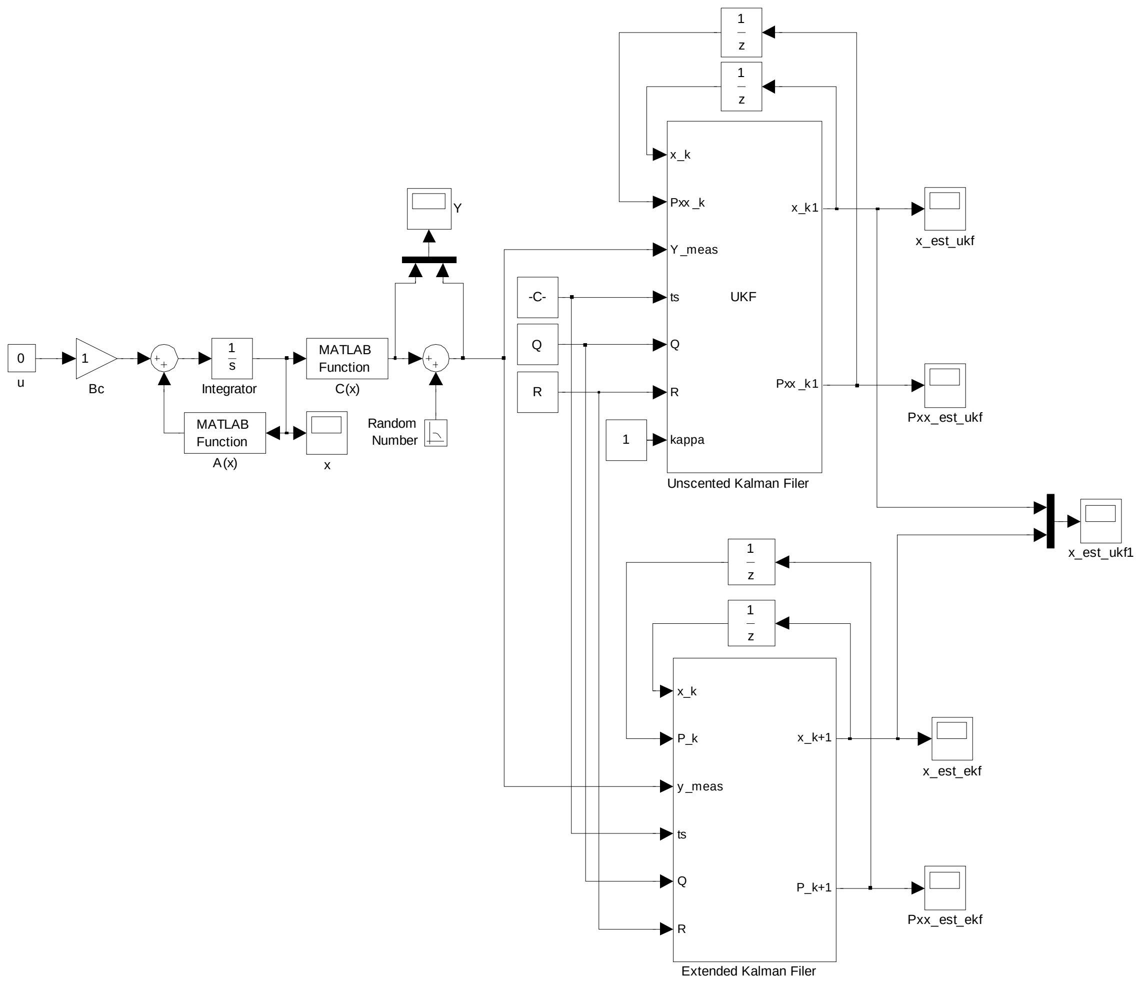
The results from the simulations clearly show that the UKF developed here is more accurate than the EKF [2]. Both the EKF and UKF were rigorously tested and validated against previous research papers. These results show both that the UKF is largely better for nonlinearities, but that the EKF performs rather well. To take advantage of the UKF, large nonlinearities must be present in the physical dynamics of the system. In summary, we have shown that the UKF has a lower expected error than the EKF for all instances of spacecraft attitude determination. We showed, in the pendulum problem, that as the nonlinearity of the dynamics increase, that the UKF shows increased performance over the EKF. However, for slightly nonlinear or linear estimation, the EKF performs well and will provide accurate solutions. The one remaining question is the computational expense that the extra computations cost. In our simulations the UKF performed approximately 2.4 times slower than the EKF, which was consistent with [4]. As the optimization of any process is measured by a cost function, one must evaluate and prioritize the resources available. Literature tells us that the UKF has 2.5 times the cost in computational time of the EKF [4]. For spacecraft with relaxed attitude-control requirements and low computational power, it could be argued that the EKF could perform sufficiently without the added expense. However, the UKF can certainly be used in the worst case conditions, such as partial loss of attitude control, and in the "lost in space" scenario where anomalies in the separation event from the launch vehicle imparts a large torque on the spacecraft hurling it into an unwanted orientation. These scenarios, although somewhat unlikely, mostly likely cannot be recovered from using an EKF estimator.

B. FUTURE WORK

This thesis presents partial validation of the UKF and EKF estimators. Although the results are favorable and largely resemble other research papers, a more realistic simulation would require hardware. Further developments in the model can also be applied. A high order magnetic field model could be implemented if the computing resources were available. Most importantly, Monte Carlo simulations should be run to show the full performance characteristics of both filters. All of this work is completely possible for further thesis students and laboratory research such as the currently being performed in the Nanosatellite Advance Concepts Laboratory.

APPENDIX A – SIMPLE PENDULUM SIMULATION

SIMULINK Block Diagram - Simple Pendulum Simulation



Model Initialization Parameters

time_step=0.05;
sigmanoise=1e-2;
R=sigmanoise^2;
Q=diag([0.00001,0.00001,0.001]);

fncA.m

```
function deriv = fncA(x)
m=50;
g=9.81;
l=0.1;

deriv=zeros(3,1);
deriv(1,1)=x(2,1);
deriv(2,1)=-(m*g*l*sin(x(1)))/(x(3)+m*l^2);
```

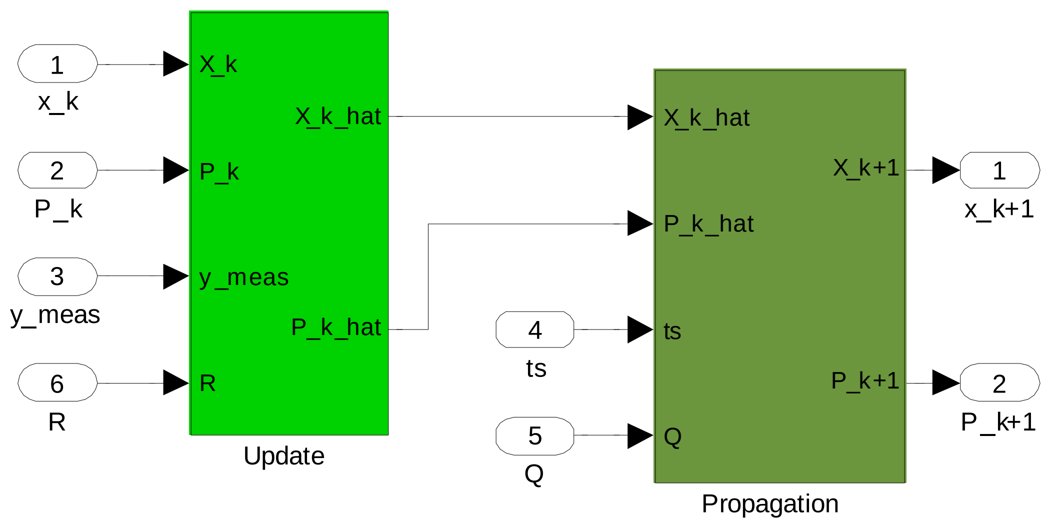
fncC.m

```
%#eml
function y = fncC(x)
Bo=0.5;
alf=30*pi/180;

y=Bo*sin(x(1,1)+alf);
```

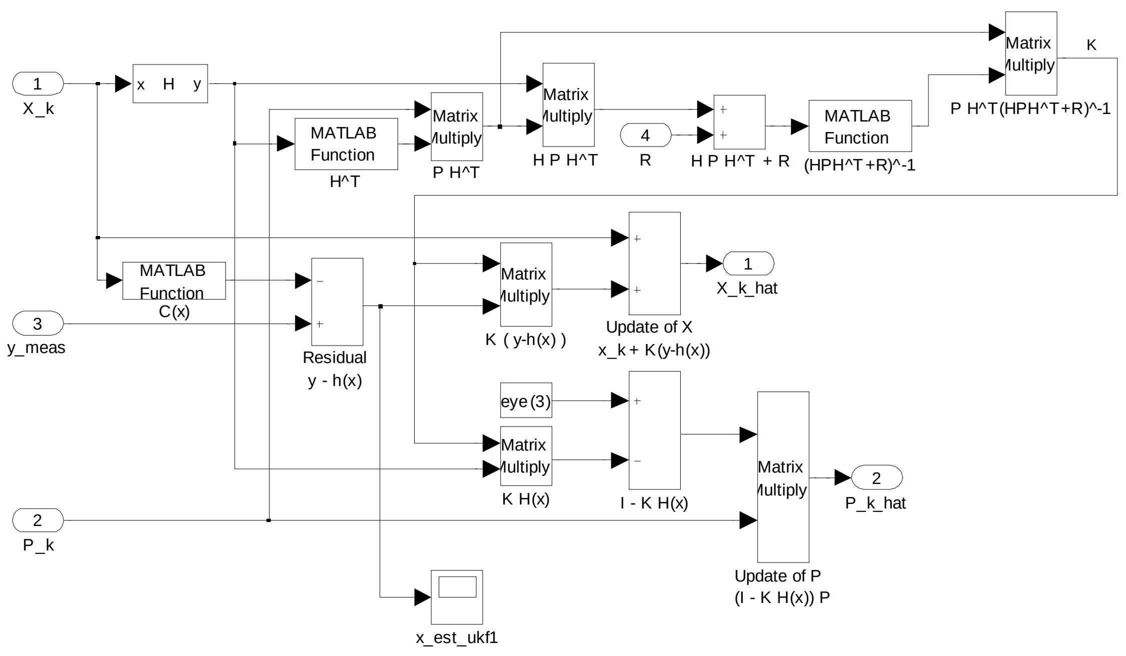
Unscented Kalman Filter – Embedded Matlab Block

```
%#eml

function [x_k1,Pxx_k1] = UKF(x_k,Pxx_k,Y_meas,ts,Q,R,kappa)
% This block supports the Embedded MATLAB subset.
% See the help menu for details.

Dx=size(x_k,1);
Dy=size(Y_meas,1);
NSig=2*Dx+1;
sig_x=(chol((Dx+kappa)*Pxx_k))';
%%%%%%%%%%%%%%%%%%%%%
x_sig_k=x_k*ones(1,NSig)+[zeros(Dx,1) sig_x -sig_x];
RK=zeros(Dx,4);
x_sig_k1=zeros(Dx,NSig);
y_sig_k1=zeros(Dy,NSig);

for i=1:NSig
    RK(:,1)=fncA(x_sig_k(:,i));
    RK(:,2)=fncA(x_sig_k(:,i)+1/2*ts*RK(:,1));
    RK(:,3)=fncA(x_sig_k(:,i)+1/2*ts*RK(:,2));
    RK(:,4)=fncA(x_sig_k(:,i)+ts*RK(:,3));
    x_sig_k1(:,i)=x_sig_k(:,i)+1/6*ts*RK*[1 2 2 1]';
    y_sig_k1(:,i)=fncC(x_sig_k1(:,i));
end

W=ones(NSig,1)/(2*(Dx+kappa));
W(1,1)=kappa/(Dx+kappa);

x_k1p=x_sig_k1*W;
y_k1p=y_sig_k1*W;

Pxx_k1p=Q;
Pyy_k1p=R;
Pxy_k1p=zeros(Dx,Dy);

for i=1:NSig
    xdif=x_sig_k1(:,i)-x_k1p;
    ydif=y_sig_k1(:,i)-y_k1p;
    Pxx_k1p=Pxx_k1p+xdif*xdif'*W(i,1);
    Pyy_k1p=Pyy_k1p+ydif*ydif'*W(i,1);
    Pxy_k1p=Pxy_k1p+xdif*ydif'*W(i,1);
end

K=Pxy_k1p/Pyy_k1p;
Pxx_k1=Pxx_k1p-K*Pxy_k1p';
x_k1=x_k1p+K*(Y_meas-y_k1p);
```

