ON-ORBIT MASS PROPERTY IDENTIFICATION FOR A COUPLED SPACECRAFT SYSTEM

A Project
Presented to
The Faculty of the Department of Aerospace Engineering
San José State University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

By
Gregory Kessing
May 2021

The Designated Project Advisor(s) Approves the Project Titled

On-Orbit Mass Property Identification for a Coupled Spacecraft System

by

Gregory Kessing

APPROVED FOR THE DEPARTMENT OF AEROSPACE ENGINEERING

SAN JOSÉ STATE UNIVERSITY

May 2021

Prof. Long Lu SJSU Faculty Advisor

ABSTRACT

On-Orbit Mass Property Identification for a Coupled Spacecraft System

by Gregory Kessing

With ever increasing forays into space come an increasing number of encounters with objects, man-made or otherwise, that are unknown in their nature. Whether the unknown object is manipulated or maneuvered away from is ultimately up to the mission designer. If manipulating the unknown object is the desired objective, then a detailed analysis of the properties of the acquired object is required. This project provides the analysis to do so using an identification maneuver to vary the properties of the system, after rendezvous with the unknown object. The proposed algorithm is designed to use the maneuver to estimate the mass properties of the coupled system and isolate the properties of the acquired object from that data. Throughout the duration of the identification maneuver, a recursive least squares algorithm is used to identify the inertia matrix of the system. After implementation of the proposed sequence, the estimation of the system inertia matrix provides the desired output to well within an allowable error tolerance. The robustness of the system is then analyzed for its ability to handle noise generated by the sensors used to gather the required data. A recursive least squares filter is used to control for the undetermined levels of sensor and process noise. State estimation is proposed to handle unavailable or unmeasurable states of the system. Future work and research topics are proposed to further improve the system developed in this report.

ACKNOWLEDGEMENTS

I would like to first thank Professor Lu for his guidance throughout the course of this project and the introduction to guidance, navigation, and controls topics through the AE 168 and AE 245 courses. These courses were instrumental in my determination of project topic and my area of interest in aerospace engineering. I would also like to thank my family and friends for their support and understanding in my pursuit of a graduate degree and change in career path.

TABLE OF CONTENTS

LIST OF FIGURES

NOMENCLATURE

$(*)_A$	Variable for known spacecraft
$(*)_B$	Variable for unknown spacecraft
$(*)_T$	Variable for total or combined spacecraft duo
τ	Torque
I	Inertia matrix (tensor)
F	Linear force
a	Linear acceleration
v	Linear velocity
α	Angular acceleration
ω	Rotational velocity
$\dot{*}$	Time derivative of a variable, $*$ as the placeholder
$\vec{r}_*$	Radius vector to specified object
m	Mass
J_{**}	Inertia value representing the inertia
H	Angular momentum
A	State matrix
B	Input matrix
C	Output or measurement matrix
D	Feed-forward matrix
K_P	Proportional Gain
K_I	Integral Gain
K_D	Derivative Gain
$\boldsymbol{x}$	State matrix
$\hat{\boldsymbol{x}}$	Estimated state matrix
$\boldsymbol{u}$	Input matrix
$\boldsymbol{w}$	Process noise matrix
$\boldsymbol{z}$	Measurement matrix
$\boldsymbol{v}$	Measurement noise matrix
$\boldsymbol{P}$	Covariance matrix
$\boldsymbol{K}$	Filter gain matrix
$\boldsymbol{R}$	Measurement covariance matrix

$\boldsymbol{Q}$	Process noise covariance matrix
$\boldsymbol{h}^{*}$	Proposed filter matrix
$\boldsymbol{\zeta}$	Error value
$\boldsymbol{S}$	Sum of the cost function
$\boldsymbol{\phi}$	Inverse of the covariance matrix

CHAPTER 1: INTRODUCTION

1.1 Motivation

As more advanced satellites are sent into orbit by rockets with ever increasing payload capacity the operations that humanity has decided to undertake in space have grown in scope drastically over the last couple of decades. There is now talk of a multitude of new activities in space which require novel solutions. This activity includes Earth orbit cleanup of debris, refueling of man-made satellites, and potential capture of asteroids with the intent of maneuvering them into Earth or Lunar orbit. The most pressing of these proposed activities is the cleanup of debris in Earth Orbit.