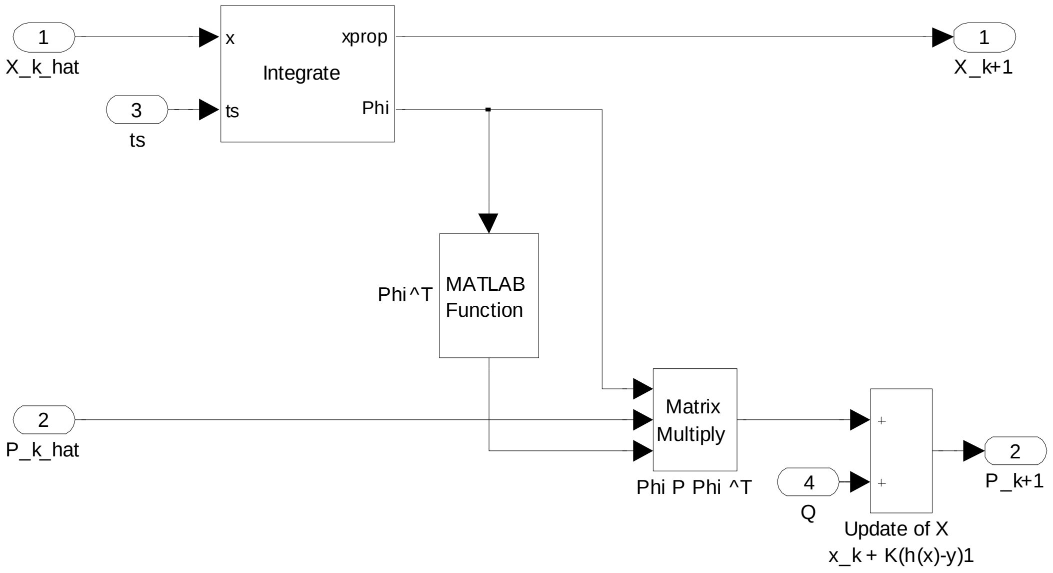
Extended Kalman Filter Block



Extended Kalman Filter Block-Update Block



Extended Kalman Filter –Propagation Block



Integrate- Embedded Matlab Block

```
function [xprop, Phi] = Integrate(x,ts)
% This block supports the Embedded MATLAB subset.
% See the help menu for details.

KX1=fncA(x);
KP1=F(x);
KX2=fncA(x+1/2.0*ts*KX1);
KP2=F(x+1/2.0*ts*KX1)*(eye(3)+1/2.0*ts*KP1);
KX3=fncA(x+1/2.0*ts*KX2);
KP3=F(x+1/2.0*ts*KX2)*(eye(3)+1/2.0*ts*KP2);
KX4=fncA(x+ts*KX3);
KP4=F(x+ts*KX3)*(eye(3)+ts*KP3);

xprop = x + 1/6.0*ts*(KX1+2*KX2+2*KX3+KX4);
Phi = eye(3) + 1/6.0*ts*(KP1+2*KP2+2*KP3+KP4);
return

function deriv=F(x)
m=50;
g=9.81;
l=.1;

deriv=zeros(3,3);
deriv(1,2)=1;
deriv(2,1)=-m*g*l*cos(x(1,1))/(x(3,1)+m*l^2);
deriv(2,3)=m*g*l*sin(x(1,1))/(x(3,1)+m*l^2)^2;
return
```

pendulum_plots.m

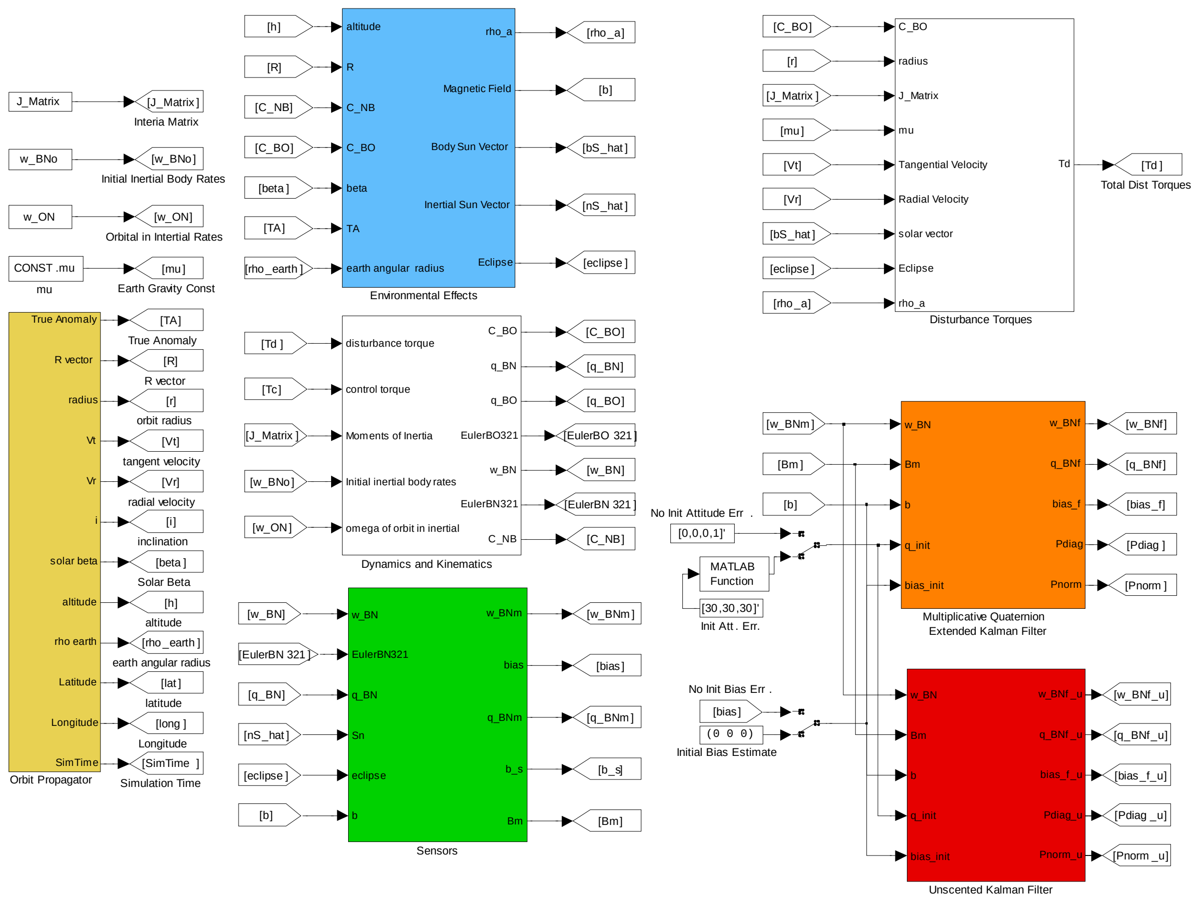
```
x_true=zeros;
[m n p]=size(x);
x_true(1:m,1:p)=x(1:m,1,1:p);
x_true=x_true';


%% State 1: Theta

%EKF Error
figure(1)
plot(t, (x_est_ekf(:,1)-x_true(:,1))*180/pi);
grid on
title('EKF Angle Error (Deg)')
xlabel('Time (S)')
ylabel('Angle Error (Deg)')
% ylim([-5 5])
hold on
plot(t, 3*sqrt(Pxx_est_ekf(:,1))*180/pi','-.');
plot(t, -3*sqrt(Pxx_est_ekf(:,1))*180/pi','-.');
%UKF Error
```

```
figure(2)
plot(t, (x_est_ukf(:,1)-x_true(:,1))*180/pi);
grid on
title('UKF Angle Error (Deg)')
xlabel('Time (S)')
ylabel('Angle Error (Deg)')
% ylim([-5 5])
hold on
plot(t, 3*sqrt(Pxx_est_ukf(:,1))*180/pi','-.');
plot(t, -3*sqrt(Pxx_est_ukf(:,1))*180/pi','-.');

%% State 2: Omega - Angular Rate

%EKF Error
figure(4)
plot(t, (x_est_ekf(:,2)-x_true(:,2))*180/pi);
grid on
title('EKF Angular Rate Error (Deg/Sec)')
xlabel('Time (S)')
ylabel('Angular Rate Error (Deg/Sec)')
% ylim([-10 10])
hold on
plot(t, 3*sqrt(Pxx_est_ekf(:,2))*180/pi','-.');
plot(t, -3*sqrt(Pxx_est_ekf(:,2))*180/pi','-.');

%UKF Error
figure(5)
plot(t, (x_est_ukf(:,2)-x_true(:,2))*180/pi);
grid on
title('UKF Angular Rate Error (Deg/Sec)')
xlabel('Time (S)')
ylabel('Angular Rate Error (Deg/Sec)')
% ylim([-10 10])
hold on
plot(t, 3*sqrt(Pxx_est_ukf(:,2))*180/pi','-.');
plot(t, -3*sqrt(Pxx_est_ukf(:,2))*180/pi','-.');

%% State 3: Iy - Moment of Inertia

%EKF Error
figure(7)
plot(t, x_est_ekf(:,3)-x_true(:,3));
grid on
title('EKF Moment of Inertia Error')
xlabel('Time (S)')
ylabel('Moment of Inertia (kg m^2)')
% ylim([-5 5])
hold on
plot(t, 3*sqrt(Pxx_est_ekf(:,3))','-.');
plot(t, -3*sqrt(Pxx_est_ekf(:,3))','-.');

%UKF Error
figure(8)
plot(t, x_est_ukf(:,3)-x_true(:,3));
```

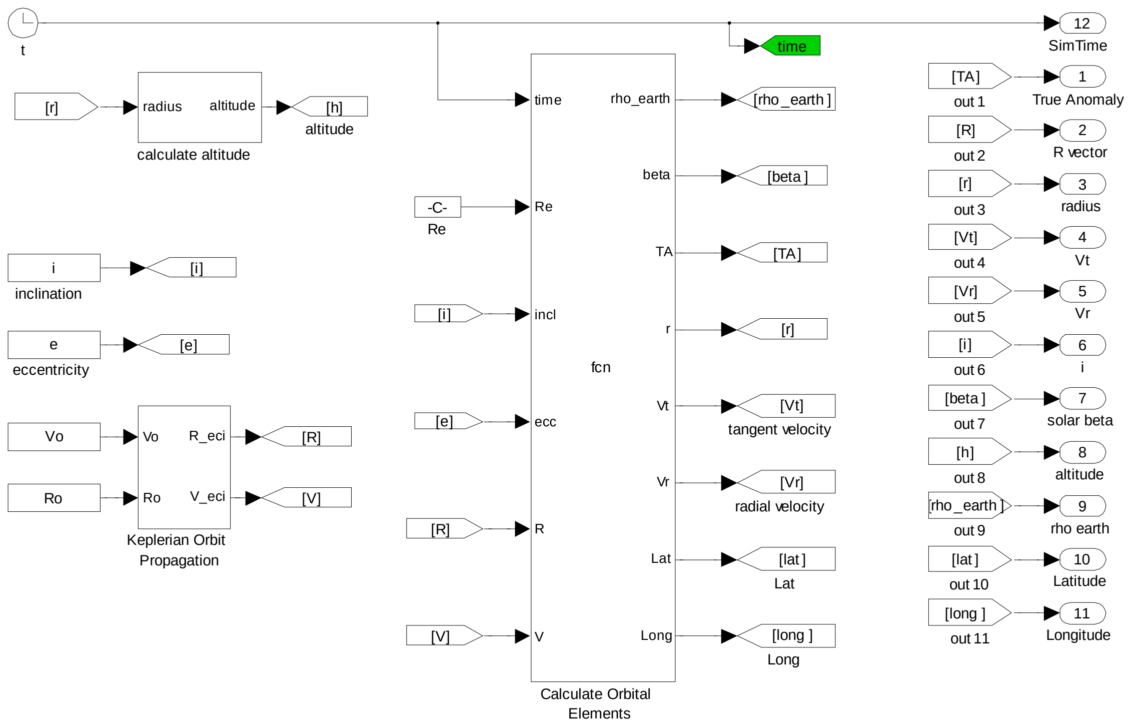
```
grid on
title('UKF Moment of Inertia Error')
xlabel('Time (S)')
ylabel('Moment of Inertia (kg m\^2)')
% ylim([-5 5])
hold on
plot(t, 3*sqrt(Pxx_est_ukf(:,3))','-.');
plot(t, -3*sqrt(Pxx_est_ukf(:,3))','-.');
```

APPENDIX B – SPACECRAFT ATTITUDE DETERMINATION SIMULATION

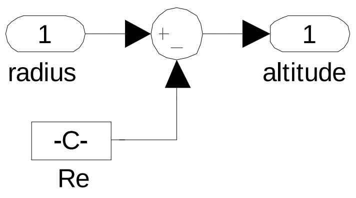
ADS_SpacecraftSim.mdl



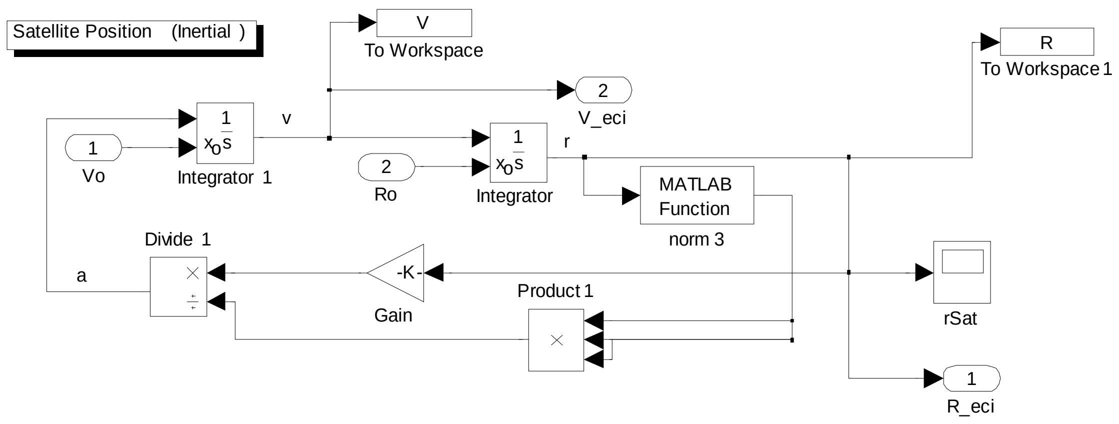
Orbit Propagator-Simulink Block



Calculate Altitude-Simulink Block



Keplerian Orbit Propagation- Simulink Block



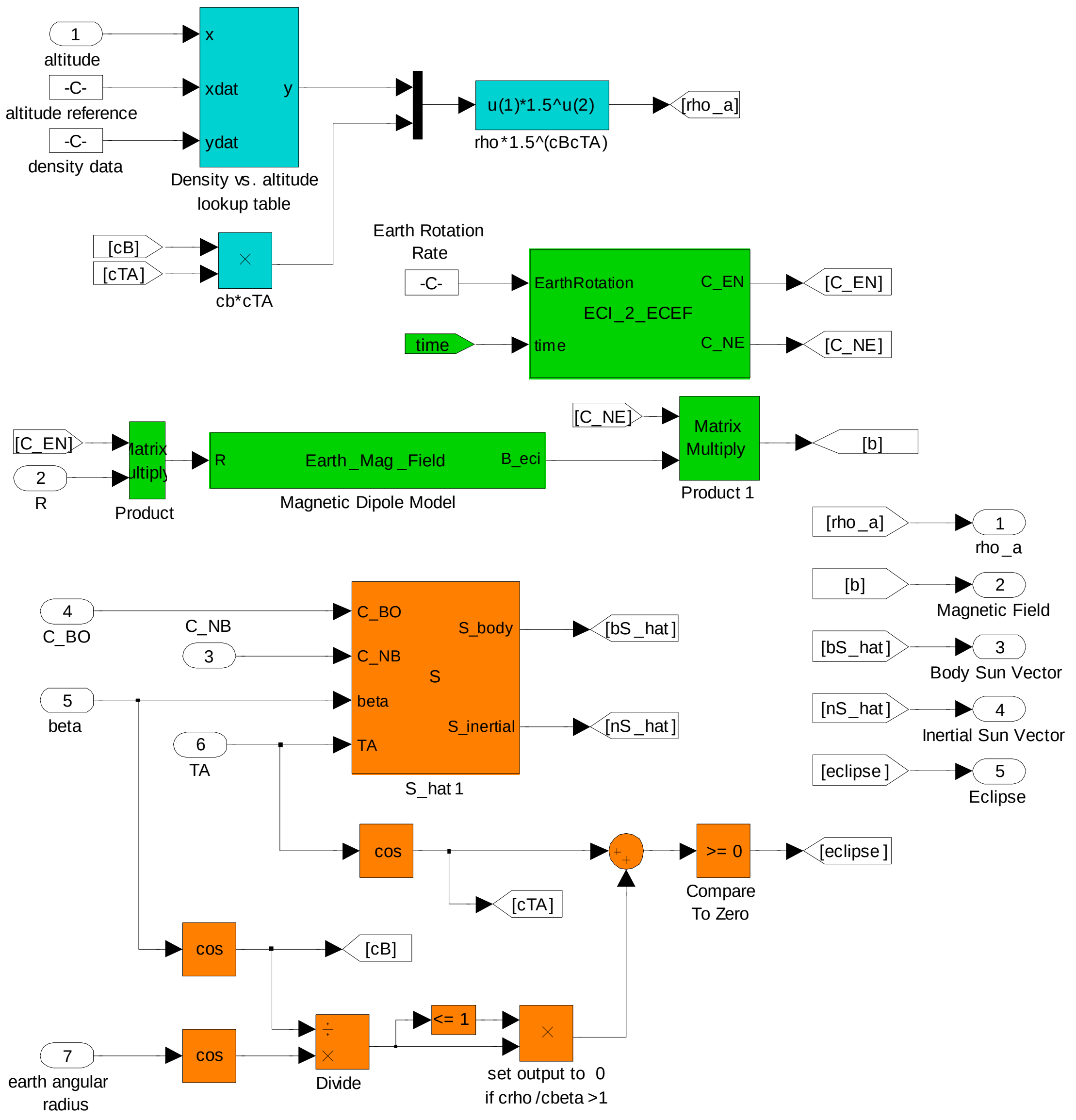
Calculate Orbital Elements – Embedded Matlab Code

```
function [rho_earth,beta,TA,r,Vt,Vr,Lat,Long] =
fcn(time,Re,incl,ecc,R,V)
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
%
% This function computes the classical orbital elements
% from the state vector (R,V) using Algorithm 4.1.  As well as
% other orbital parameters needed by the model.
%
% mu - gravitational parameter (mˆ3/sˆ2)
% R - position vector in the geocentric equatorial frame (m)
% V - velocity vector in the geocentric equatorial frame (m/s)
% r, v - the magnitudes of R and V
% vr - radial velocity component (m/s)
% H - the angular momentum vector (mˆ2/s)
% h - the magnitude of H (mˆ2/s)
% incl - inclination of the orbit (rad)
% N - the node line vector (mˆ2/s)
% n - the magnitude of N
% cp - cross product of N and R
% RA - right ascension of the ascending node (rad) not used
**************
% E - eccentricity vector
% ecc - eccentricity (magnitude of E)
% eps - a small number below which the eccentricity is
% considered to be zero
% w - argument of perigee (rad) not used
*******************************
% TA - true anomaly (rad)
% Vt - tangential velocity (m/s)
% Vr - radial velocity (m/s)
% rho_earth - earth anglular radius
% beta - beat angle (rad)
% Lat - Latitude of satellite (rad)
% Long - Longitude of satellite (rad)
% ----------------------------------------------------------
mu = 398.6004418e12;              % m^3/s^2
eps = 1.0e-10;
r = norm(R);
v = norm(V);
vr = dot(R,V)/r;
H = cross(R,V);
h = norm(H);

%   Calc inclination
%{
c = H(3)/h;
if (c < -1) && (c > 1)
    c = c - pi;
end
incl = acos(c);
%}
%   Calc right ascension of the ascending node (rad)
N = cross([0 0 1],H);
```

```
n = norm(N);

%   Calc Eccentricity

E = 1/mu*((v^2 - mu/r)*R - r*vr*V);
%ecc = norm(E);

%   True Annomoly
if ecc > eps
    c = dot(E,R)/ecc/r;
    if (c < -1) && (c > 1)
        c = c - pi;
    end
    TA = acos(c);
    if vr < 0
        TA = 2*pi - TA;
    end
else
    cp = cross(N,R);
    c = dot(N,R)/n/r;
    if (c < -1) && (c > 1)
        c = c - pi;
    end
    if cp(3) >= 0
        TA = acos(c);
    else
        TA = 2*pi - acos(c);
    end
end

%   Calculate the tangential and radial velocities
Vt = h/r;
Vr = mu/h*ecc*sin(TA);

%   Calculate earth angular radius
rho_earth = asin(Re/r);

%   Beta calcs
wb_0 = 0;
ub_0 = 0;
wb_dot = (-9.9639/86400)*rho_earth^(3.5)*cos(incl)/(1-ecc^2)^2;
wb = (wb_0 + wb_dot*time)*pi/180;
gamma = 23.442*pi/180;                   %rad
ub_dot = (0.985648/86400)*pi/180;        %rad
ub = ub_0+ub_dot*time;

beta = asin(sin(ub)*sin(gamma)*cos(incl) + ...
    cos(ub)*sin(incl)*sin(wb)-sin(ub)*cos(gamma)*sin(incl)*cos(wb));

%   For Mag Calc ----------------------------------------------
%   Calculate Earth Coordinate by Simulate the Earth's Rotation
%   Track the movement of (0 Lat, 0 Long)
PhiE = time*2*pi/(23.93*3600);
ThetaE = 0;
```

```
EarthCoord = [Re,ThetaE,PhiE];

%   Calculate Sat Coordinate in Polar
X = R(1);
Y = R(2);
Z = R(3);

Theta = atan2(sqrt(X^2+Y^2),Z);
Phi = atan2(Y,X);

SatCoord = [r,Theta,Phi];

%   Calculate Lat and Long
Theta0 = pi/2-SatCoord(2);
Phi0 = SatCoord(3);

Phi1 = EarthCoord(3);

Lat = Theta0;
Long = (Phi0-Phi1);

if Long > pi
    Long = Long-2*pi;
end
if Long < -pi
    Long = Long+2*pi;
end
```

Environmental Effects- Simulink Block



EFI 2 ECEF-Embedded Matlab Code

```
function [C_EN,C_NE]= ECI_2_ECEF(EarthRotation,time)
%   transformation of eci to ecef coordinates

theta = EarthRotation(3)*time;

C_EN = [ cos(theta) -sin(theta) 0;
         sin(theta) cos(theta) 0;
         0          0          1];
C_NE =C_EN';
```

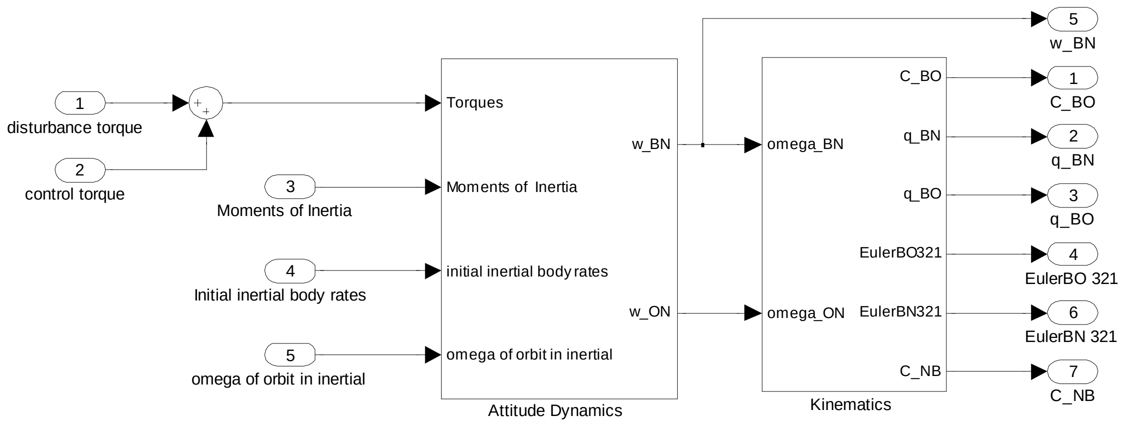
Magnetic Dipole Model - Embedded Matlab Code

```
function B_eci  = Earth_Mag_Field(R)
% Magnetic dipole model - in Tesla

theta = 11.7;                                                % deg
DCM = [1 0 0; 0 cosd(theta) sind(theta); 0 -sind(theta) cosd(theta)]';
mu0 = 4*pi*10e-7;                                            % N/Amp^2
M = DCM*[0 0 8e22]';                                         % A*m^2

r = norm(R);
r_hat = R/r;

B_eci = mu0*(3*dot(M,r_hat)*r_hat-M)/(4*pi*r^3);
```