For the last 60 years, mankind has sent satellites into orbit but during this time, we have not been kind stewards of the space around Earth. We have created an estimated 34,000 pieces of debris larger than 10cm in diameter [1]. This has created a hazard when launching spacecraft into Earth orbit and into the solar system at large due to the increased chances of an impact with debris. When two objects collide at an orbital velocity of 7.8 km/s for low Earth orbit, they create further debris clouds which increase the chances of another impact. Due to this threat of orbital debris, NASA and the ESA have undertaken first steps towards beginning the cleanup process of the space around Earth. As such, a spacecraft whose goal is to de-orbit this debris may require knowledge about the mass properties of a single satellite which it aims to de-orbit.

The mass properties of a single satellite, usually the inertia tensor, has long been a topic of interest in the field of guidance, navigation, and control due to its requirement in calculating a future trajectory and the thrust cycle needed to achieve such a trajectory. For most satellites which are intended to remain in space, a baseline inertia tensor is calculated before a satellite leaves the Earth. During the lifespan an expected amount of fuel will be expended, changing the

inertia tensor along with the center of gravity and total mass. This change can be estimated by either calculating the change in mass properties through fuel expenditure and container location or using estimation algorithms utilizing known inputs and outputs of the spacecraft. For space-based debris, these methods of determining an inertia tensor were either never taken or are impossible to complete due to their unknown status once in space.

1.2 Literature Review

This section will cover literature attempting to solve the problem of de-orbiting debris and/or spacecraft remains and relating to the topic of this paper with the goal of discussing the following critical points:

- Mass property identification
- Inertia tensor
- Mass identification and center of mass
- Separation of the individual mass properties from the mass properties of the coupled system
- Sensor noise elimination schemes

During a debris de-orbiting maneuver, one of the possible techniques is a rigid capture where the object is manually de-orbited by another spacecraft. In this situation, a rigid capture system will require knowledge of the mass properties of the debris if the control system is to properly determine the correct thruster/reaction wheel outputs required to guide the coupled system along the ideal trajectory [2]. This operation greatly reduces the propellant used during de-orbiting and allows for an increased number of objects to be taken out of orbit per kilogram of propellant used

[3]. The mass property determination can be introduced during a de-tumbling maneuver required to fully control and reorient the debris for a final de-orbit maneuver [4]. Another technique for orbit degradation, is the utilization of a tethered spacecraft to impart a force slowing the horizontal velocity of the debris. This will occur until the debris is on a trajectory to fall into the atmosphere. In this case, the mass properties can be used to calculate the required total impulse for maneuver to succeed [5].

A common method for mass property determination is to use Newton's Second Law for rotational systems.

$$\tau = I\alpha \tag{1.2.1}$$

Using this law allows for accurate determination of mass properties through utilizing a known input force and measuring the angular acceleration after the force is applied. Expanding this equation into its constituent components displays the true unknowns that are required to solve for the inertia matrix.

$$\alpha = \frac{I^{-1}(r - r_{cm}) \times T}{m^{-1}T} \tag{1.2.2}$$

The unknowns in Eq. (1.2.2) are I the inertia matrix, r_{cm} the position of the center of mass, and m^{-1} the mass inverse. The thrust (T), position of the thruster (r), and angular acceleration (α) are all known or can be found through sensor data [6]. Using multiple test configurations during the analysis window the number of unknowns can be matched by the number of situations tested. This will allow for the unknowns to be solved linearly so long as the weight ratio between the objects is no more than 20:1. Another criteria for this method is that external forces and moments affecting the spacecraft are minimal otherwise more testing will be required. During testing the

configurations tested should be as different as possible to allow the control system the widest possible dataset [7].

Another potential for introducing angular acceleration is from forces derived from the magnetic field from the Earth by utilizing torque rods. Magnetic systems are mostly used to end the tumbling caused by detaching from a primary spacecraft but can be adapted to be used for satellite acquisition. During this acquisition the pair of satellites will need to be stabilized during which time the known force output from the magnetic system can be used to identify the mass properties of the coupled spacecraft [8].

Using thruster jets to produce a controlled torque will be both expensive in terms of fuel expenditure, and in terms of wear and tear on the spacecraft itself. Fuel is a limited commodity onboard a spacecraft and refueling such a craft as is discussed in this paper is both inordinately complex and cost in-effective. An alternative to thrusters is reaction wheels which produce torque about the center of mass with no fuel expenditure through the use of electric motors powered by solar panels and batteries onboard the craft. Algorithms developed for the use of reaction wheels have been able to reproduce the inertial properties of the ISS within a margin of error of 0.01%. The mass and center of mass location were also able to be estimated with similar success over a slightly longer timespan [9].