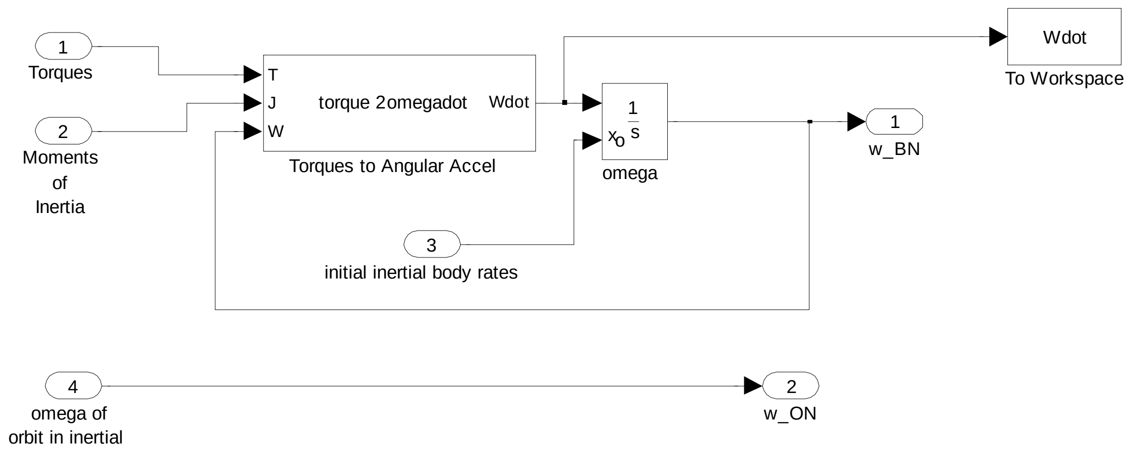
S_hat1 – Embedded Matlab Code

```
function [S_body,S_inertial] = S(C_BO,C_NB,beta,TA)
% This block supports an embeddable subset of the MATLAB language.
% See the help menu for details.

B=beta;

S_orbit = [cos(B)*sin(TA);...
              sin(B);...
           cos(B)*cos(TA)];

S_body = C_BO*S_orbit;

S_inertial = C_NB*S_body;
```

Dynamics and Kinematics - Simulink Block



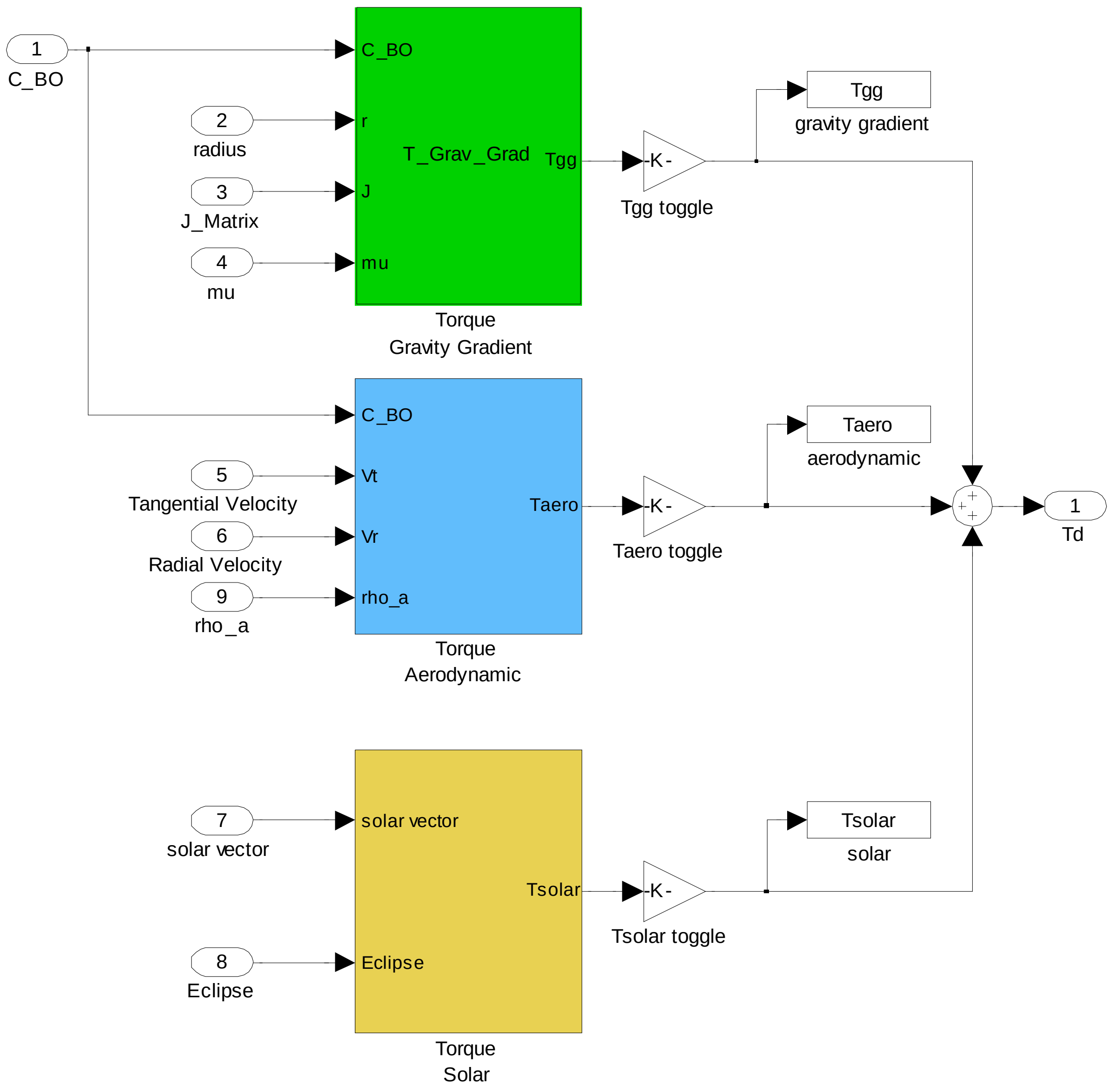
Attitude Dynamics - Simulink Block



torque2omegadot.m EML

```
function Wdot = torque2omegadot(T, J, W)

% This function takes input of applied torque (T) in component
% elements, current angular velocity (W) in component
% elements, and the moment-of-inertia matrix (J) as a diagonal
% matrix containing the MOIs for the principal axes of the body
% along the diagonal.  Angular acceleration is then computed and
% output as a 3x1 vector (Wdot).

Wx = W(1); Wy = W(2); Wz = W(3);
Jxx = J(1,1); Jyy = J(2,2); Jzz = J(3,3);
Tx = T(1); Ty = T(2); Tz = T(3);

wdotx = (Tx-(Jzz-Jyy)*Wz*Wy)/Jxx;
wdoty = (Ty-(Jxx-Jzz)*Wx*Wz)/Jyy;
wdotz = (Tz-(Jyy-Jxx)*Wy*Wx)/Jzz;

Wdot = [wdotx; wdoty; wdotz];
```

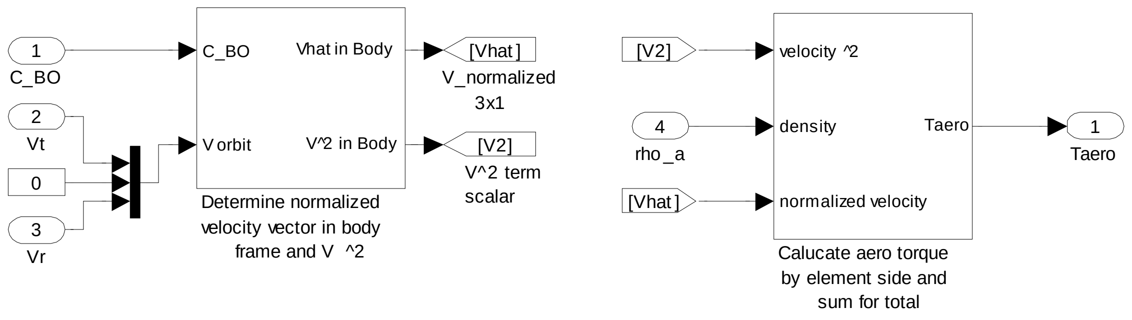
Disturbance Torques– Simulink Block



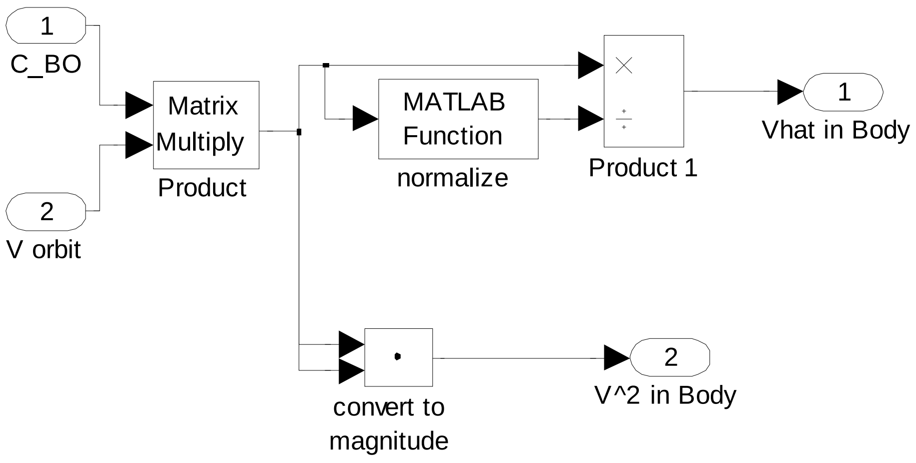
Torque Gravity Gradient – EML

```
function Tgg  = T_Grav_Grad(C_BO, r, J, mu)
% T_Grav_Grad takes inputs of the spacecraft inertia matrix (J),
% current orbit radius (r) in m, and the Orbit-to-Body Frame DCM (C_BO)
% to calculate the gravity gradient torque in the body frame (Tgg) and
% orbit frame (T_o).  The orb_vec vector defines which orbit frame axis
% is aligned with the force producing the torque.  In this case, the z-
axis
% points along nadir in the orbit frame, and corresponds to the r-
vector
% direction.

orb_vec = [0; 0; 1];

c = C_BO*orb_vec;

Jxx = J(1,1);
Jyy = J(2,2);
Jzz = J(3,3);

Tgg = 3*mu/r^3*[(Jzz-Jyy)*c(2)*c(3);...
                (Jxx-Jzz)*c(1)*c(3);...
                (Jyy-Jxx)*c(1)*c(2)];
```

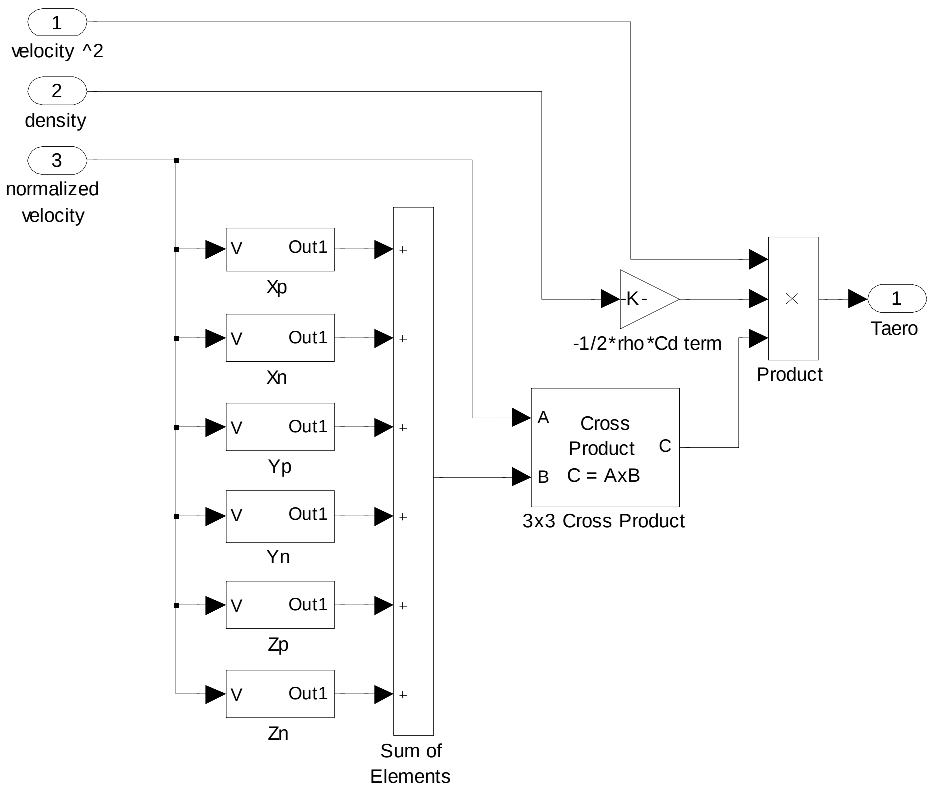
Torque Aerodynamic – Simulink Block



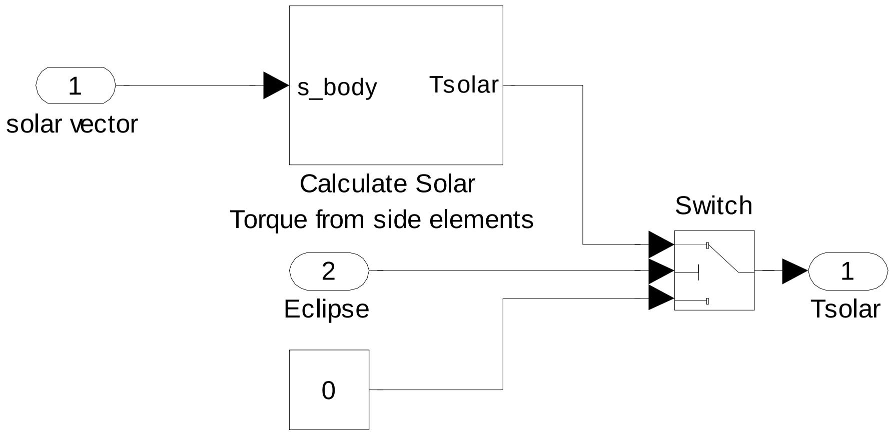
Determine Normalized Velocity Vector – Simulink Block



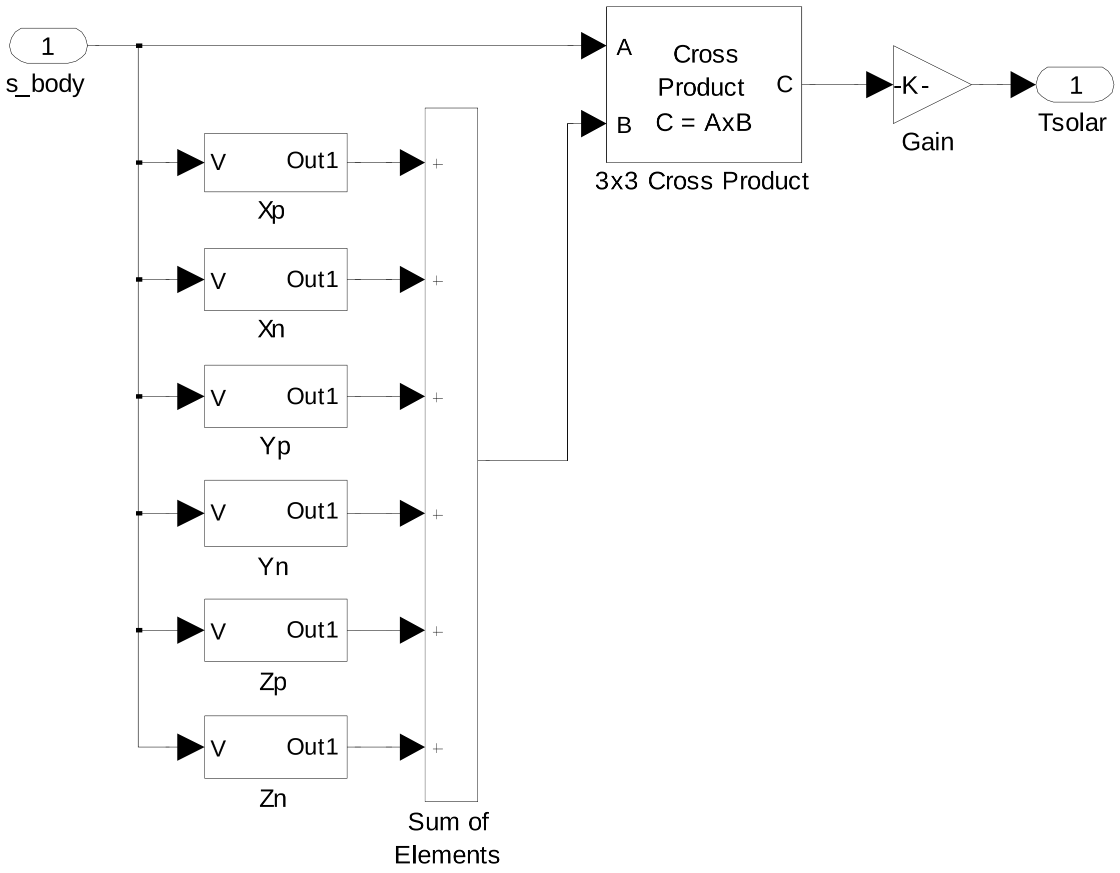
Calculate Aero Torque – Simulink Block



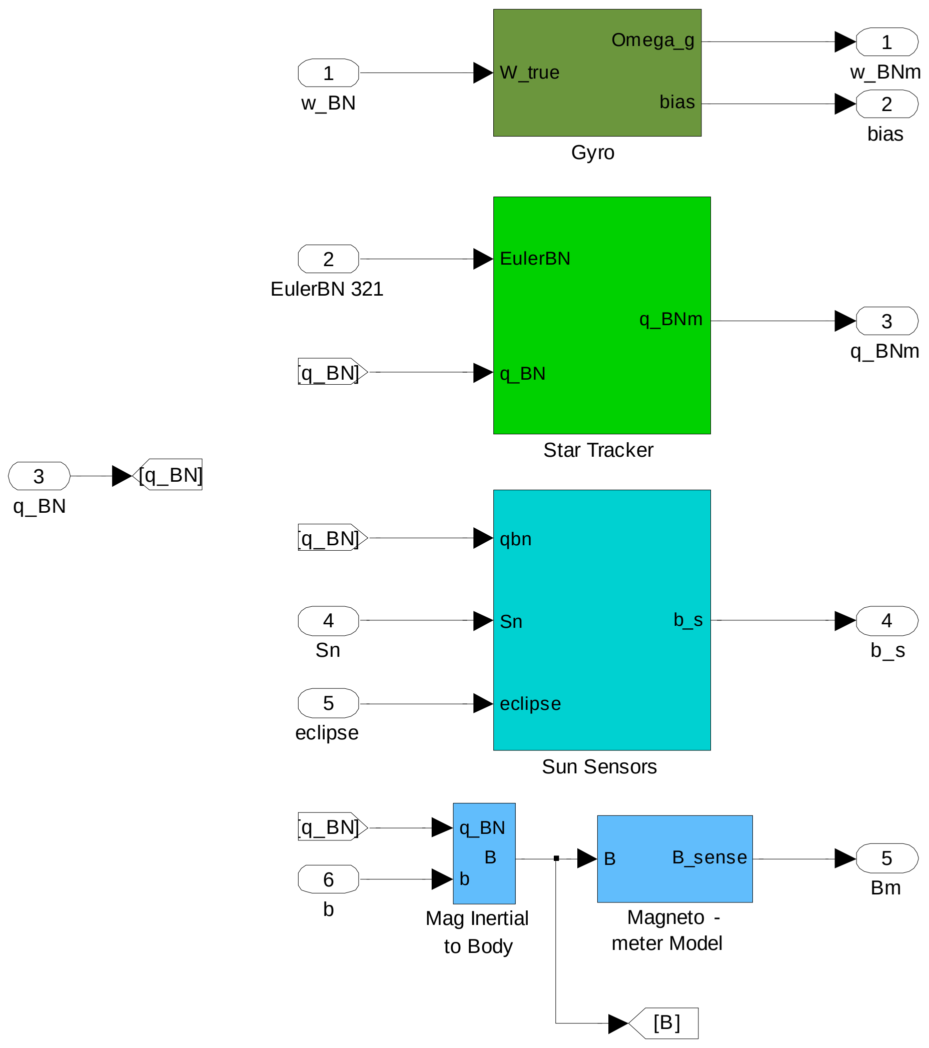
Solar Torque – Simulink Block



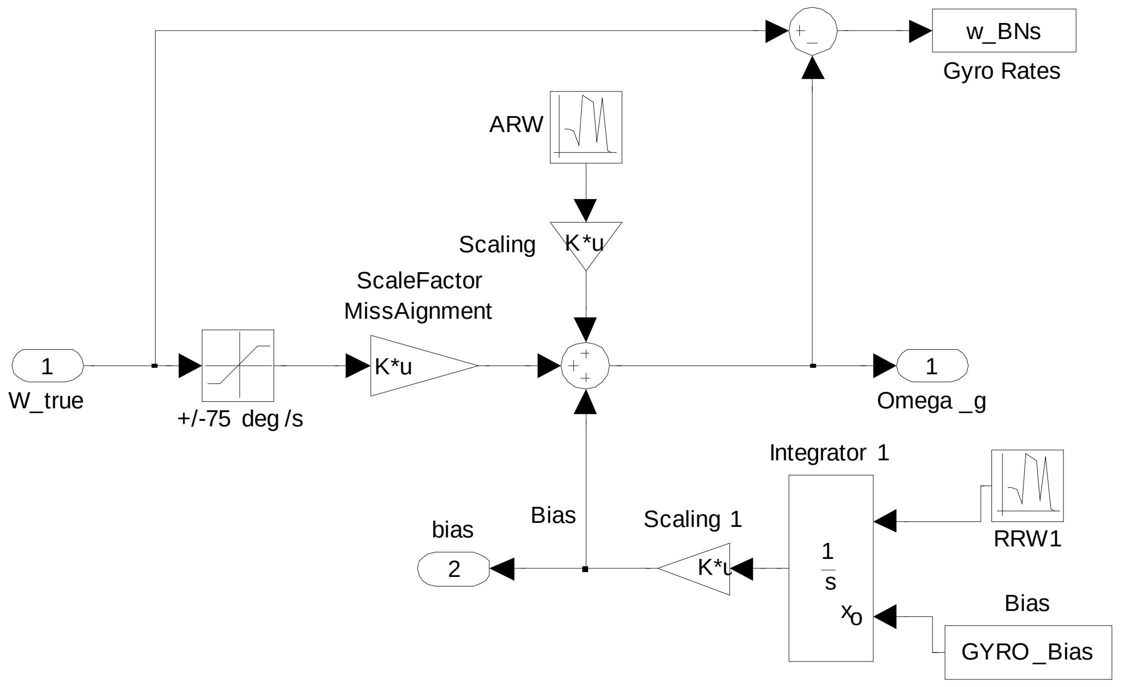
Calculate Solar Torque – Simulink Block



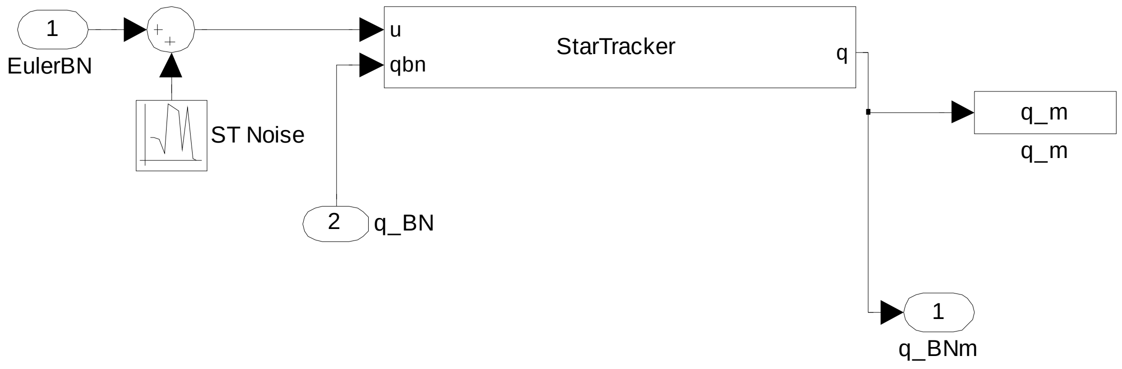
Sensors – Simulink Block



Gyro – Simulink Block



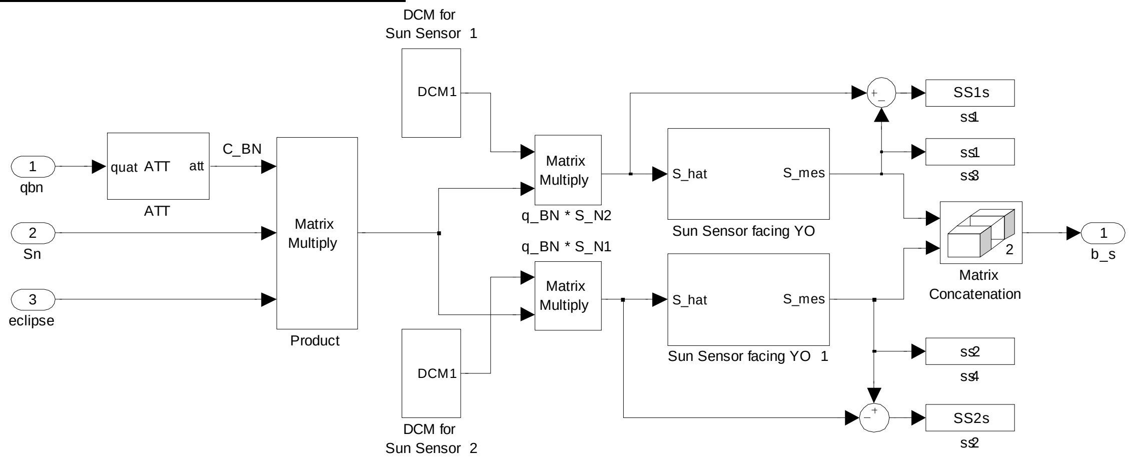
Star Tracker – Block Diagram



Star Tracker – Embedded Matlab Code

```
function q = StarTracker(u,qbn)

ph=u(1)/2;    th=u(2)/2;    ps=u(3)/2;

sph = sin(ph);  sth = sin(th);  sps = sin(ps);
cph = cos(ph);  cth = cos(th);  cps = cos(ps);

q = [sph*cth*cps-cph*sth*sps;
     cph*sth*cps+sph*cth*sps;
     cph*cth*sps-sph*sth*cps;
     cph*cth*cps+sph*sth*sps];

[Y I]=max(abs(q));
q=q/norm(q)*sign(q(I,1)/qbn(I,1));
```