Data from a maneuver by the Saturn orbiter Cassini has been used many times to test mass property estimation algorithms. The data regarding spin rates of reaction wheels, and rotation rates of the spacecraft allow for accurate estimations to be made. The estimation is made easier due to an axis by axis rotation sequence that was performed by the craft during the time period that the data was collected. A least squares estimation was able to produce accurate results simply and effectively accounting for mass lost due to propulsion [10].

1.2.1 Optimization

Accuracy of mass property estimation can be further increased through the utilization of multiple methods. A proposed combination are conservation of rotational kinetic energy, Eq. (1.2.3), and angular momentum, Eq. (1.2.4), unilaterally resulted in greatly improved estimation and lower standard deviations [11].

$$\frac{1}{2}\omega^T I\omega|_t = T|_0 + \int_0^t \omega^T \sum M \, d\tau \tag{1.2.3}$$

$${}^N Q^k\left(I\omega + \sum I_{w,i}\Omega_i a_i\right)|_k = \quad {}^N Q^j\left(I\omega + \sum I_{w,i}\Omega_i a_i\right)|_j \tag{1.2.4}$$

This technique in conjunction with backstepping can produce desirable control results without destabilizing a complicated system. This is achieved by utilizing ω as a low-level control while a control law using the input *u* to stabilize Eq. (1.2.5) by forcing $\omega \rightarrow \omega_d(\rho)$ Eq. (1.2.6) is instituted [12].

$$\dot{\omega} = J^{-1}S(\omega)J\omega + J^{-1}u \tag{1.2.5}$$

$$\dot{\rho} = H(\rho)\omega \tag{1.2.6}$$

A control-Lyapunov function is derived using backstepping for the system of Eq. (1.2.5) and Eq. (1.2.6). This results in an optimal control system when the cost function is considered [12].

Several observer-based inertia identification control systems have been purposed which have proven to be accurate assuming no measurement errors [13,14]. An observer-based control system including measurement error estimation and external disturbance is put forth resulting in a higher accuracy algorithm for real world situations. In addition, estimation bounds are put into place based on known limits of sensing equipment [15].

1.2.2 Disturbance and Noise Filtering

Noise reduction methods in a reaction wheel-based estimation scheme are a necessity and can be the difference between mission success and failure. The Butterworth filter is a method that maintains a frequency response that is near flat which results in the transfer function in Eq. (1.2.7) where G_0 represents the static gain, ω_c the cut-off frequency, and s_k the pole calculated by the cut-off frequency.

$$H(s) = \frac{G_0}{\prod_{k=1}^{n} \frac{s-s_k}{\omega_c}} \tag{1.2.7}$$

The zero-phase shift filter (Y_0) utilizes a forward low pass filter (Y_F) and a rearward low pass filter (Y_R) to filter the signal without impacting shifting the phase of the signal itself thereby maintaining integrity. Shown below in Eq. (1.2.8).

$$Y_0 = \frac{Y_F + Y_R}{2} \tag{1.2.8}$$

A continuous-discrete extended Kalman filter (EKF) is determined to be superior to the Butterworth filter and the zero-phase shift filter when accuracy over time is prioritized. After determining angular rates using EKF, the rates are applied to a batch least square inertia estimation technique. The EKF produced results closest to the true Inertia tensor and RMS value of noise [16]. Subspace parameter identification can be used in conjunction with a Kalman filter to increase accuracy further, but due to increased computation costs is rarely used onboard a spacecraft. The use of QR decomposition and singular value decomposition are the main reasons for this. Instead, if a Hankel matrix is constructed and an RQ decomposition is performed, much of the computation cost can be eliminated. Estimates of mass properties can then be found using a recursive estimation of the observation matrix [17].

Dynamic spacecraft systems are difficult to estimate due to parameters changing with time or elements of the spacecraft being flexible. A method of estimating the parameters has been defined using recursive predictor-based subspace identification (RPBSID) [18]. The method is based around an estimation of Markov Parameters, or the impulse response of the system, being found using a Kalman filter and a white noise sequence being introduced as seen in Eq. (1.2.9) and Eq. (1.2.10).

$$x_{k+1} = A_k x_k + B_k u_k + K_k e_k \tag{1.2.9}$$

$$y_k = C_k x_k + e_k \tag{1.2.10}$$

Where K_k is the Kalman gain matrix, e_k is the white noise sequence, and k is the given timestep. Estimation of the state vector (x_k) is achieved using a least squares recursive form in order to reduce the computational cost of the algorithm. Then state space parameters can be identified through the recursive computation of the defining matrices [18].

For missions in low Earth orbit, external torque from the gravity-gradient and disturbance from the atmosphere must be considered when completing any estimation of properties of the spacecraft. An extended Kalman filter (EKF) scheme estimator can be used to eliminate external torque influences from the resultant mass properties. This use of an EKF is more accurate than using change in angular momentum alone [19].

Another method of increasing accuracy of the least squares estimation technique is introduced as tracking differentiation (TD) with the use of an extended Kalman filter. Tracking differentiation is defined in Eq. (1.2.11) where v is the input signal, x_a is the filter value of v, x_b is the derivative of x_a, r is the tracking velocity of tracking differentiation, and h is the step size of the simulation. The TD algorithm is given by Eq. (1.2.11) below.

$$\begin{Bmatrix} fh = \text{fhan}(x_a(k) - v(k), x_b(k), r, h_0) \\ x_a(k+1) = x_a(k) + hx_b(k) \\ x_b(k+1) = x_b(k) + hfh \end{Bmatrix} \quad (1.2.11)$$

In the algorithm v is the input signal, x_a and x_b are the filter value and derivative of the filter value respectively. r represents the tracking of the algorithm and h is the step size. $fhan$ represents a switching algorithm designed to zero in on the true or ideal value.

Tracking differentiation is able to filter out noise better than traditional differentiation but cannot fully eliminate the sensor noise so it is applied again in a TD-TD setup then a recursive least squares estimation algorithm is applied as shown in Figure 1 [20].

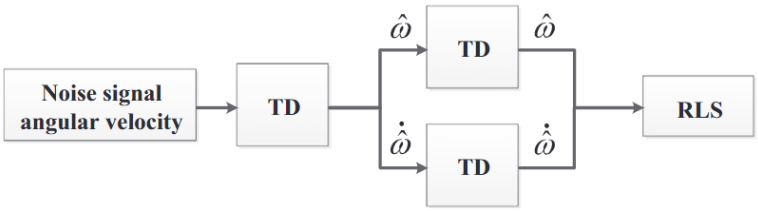

Figure 1.1: TD-TD-RLS

1.3 Report Outline

A proof of concept algorithm has been proposed and implemented to determine the mass properties of an unknown object in a combined known/unknown (a/b) duo spacecraft system. This is accomplished through the use of a recursive least squares algorithm to firstly identify the total system inertia matrix from the known state of the system. A retraction maneuver is then completed to change the total inertia matrix and produce a different, yet mathematically related, movement profile. Using the original and the new profile the estimates of the remaining unknown parameters are solved for.

The Least Squares (LS) algorithm that is used for the inertia matrix estimation. Eq. (1.3.1) describes the equation of rotational motion for a rigid body in freefall. The '*T*' subscript in the following equations denotes the total of the property that it is attached to (ie. I_T is the total inertia matrix for the combined spacecraft system).

$$\dot{\boldsymbol{\omega}}_T = -{I_T}^{-1}\boldsymbol{\omega}_T \times \boldsymbol{I_T}\boldsymbol{\omega}_T + \boldsymbol{I_T}^{-1}\boldsymbol{u} \tag{1.3.1}$$

$$\boldsymbol{y} = \boldsymbol{\omega}_T$$

The cost function in Eq. (1.3.2) is minimized resulting in an estimate of the state matrix shown in Eq. (1.3.3).

$$\boldsymbol{I} = \tfrac{1}{2}(z - \boldsymbol{H}\hat{\boldsymbol{x}})^T(z - \boldsymbol{H}\hat{\boldsymbol{x}}) \tag{1.3.2}$$

$$\hat{\boldsymbol{x}} = (\boldsymbol{H}^T\boldsymbol{H})^{-1}\boldsymbol{H}^T\boldsymbol{z} \tag{1.3.3}$$

Once the estimation is determined to be sufficient, noise is imparted to simulate a realistic sensor that has a specific noise floor, beneath which any signal will be indistinguishable. To combat this a recursive least squares filter is used to isolate the true data from the noise. Validation in the form of a Monte Carlo simulation is utilized to determine the efficacy of the method. The validation step shows that recursive least squares does not filter the noise to a sufficient level and a more advanced filter is proposed in the form of a Kalman Filter.