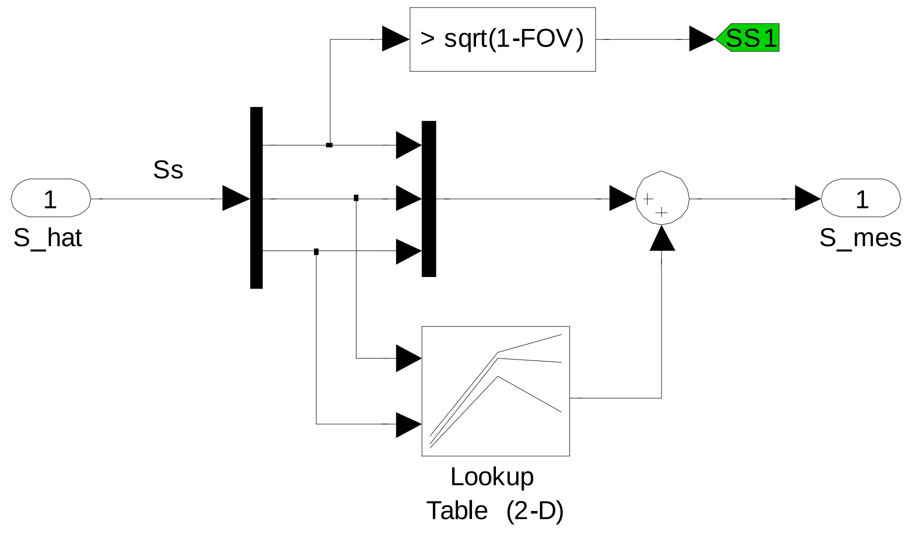
Sun Sensors – Block Diagram



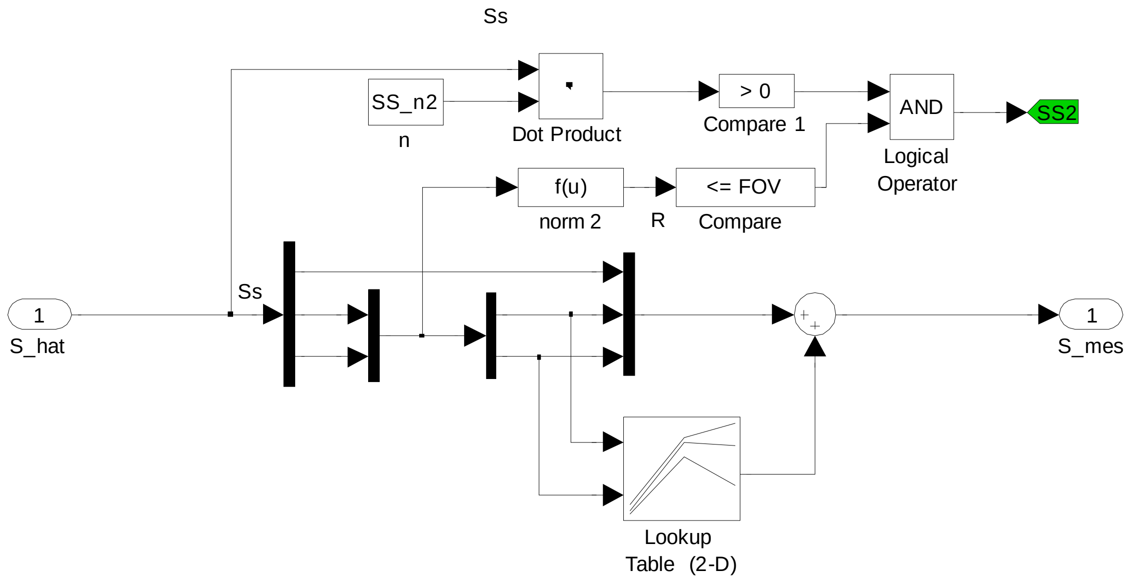
ATT.m

```
function att = ATT( quat )
att = transpose(XI(quat)) * PSI(quat);
return
```

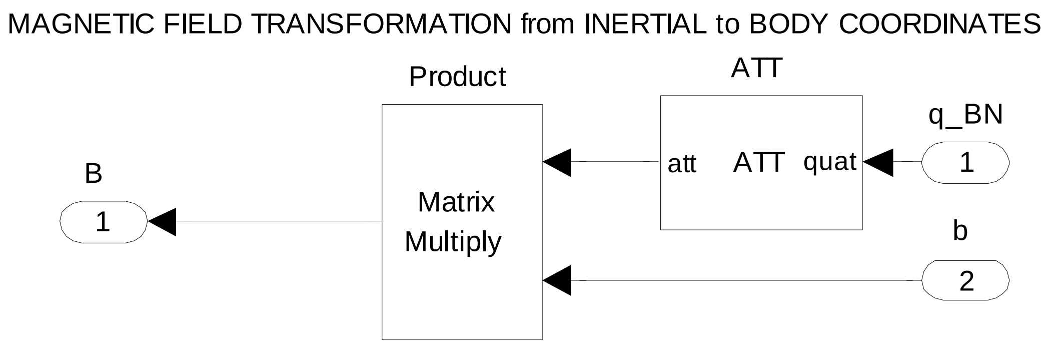
Sun Sensor Facing YO – Simulink Block



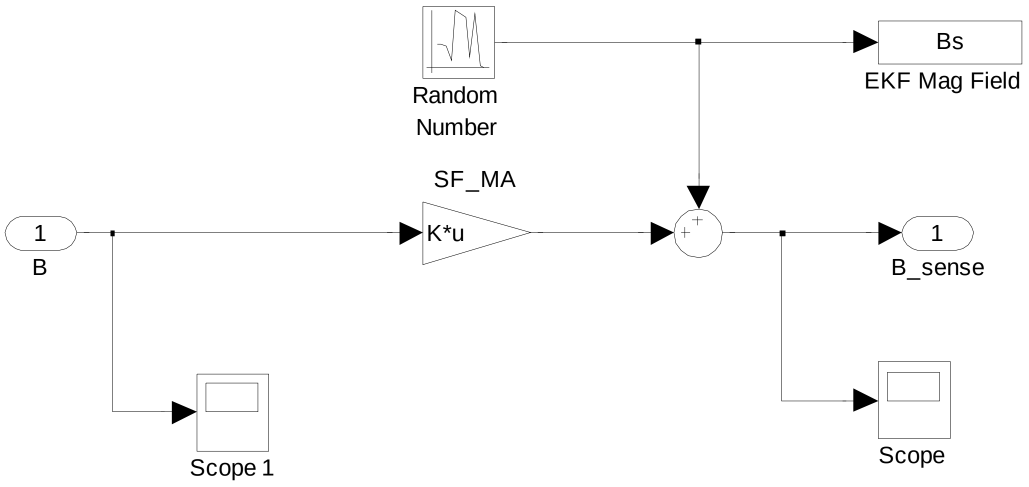
Sun Sensor Facing YO1 – Block Diagram



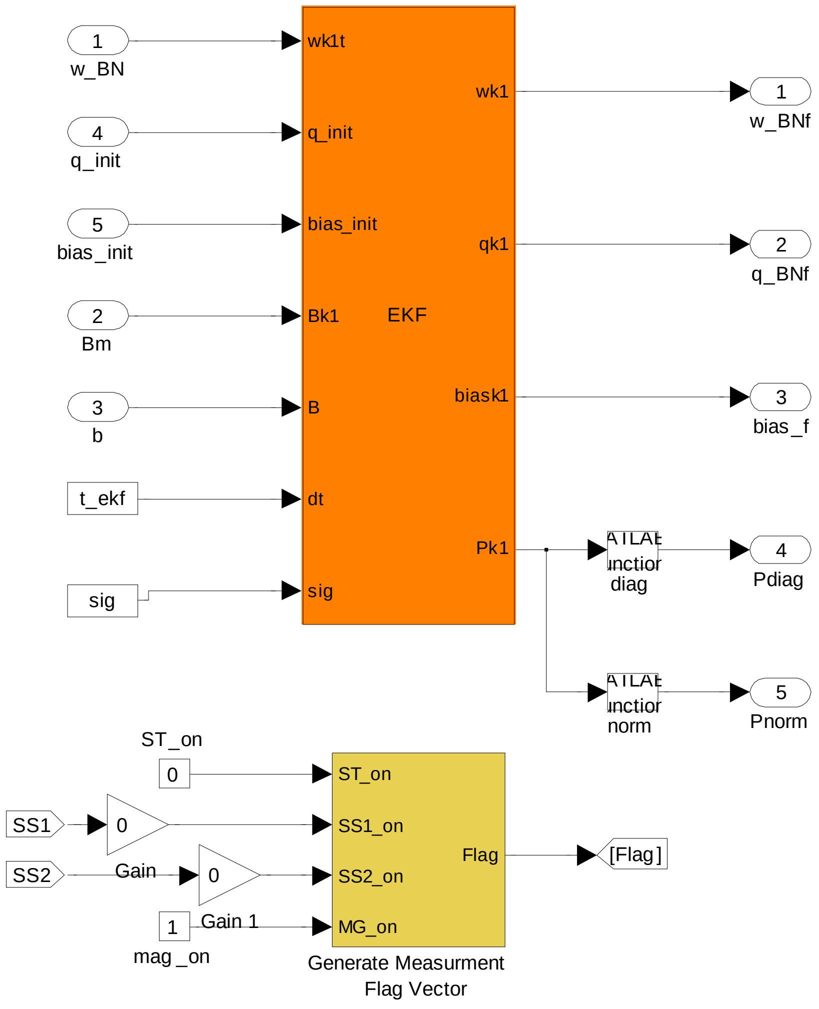
Mag Inertial to Body – Simulink Block


MAGNETIC FIELD TRANSFORMATION from INERTIAL to BODY COORDINATES
Product
ATT
q_BN
1
att
ATT
quat
B
1
Matrix Multiply
b
2