The (Extended) Kalman Filter is implemented as shown in Figure 1.2, where the input '*u*' is created by a set of thrusters or reaction wheels designed to impart a torque on the spacecraft system. The system model represents the inertia matrix and input matrix of the combined system which will determine the angular velocity, ω, as a result of the input torque. In the testing scenario, the full system model will be known but noise will be imparted and only some of the

states measured to give a realistic simulation scenario to the filter. The resultant, '$\hat{\boldsymbol{x}}$', is the state matrix reconstructed which defines the filtered properties of the system, whether or not they were directly measured. The data flow is shown below:

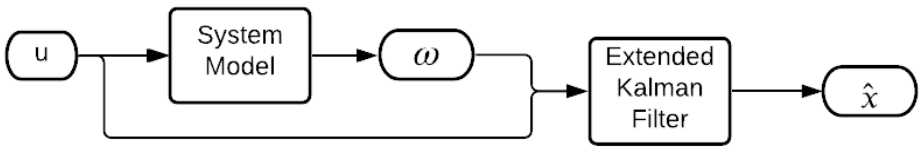

Figure 1.2: Extended Kalman Filter

The EKF model equations are given in Eq. (1.3.4), where the functions *f* and *h* predict the state and the measurement of the state respectively. Simulated system noise and sensor noise are added to the system through ***w(t)*** and ***v(t)*** for each predictor through a gaussian white noise process.

$$\dot{\boldsymbol{x}} = f\big(\boldsymbol{x}(t), \boldsymbol{u}(t) + \boldsymbol{w}(t)\big) \tag{1.3.4a}$$

$$\boldsymbol{z}(t) = h\big(\boldsymbol{x}(t) + \boldsymbol{v}(t)\big) \tag{1.3.4b}$$

Due to the nature of a continuous-time EKF, the predictions and updates are coupled and will be handled simultaneously by the set of Eqs. (1.3.5).

$$\dot{\hat{\boldsymbol{x}}} = f(\hat{\boldsymbol{x}}, \boldsymbol{u}) + \boldsymbol{K} * \{\boldsymbol{z} - h(\hat{\boldsymbol{x}})\}$$

$$\dot{\boldsymbol{P}} = \boldsymbol{FP} + \boldsymbol{PF}^T - \boldsymbol{KHP} + \boldsymbol{Q}$$

$$\boldsymbol{K} = \boldsymbol{PH}^T\boldsymbol{R}^{-1} \tag{1.3.5}$$

$$\boldsymbol{F} = \frac{\partial f}{\partial x}\Big|_{\hat{\boldsymbol{x}},\boldsymbol{u}}$$

$$\boldsymbol{h} = \frac{\partial h}{\partial x}\Big|_{\hat{\boldsymbol{x}}}$$

The combined flowchart of the EKF and RLS estimation algorithm is shown in Figure 1.3.

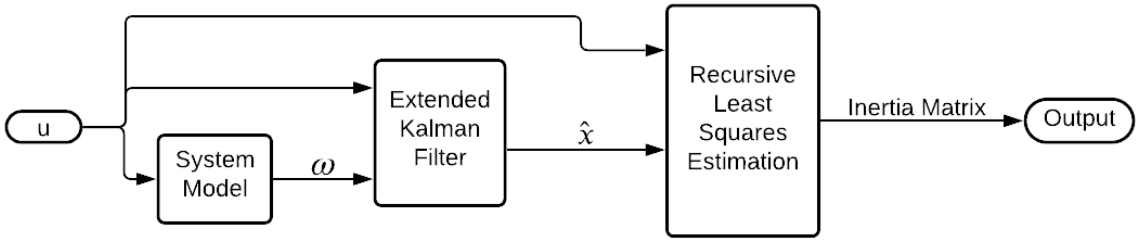

Figure 1.3: Extended Kalman Filter and Recursive Least Squares Algorithm

The resulting inertia matrix is used in conjunction with the spacecraft a inertia matrix to solve for the inertia matrix for spacecraft b using Eq. (1.3.6).

$$\bar{\boldsymbol{I}}_T = \sum_{i=1}^{n} \boldsymbol{I}_i + m_i \boldsymbol{r}_{cm_i}{}^2$$

$$\bar{\boldsymbol{I}}_T = \boldsymbol{I}_a + m_a \boldsymbol{r}_{cm_a}{}^2 + \boldsymbol{I}_b + m_b \boldsymbol{r}_{cm_b}{}^2$$

$$\bar{\boldsymbol{I}}_T - \boldsymbol{I}_a - m_a \boldsymbol{r}_{cm_a}{}^2 - m_b \boldsymbol{r}_{cm_b}{}^2 = \boldsymbol{I}_b \tag{1.3.6}$$

The algorithm will be simulated using MATLAB to verify the robustness and performance where a series of differing virtual payloads will be estimated, and the algorithm verified by comparing the output to the given mass properties. In proving robustness, noise from both the system and sensors will be simulated via a Gaussian distribution. The noise will be based off the hobbyist available BNO055 sensor with an accelerometer sensitivity rating of 0.077 m/s^2 and a gyroscope accuracy of 0.0086 rad/s [21]. The choice of this sensor comes down to availability, ease of use, and the high noise floor when compared to industry sensors. The increase in noise will allow for a high confidence factor for the algorithm when complete.

CHAPTER 2: SPACECRAFT DYNAMICS

2.1 Frame of Reference

When handling moving objects there are two common frames of reference. These are referred to in general as the fixed frame and the inertial frame. When talking about spacecraft or aircraft inside of the gravity well of Earth these corelate to the Earth Centered Inertial frame, or ECI, and the body frame. The ECI frame is, as the name suggests, centered on the Earth with the $\vec{z}$ axis going through the north pole, the $\vec{x}$ axis going through the intersection of the lines of zero degrees latitude and zero degrees longitude, and the $\vec{y}$ axis at a right angle to both of the other axes. This can be seen in Figure 2.1 below.

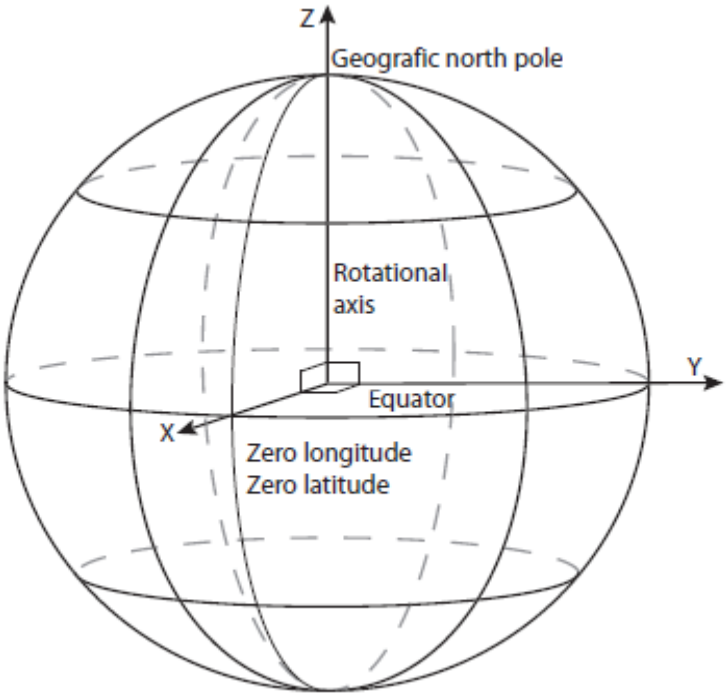

Figure 2.1: Earth Centered Inertia Frame [22]

The body frame is connected to the body of the object that is being tracked during the motion being described. This can be a satellite orbiting the Earth, or a plane flying through the atmosphere.

The body frame moves with the object and the object is seen as static at $t = 0$ from the point of view of the body frame. Figure 2.2 displays the body frame with respect to the ECI frame.

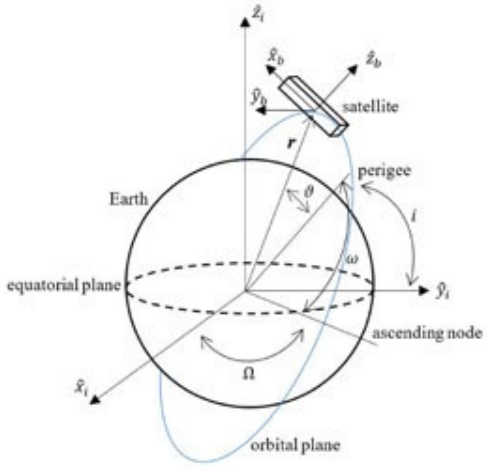

Figure 2.2: ECI Frame and Body Frame [23]

2.2 Euler Angles

The change in angle between the two primary frame of references can be described through a series of rotations. Three body-axis rotations are an easy way to convert from one reference frame to another. The first rotation is about any single axis, with the second rotation being about one of the two axes not used in the first rotation, and the third rotation can be about any of the two axes not used in the second rotation. This means that there are twelve total ways to complete the rotations required to move between the two reference frames. The three angles that can complete this conversion are called Euler angles and are shown in Figure 2.3 as $\alpha, \beta, and\ \gamma$.