Magnetometer Model – Simulink Block



Multiplicative Quaternion Extended Kalman Filter- Simulink Block

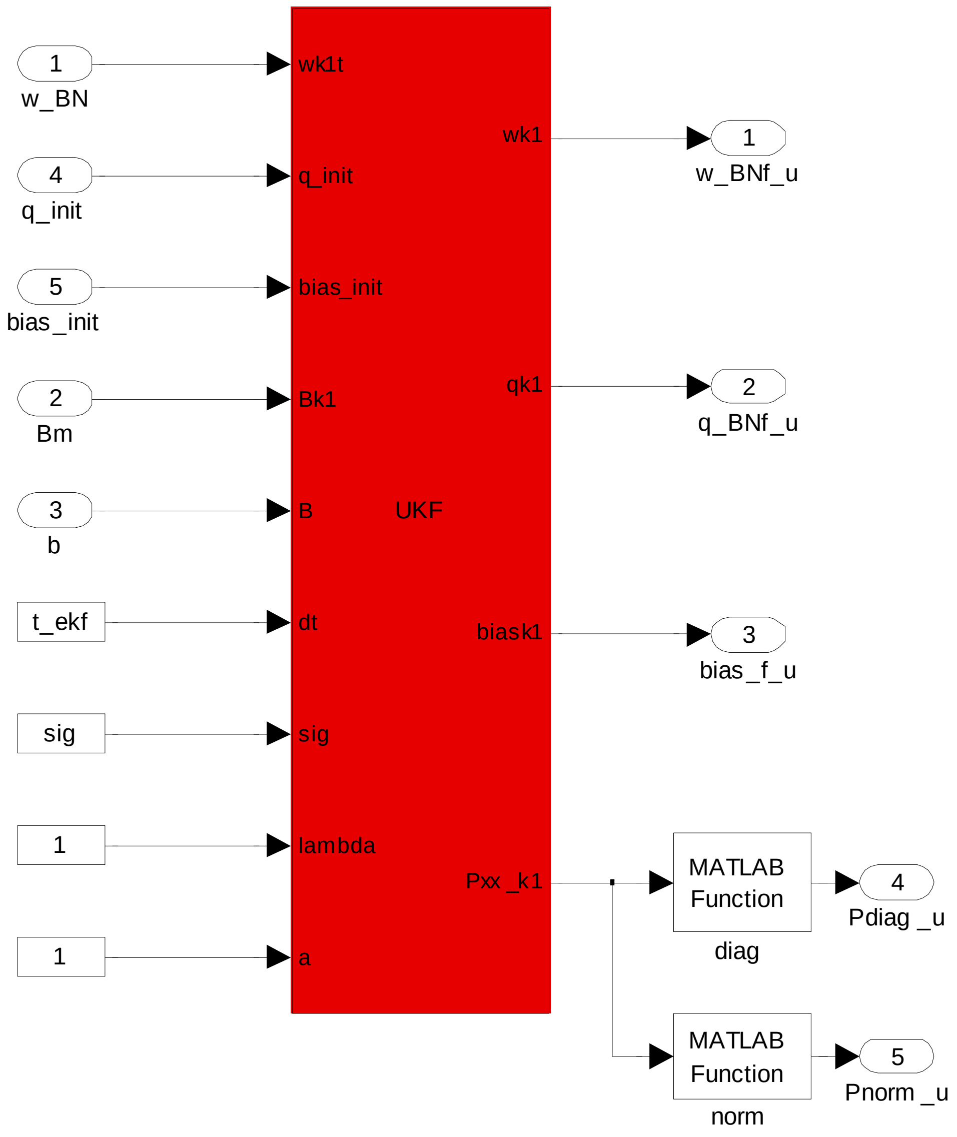


MEKF – Embedded Matlab Code

```
function [wk1,qk1,biask1,Pk1] = EKF(wk1t,q_init,bias_init,Bk1,B,dt,sig)

sig_v = sig(1);
sig_u = sig(2);
sig_mag = sig(5);

persistent qk biask wk Pk;
% Initialize States and Measurement
if isempty(qk)
    qk=q_init;
    biask = bias_init;
    wk = wk1t;
    Pk=[ (0.8)^2*eye(3) zeros(3); zeros(3) (3*pi/180)^2*eye(3)];

    wk1=wk;
    qk1=qk;
    biask1=biask;
    Pk1=Pk;
    return;
end

%% Propagation
biask1 = biask;


Skew_w = SKEW(wk);
Mag_w  = norm(wk);

psik = (sin(1/2*Mag_w*dt)/Mag_w)*wk;
Omega = [cos(1/2*Mag_w*dt)*eye(3)-SKEW(psik) psik;
        -psik'                 cos(1/2*Mag_w*dt) ];
qk1 = Omega*qk;

Phi_11 = eye(3)-Skew_w*sin(Mag_w*dt)/Mag_w + Skew_w^2*(1-
cos(Mag_w*dt))/Mag_w^2;        % 7.59b
Phi_12 = Skew_w*(1-cos(Mag_w*dt))/Mag_w^2 - eye(3)*dt -...
% 7.59c
    Skew_w^2*(Mag_w*dt-sin(Mag_w*dt))/Mag_w^3;
Phi_21 = zeros(3);
% 7.59d
Phi_22 = eye(3);
% 7.59e

Phi = [Phi_11 Phi_12; Phi_21 Phi_22];
% 7.59a

Gk = [-eye(3) zeros(3); zeros(3) eye(3)];
Qk = [ (sig_v^2*dt+1/3*sig_u^2*dt^3)*eye(3) -(1/2*sig_u^2*dt^2)*eye(3)
;
```

```
      -(1/2*sig_u^2*dt^2)*eye(3)             (sig_u^2*dt)*eye(3)
];

Pk1 = Phi*Pk*Phi'+Gk*Qk*Gk';

%% Update Loop  -------------------------------------------------------
Att = ATT(qk1);
delX = zeros(6,1);
%   Update for Magnetometer Measurement -------------------------------
H = [SKEW(Att*B) zeros(3,3)];

R = sig_mag^2*eye(3);

% Gain
K = (Pk1*H')/(H*Pk1*H' + R);

% Update
Pk1 = (eye(6) - K*H)*Pk1;

res = Bk1 - Att*B;
delX = delX + K*(res-H*delX);

qk1 = qk1+1/2*XI(qk1)*delX(1:3,:);
qk1 = qnormalize(qk1'*qk1,qk1);

biask1 = biask1 + delX(4:6,:);

wk1 = wk1t - biask1;
%   Save previous values
qk = qk1;
biask = biask1;
wk = wk1;
Pk = Pk1;

return
%---------------------------------------------------------------------

%% Normalizing routine for quaternions
function qk1 = qnormalize(qnorm,qk1)
while (qnorm) > 1
    if qnorm < 1 + 1e-9
        qk1 = ((3 + qnorm)/(1 + 3*qnorm))*qk1;
        %   rescale quaternion to (err^3)/32
    else
        qk1 = qk1/sqrt(qnorm);
        %   renormalize quaternion
    end
    qnorm = qk1'*qk1;
end
return
%---------------------------------------------------------------------
```

Unscented Kalman Filter – USQUE- Simulink Block



UKF- USQUE– Embedded Matlab Code

```
function [wk1,qk1,biask1,Pxx_k1] =
UKF(wk1t,q_init,bias_init,Bk1,B,dt,sig,lambda,a)

%
======================================================================
==
%                              Initialization
%
======================================================================
==



% Variance of Sensors
sig_v   = sig(1);
sig_u   = sig(2);
sig_mag = sig(5);
f = 2*(a+1);                                % Ref [3] pg 6


persistent qk biask Pxx_k
% Initialize States and Measurement
if isempty(qk)
    qk=q_init;
    biask = bias_init;
    disp(q_init);
    disp(bias_init);
    Pxx_k=[(1)^2*eye(3) zeros(3); zeros(3) (3*pi/180)^2*eye(3)];
end

%
======================================================================
==
%                        Calculation of Sigma Points
%
======================================================================
==

Qbar_k = dt/2*[(sig_v^2-1/6*sig_u^2*dt^2)*eye(3)      zeros(3)    ;
% Ref [3] 42
                        zeros(3)                  sig_u^2*eye(3) ];


%  Sigma points equations
x_k = [[0 0 0]'; biask];
Dx=size(x_k,1);
Dy=size(Bk1,1);
NSig=2*Dx+1;
sig_x=chol((Dx+lambda)*(Pxx_k+Qbar_k))';                        % Ref
[3]  5a
```

```
chi_sig_k=x_k*ones(1,NSig)+[zeros(Dx,1) sig_x -sig_x];        % Ref [3]
5a
chi_sig_k1=zeros(6,NSig);                                   % Ref [3] 32
y_sig_k1=zeros(3,NSig);
q_k1=zeros(4,1);


for i=1:NSig

    del_q_k=delp2delq(chi_sig_k(1:3,i),a,f);
    q_sig_k = quaterr(del_q_k, qk);

    %
====================================================================
    %       Propagate Forward the Quaternion (still in the loop!)
    %
====================================================================

    w_sig_k = wk1t - chi_sig_k(4:6,i);                          % Ref [3]  35
    Mag_w  = norm(w_sig_k);
    psik = (sin(1/2*Mag_w*dt)/Mag_w)*w_sig_k;
    zk = cos(1/2*Mag_w*dt)*eye(3)-SKEW(psik);

    Omega = [  zk,           psik;                              % Ref [3]  29
             -psik',  cos(1/2*Mag_w*dt) ];

    q_sig_k1 = Omega*q_sig_k;                                   % Ref [3]  34

    % =========================
    %   Saving the q(-)k+1(0)                                   % Ref [3]  36
    % =========================

    if i==1

        q_k1=q_sig_k1;
    end


    del_q_k1 = quaterr(q_sig_k1, [-q_k1(1:3,1);q_k1(4,1)]);
    chi_sig_k1(1:3,i) = f*del_q_k1(1:3,1)/(a+del_q_k1(4,1));  % Ref [3]
37b
    chi_sig_k1(4:6,i) = chi_sig_k(4:6,i);
    y_sig_k1(:,i) = ATT(q_sig_k1)*B;

    %
====================================================================
    % Note: The bias does not change so chi_sig_k1(4:6,i) stays the
same
    %
====================================================================

end
```

```
    %
=====================================================================
    %                    Following  USQUE Method Ref [3] pg 6
    %
=====================================================================



%  Calculating Weights
R = sig_mag^2*eye(3);

W=ones(NSig,1)/(2*(Dx+lambda));
W(1,1)=lambda/(Dx+lambda);

%  Mean Point Calculations
x_k1p=chi_sig_k1*W;
y_k1p=y_sig_k1*W;


%
=======================================================================
==
%                      Covariance and Gain Calculations
%
=======================================================================
==


% Error Covariance Calculation
Pxx_k1p=Qbar_k;
Pyy_k1p=R;
Pxy_k1p=zeros(Dx,Dy);

for i=1:NSig
    xdif=chi_sig_k1(:,i)-x_k1p;
    ydif=y_sig_k1(:,i)-y_k1p;
    Pxx_k1p=Pxx_k1p+xdif*xdif'*W(i,1);
    Pyy_k1p=Pyy_k1p+ydif*ydif'*W(i,1);
    Pxy_k1p=Pxy_k1p+xdif*ydif'*W(i,1);
end

% Gain and Update
K = Pxy_k1p/Pyy_k1p;  % Gain

Pxx_k1 = Pxx_k1p-K*Pxy_k1p';  % Error Covariance Update
x_k1 = x_k1p+K*(Bk1-y_k1p);   % State Update

% Calculation of Updated Quaternion!
del_q_k1=delp2delq(x_k1(1:3,:),a,f);
qk1 = quaterr(del_q_k1,q_k1);
qk1 = qnormalize(qk1);
```

```
biask1 = x_k1(4:6,1);
wk1 = wk1t-biask1;

%
======================================================================
==
%                       Setup for Next Update
%
======================================================================
==


qk=qk1;
biask=biask1;
Pxx_k = Pxx_k1;

return
%
%
% %-------------------------------------------------------------------
------
%% Normalizing routine for quaternions
function qk1 = qnormalize(qk1)
qnorm=qk1'*qk1;
while (qnorm) > 1
    if qnorm < 1 + 1e-9
        qk1 = ((3 + qnorm)/(1 + 3*qnorm))*qk1;
        %   rescale quaternion to (err^3)/32
    else
        qk1 = qk1/sqrt(qnorm);
        %   renormalize quaternion
    end
     qnorm = qk1'*qk1;
end
return
%---------------------------------------------------------------------
----
```