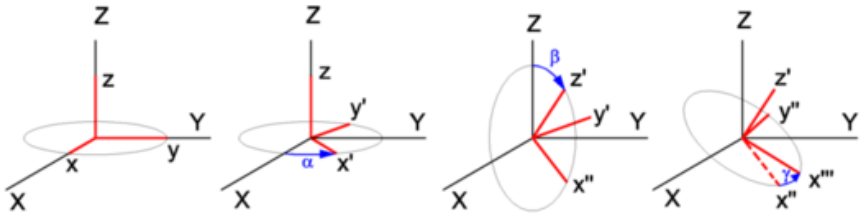

Figure 2.3: Euler Rotations [24]

The Euler angle rotations move through the axes with a common naming scheme for the axes shown below:

$$\mathrm{A} \rightarrow A' \rightarrow A'' \rightarrow \mathrm{B}$$

Each of the rotations is based off a relation between one axis and another which is often called a Direct Cosine Matrix, or DCM. Each element of the DCM is based off the function below:

$$\mathrm{C_{ij}} = \mathrm{b_i} \cdot a_j \tag{2.2.1}$$

This is a general term and is not specific to rotating from the beginning **A** axis to the end **B** axis. Writing out the matrix results in the following:

$$C^{b/a} = \begin{bmatrix} \mathrm{b_1} \cdot a_1 & \mathrm{b_1} \cdot a_2 & \mathrm{b_1} \cdot a_3 \\ \mathrm{b_2} \cdot a_1 & \mathrm{b_2} \cdot a_2 & \mathrm{b_2} \cdot a_3 \\ \mathrm{b_3} \cdot a_1 & \mathrm{b_3} \cdot a_2 & \mathrm{b_3} \cdot a_3 \end{bmatrix} \tag{2.2.2}$$

The notation of "b/a" is specific as it denotes which way the matrix is converting, and in this specific case, $b \leftarrow a$. The transformation equation can be written as the following:

$$b = C^{b/a} a \tag{2.2.3}$$

When using the DCM to convert from $B \leftarrow A$ the DCM of each individual rotation can be multiplied together to achieve the DCM $\boldsymbol{C}^{B/A}$ as shown in the following equation.

$$\boldsymbol{C}^{B/A} = C_1(\theta_1) C_2(\theta_2) C_3(\theta_3) \tag{2.2.4}$$

Using this method of rotation about Euler angles means that any of the states (for example, velocity or position) can be easily converted from one frame to another allowing for kinematics to be observed in one frame and controlled in another.

2.3 Equations of Motion

The Equations of Motion, EoM, for a system describe its characteristics as a function of time. Whether the system is perceived to be affected by a gravity well or not, the EoM can be derived from Newton's 2nd law of EoM. Once defined, the equations allow for mathematical relations to be derived between variables describing different states within a system.

Newton's 2nd law states that the acceleration of an object is directly proportional to the mass of said object, or in equation form:

$$F = m * \frac{\delta v}{\delta t} = \frac{\delta m v}{\delta t} \tag{2.3.1}$$

Simplifying again results in force being equal to the change in momentum with time.

$$F = \frac{\delta H}{\delta t} \tag{2.3.2}$$

When talking about spacecraft, it can be assumed that little or minimal force is applied to the system from rigid connections to the earth or another object by which the body axes are defined. In this circumstance rotational motion becomes the primary influencer on the routine motion of the spacecraft. Newton's 2nd law of motions becomes Newton's 2nd law of rotational motion.

$$\tau = J * \alpha \tag{2.3.3}$$

In Eq. (2.3.4), τ represents torque, the angular equivalent to force. J represents the rotational inertia of the system. Expanding the torque equation results in the following:

$$\begin{bmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \end{bmatrix} = \begin{bmatrix} J_{xx} & J_{xy} & J_{xz} \\ J_{yx} & J_{yy} & J_{yz} \\ J_{zx} & J_{zy} & J_{zz} \end{bmatrix} \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} \tag{2.3.4}$$