ADS_MainScript.m

```
%%  Spacecraft Attitude Determination Script
%   Note that this code runs both the EKF and the UKF


%   Created by Orlando X. Diaz
%   Advisor Dr. Marcelo Romano
%   Co-Advisor Dr. Hyun-wook Woo

%%  Format
    clear all
    close all
    clc

    global CONST
    R2D = 180/pi;
    D2R = pi/180;
%%  Set Simulation Conditions

    InitialEuler = [0,0,0];%deg
    ReferenceEuler = [0 0 0];%deg


%***  Toggle switches turn the labeled functions on (1) or off (0).
***
    Tgg_toggle       = 1;%
    Taero_toggle     = 1;%
    Tsolar_toggle    = 1;%
    timeOn           = 1;
    taOn             = 0;
    cboOn            = 0;
    qbnOn            = 1;
    qbnmOn           = 1;
    rOn              = 0;
    hOn              = 0;
    e321On           = 1;
    wbnOn            = 1;
    tcOn             = 0;
    hsOn             = 0;
    wbnfOn           = 1;
    biasOn           = 1;
    biasfOn          = 1;
    pdOn             = 1;
    pnOn             = 1;
    qbnfOn           = 1;
    wbnmOn           = 1;
    werrOn           = 1;
    berrOn           = 1;
    qerrOn           = 1;
```

```
%%  Set Constants
    CONST.mu        = 398.6004418e12;%m^3/s^2
    CONST.mu_moon   = 4.902802953597e12;%m^3/s^2
    CONST.mu_sun    = 1.327122E20;%m^3/s^2
    CONST.Re        = 6.378137E6;%m                      earth radius
    CONST.Rs        = 1.4959787e11;%m                    solar radius
    CONST.J2        = 1.08262668355E-3;%                 J2 term
    CONST.J3        = -2.53265648533E-6;%                J3 term
    CONST.J4        = -1.61962159137E-6;%                J4 term
    CONST.SolarPress= 4.51e-6;%N/m^2                     solar wind pressure
    CONST.SOLARSEC  = 806.81112382429;%TU
    CONST.w_earth   = -[0;0;.0000729211585530];%r/s earth rotation
    CONST.Cd        = 2.5;%                              Coefficient of Drag
    CONST.Cr        = .6;%                               Coefficient of
Reflect
    CONST.OmegaDot  = 1.991e-7;%rad/s                    ascending node
advance for sun-synch

%%  Set Orbital Elements
    %Kep elements meters and radians (a,e,i,W,w,n)

    h_p             = 500e3;%m                           altitude at perigee
    h_a             = 500e3;%m                           altitude at apogee



    RAAN = 0;%rad                                        Right Ascention
    w = 0;%rad                                           argument of perigee
    TAo = 0;%rad                                         true anomaly
    Rp  = CONST.Re+h_p;%m                                radius of perigee
    Ra  = CONST.Re+h_a;%m                                radius of apogee
    e   = (Ra-Rp)/(Ra+Rp);%(m/m)                         eccentricity
    a   = (Ra+Rp)/2;%m                                   semi-major axis
    ho  = sqrt(a*CONST.mu*(1-e^2));%mÿ2/s                initial angular
momentum
    P   = 2*pi*(a^3/CONST.mu)^.5;%sec                    Orbit Period
    i_sunsynch = acosd((CONST.OmegaDot*(1-e^2)^2*a^(7/2))...
        /(-3/2*sqrt(CONST.mu)*CONST.J2*CONST.Re^2));%eqn 4.47 from
Curtis
    i   = i_sunsynch*D2R;%deg (rad)                      orbit inclination

    [Ro,Vo] = sv_from_coe(CONST.mu,[ho e RAAN i w TAo]);%    initial
orbital state vector

%%  Set ICs

w_BNo = [0;2*pi/P;0];%rad   initial body rates
w_ON =  [0;2*pi/P;0];%rad

rand('seed',2);
randn('seed',2);
seedarw=1;
seedrrw=2;
seedst=3;
seedmag=4;
```

```
%   Sensor parameters
%   Gyro
GYRO_Bias = (3*randn(3,1))*pi/180;  % + 3 deg/sec
N_ARW = (0.029)*pi/180;
K_RRW = (0.0002)*pi/180;
ARW = N_ARW^2;                      % angular white noise Variance
RRW = K_RRW^2/3;                    % bias variance
Gg = eye(3).*(-0.01+0.02*rand(3)) +...
    (ones(3,3)-eye(3)).*(-0.0006+0.0012*rand(3)); %percent

%   Magnetometer
sigMag = 1.25e-7;
Gm = eye(3).*(-0.02+0.04*rand(3)) +...
    (ones(3,3)-eye(3)).*(-0.0028+0.0056*rand(3)); %percent

%   Sun Sensor
S1 = [0 45 0]'*pi/180;
S2 = [45 0 0]'*pi/180;

SS_n1 = [1 0 0];
SS_n2 = [1 0 0];
FOV = 0.7;
sigSS = 0.1;
J = Bessel(sigSS/2,FOV).*pi/180;

%   Star Tracker
sigST = 70 /3 /60 /60*pi/180;       %arcsec to rad (3sig)

%   Kalman Filter
dt = 0.05;                          %sec (20 Hz) model speed
t_ekf = dt;                         %sec (100 Hz) ekf speed
sig(1) = sqrt(ARW);                 %rad/Hz^(1/2), ARW
sig(2) = sqrt(RRW);                 %rad/sec^(3/2), RRW
sig(3) = sigST;                     %rad, Star Tracker Error
sig(4) = sigSS*pi/180;              %rad, Sun Sensor Error
sig(5) = sigMag;                    %tesla, magnetometer error

ReferenceOmega = w_ON;

[qBOo] = Euler_to_Quaternion(InitialEuler);
[ReferenceQuaternion] = Euler_to_Quaternion(ReferenceEuler);

qBNo = qBOo;

%%  Run Simulation
[Spacecraft]= SCproperties;

J_Matrix = Spacecraft.MOI;

[density_table] = GetDensity;

RunTime = 4000;%sec
```

```
tic
sim('ADS_SpacecraftSim',RunTime);
Total_Model_time = toc
factor = RunTime/Total_Model_time

DisturbanceTorques.Tgg = Tgg;
DisturbanceTorques.Taero = Taero;
DisturbanceTorques.Tsolar = Tsolar;

SensorMeasurements.ST = q_BNm;
SensorMeasurements.Gyro = w_BNm;
SensorMeasurements.bias = bias;
SensorMeasurements.SS1 = ss1;
SensorMeasurements.SS2 = ss2;
SensorMeasurements.Mag = Bm;

FilterEst.Q = q_BNf;
FilterEst.Gyro = w_BNf;
FilterEst.bias = bias_f;
```

PlotUKFError.m

```
bias_e_u1=squeeze(bias_e_u1);
[m,n]=size(bias_e_u1);
if m<n
    bias_e_u1=bias_e_u1';
end

for i=1:3
    figure(1)
    subplot(3,1,i)
    plot(SimTime1,p_BNe_u1(:,i))
    hold on
    plot(SimTime1,3*sqrt(Pdiag_u1(:,i)),'-.r')
    plot(SimTime1,-3*sqrt(Pdiag_u1(:,i)),'-.r')
    grid on
    xlim([0 4000])
    ylim([-.2 .2])
    xlabel('Time (Sec)')
    label=['\deltap' num2str(i)];
    ylabel(label)

end
subplot(3,1,2)
ylim([-.05 .05])
subplot(3,1,1)
title('Generalized Rodriquez Parameter Error for UKF')

for i=4:6
    figure(2)
    subplot(3,1,i-3)
    plot(SimTime1,bias_e_u1(:,i-3));
    hold on
    plot(SimTime1,3*sqrt(Pdiag_u1(:,i)),'-.r');
    plot(SimTime1,-3*sqrt(Pdiag_u1(:,i)),'-.r');
    grid on
    xlim([0 4000])
    ylim([-5E-4 5E-4])
    xlabel('Time (Sec)')
    label=['\delta \beta ' num2str(i-3)];
    ylabel(label)
end
subplot(3,1,1)
title('Bias Error for UKF')


bias_e_e1=squeeze(bias_e_e1);
[m,n]=size(bias_e_e1);
if m<n
```

```
    bias_e_e1=bias_e_e1';
end
for i=1:3
    figure(3)
    subplot(3,1,i)
    plot(SimTime1,q_BNe_e1(:,i))
    hold on
    plot(SimTime1,3/2*sqrt(Pdiag1(:,i)),'-.r');
    plot(SimTime1,-3/2*sqrt(Pdiag1(:,i)),'-.r');
    grid on
    xlim([0 4000])
    ylim([-.15 .15])
    ylim([-.2 .2])
    xlabel('Time (Sec)')
    label=['\deltaq' num2str(i)];
    ylabel(label)

end
subplot(3,1,1)
title('Quaternion Error for EKF')
subplot(3,1,2)
ylim([-.02 .02])

for i=4:6
    figure(4)
    subplot(3,1,i-3)
    plot(SimTime1,bias_e_e1(:,i-3))
    hold on
    plot(SimTime1,3*sqrt(Pdiag1(:,i)),'-.r');
    plot(SimTime1,-3*sqrt(Pdiag1(:,i)),'-.r');
    grid on
    xlim([0 4000])
    ylim([-5E-4 5E-4])
    xlabel('Time (Sec)')
    label=['\delta \beta ' num2str(i-3)];
    ylabel(label)

end
subplot(3,1,1)
title('Bias Error for EKF')

clear norm_p_u norm_p_e norm_bias_u norm_bias_e
norm_p_u=zeros(m,1);
norm_p_e=zeros(m,1);
norm_bias_u=zeros(m,1);
norm_bias_e=zeros(m,1);
for i=1:m;
norm_p_u(i,1)=norm(p_BNe_u1(i,:));
norm_p_e(i,1)=norm(2*p_BNe_e1(i,:));
norm_bias_u(i,1)=norm(bias_e_u1(i,:));
norm_bias_e(i,1)=norm(bias_e_e1(i,:));
end

figure(5)
```

```
semilogy(SimTime1,norm_p_u)
hold on
semilogy(SimTime1,norm_p_e,'-.r')
grid on
title('Normalized EKF and UKF Attitude Errors')
xlabel('Time (Sec)')
ylabel('Attitude Errors')
legend('Normalized UKF Generalized Rodriguez Parameter Errors',
'Normalized EKF Generalized Rodriguez Parameter Errors')
xlim([0 4000])

figure(6)
semilogy(SimTime1,norm_bias_u)
hold on
semilogy(SimTime1,norm_bias_e,'-.r')
grid on
title('Normalized EKF and UKF Bias Errors')
xlabel('Time (Sec)')
ylabel('Normalized \beta Errors')
legend('Normalized UKF Bias Errors', 'Normalized EKF Bias Errors')
ylim([0 .01])
xlim([0 4000])
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] S. Flagg, R. White, and R. Ewart, "Operationally Responsive Space Specifications and Standards: An Approach to Converging with the Community," in AIAA Space 2007 Conference & Exposition, Long Beach, 2007, pp. 1–19.

[2] E. J. Lefferts, F. L. Markley, and M. D. Shuster, "Kalman Filtering for Spacecraft Attitude Estimation," *Journal of Guidance, Control and Dynamics*, vol. 15, no. 5, 1982, pp. 417–429. AIAA-82-0070.

[3] S. J. Julier and J. K. Uhlmann, "Unscented Filtering and Nonlinear Estimation," *Proceedings of the IEEE*, vol. 92, no. 3, 2004, pp. 401–422.

[4] J. L. Crassidis and F. L. Markley, "Unscented Filtering for Spacecraft Attitude Estimation," Journal of Guidance, Control and Dynamics, vol. 26, no. 4, 2003, pp. 536–542.

[5] J. Tuthill, "Design and Simulation of a Nano-Satellite Attitude Determination System," Naval Postgraduate School, Monterey, CA, Master's Thesis December 2009.

[6] L. Stras, D. D. Kekez, et al. "The Design and Operation of The Canadian Advanced Nanospace eXperiment (CanX-1)," July 2004. [Online]. Available: http://www.utias-sfl.net/nanosatellites/CanX1. [Accessed September 2010].

[7] "AISsat-l Monitoring the Shipping Traffic from Space." *NordicSpace Online Journal*, April 2004. [Online]. Available: http://www.nordicspace.net/PDF/NSA238.pdf. [Accessed September 2010].

[8] M. D. Shuster and S. D. Oh, "Three-Axis Attitude Determination from Vector Observations," *Journal of Guidance, Control and Dynamics*, vol. 4, no. 1, 1981, pp. 70–77. AIAA-81-4003.

[9] J. L. Crassidis and J. L. Junkins, "*Optimal Estimation of Dynamic Systems*," Goong Chen and Thomas J Bridges, Eds. New York, USA: Chapman & Hall/CRC, 2004.

[10] M. S. Grewel and A. A. Andrews, "*Kalman Filtering: Theory and Practice Using MATLAB®"*. 3rd ed. John Wiley and Sons, 2008.

[11] F. Orderud, "Comparison of Kalman Filter Estimation Approaches for State Space Models with Nonlinear Measurements.," *Sem Sælands vei*, vol. 7491, pp. 7–9, March 2006.

[12] S. J. Julier, J. K. Uhlmann, and H. F. Durrant-Whyte. “A New Approach for Filtering Nonlinear Systems.” In *Proceedings of the 1995 American Control Conference*, 1628–1632, 1995.

[13] M. C. VanDyke, J. L. Schwartz, and C. D. Hall. “Unscented Kalman Filtering for Spacecraft Attitude State and Parameter Estimation.” In *Proceedings of the 2004 Space Flight Mechanics Meeting*, AAS-04-115, 2004.

[14] A. J. Blocker, “Tinyscope: The Feasibility of a Tactically Useful, Three-Axis Stabilized, Earth-Imaging Nano-Satellite,” Naval Postgraduate School, Monterey, CA, Master's Thesis December 2008.

[15] B. Wei, “*Space Vehicle Dynamics and Control*.” AIAA Education Series, 1998.

[16] L. F. Markley, “Multiplicative vs. Additive Filtering for Spacecraft Attitude Determination.” July 2004. [Online]. Available: http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov. [Accessed June 3, 2010].

[17] S. J. Julier and J. K. Uhlmann, “A New Extension of the Kalman Filter to Nonlinear Systems,” In *Proceedings of the SPIE AeroSense International Symposium an Aerospace/Defense Sensing. Simulations and Controls*, (Orlando, Florida), April 20–25, 1997.

[18] H. Curtis, *“Orbital Mechanics for Engineering Students, 2nd ed.”* Elsevier’s Butterworth–Heinemann 2005.

[19] D. Vallado, “*Fundamentals of Astrodynamics and Applications, 2nd ed.”* El Segundo: Microcosm Press, 2004.

[20] R.C. Olsen, *“Introduction to the Space Environment.”* Monterey: Naval Postgraduate School, 2005.

[21] W. Larson and J. R. Wertz, “*Space Mission Analysis and Design, 3rd ed.*” El Segundo: Microcosm Press, 2005.

[22] E. Kraft, “A Quaternion-based Unscented Kalman Filter for Orientation Tracking,” Proceedings of the 6th International Conference on Information Fusion, Cairns, Australia, 2003, pp. 47–54.

[23] C. C. Litton, “Tinyscope: The Feasibility of a Tactically Useful Earth-Imaging Nanosatellite and a Preliminary Design of the Optical Payload,” Naval Postgraduate School, Monterey, CA, Master's Thesis January 2009.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California