Multiplying the matrices through reveals a set of governing equations when dealing with angular acceleration in a system.

$$\alpha_x = J_{xx}\tau_x + J_{xy}\tau_y + J_{xz}\tau_z \tag{2.3.5a}$$

$$\alpha_y = J_{yx}\tau_x + J_{yy}\tau_y + J_{yz}\tau_z \tag{2.3.5b}$$

$$\alpha_z = J_{zx}\tau_x + J_{zy}\tau_y + J_{zz}\tau_z \tag{2.3.5c}$$

Applying steps similar to those in Eq. (2.3.1) to Eq. (2.3.3) results in deriving an equation for angular momentum from Eq. (2.3.4).

$$H = J\omega \tag{2.3.6}$$

Expanding the matrices results in:

$$\begin{bmatrix} H_x \\ \mathrm{H}_y \\ \mathrm{H}_z \end{bmatrix} = \begin{bmatrix} J_{xx} & J_{xy} & J_{xz} \\ J_{yx} & J_{yy} & J_{yz} \\ J_{zx} & J_{zy} & J_{zz} \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{2.3.7}$$

Multiplying out the matrices results in the following set of equations:

$$\begin{aligned} H_x &= J_{xx}\omega_x + J_{xy}\omega_y + J_{xz}\omega_z \\ H_y &= J_{yx}\omega_x + J_{yy}\omega_y + J_{yz}\omega_z \\ H_z &= J_{zx}\omega_x + J_{zy}\omega_y + J_{zz}\omega_z \end{aligned} \tag{2.3.8}$$

After manipulating the equations in Eq. (2.3.6), Euler's Equations of Rotational Motion show themselves as:

$$\begin{aligned} \dot{\omega}_1 &= \frac{(J_2 - J_3)}{J_1}\omega_2\omega_3 + \frac{M_1}{J_1} \\ \dot{\omega}_2 &= \frac{(J_3 - J_1)}{J_2}\omega_1\omega_3 + \frac{M_2}{J_2} \\ \dot{\omega}_3 &= \frac{(J_1 - J_2)}{J_3}\omega_1\omega_2 + \frac{M_3}{J_3} \end{aligned} \tag{2.3.9}$$

2.4 State Space Model

The State Space model is a linear, time-invariant model of a known system. The model holds within it the state vector, $\vec{x}$, the time derivative of the state vector, $\dot{\vec{x}}$, the output vector, $\vec{y}$, and the input vector, $\vec{u}$. The model dynamics matrices are made up of the state matrix, A, the control matrix, B, the output or measurement matrix, C, and the feed-forward matrix, D.

$$\dot{\vec{x}} = A\vec{x} + B\vec{u} \tag{2.4.1a}$$

$$\vec{y} = C\vec{x} + D\vec{u} \tag{2.4.1b}$$

This model is shown in Figure 2.4 as a block diagram below.

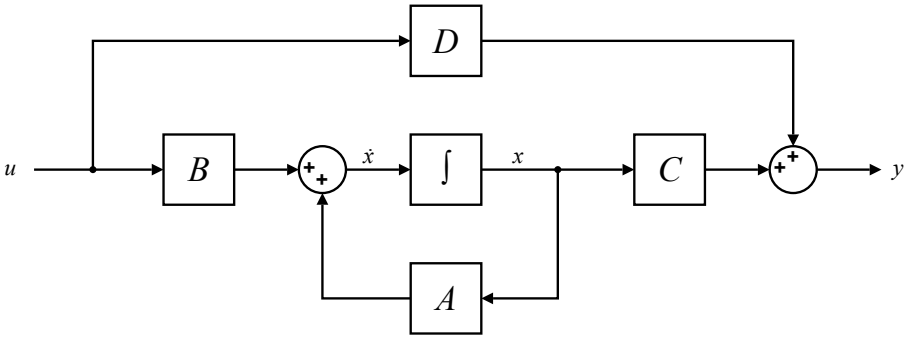

Figure 2.4: State Space Block Diagram

A state variable is defined as the base property of a system being modeled. The state variables are held in the $\vec{x}$ vector and typically consist of location or translation data about the system.

Converting the governing rotational dynamics of a spacecraft into a state space representation requires developing the equations of motion into the form seen in Eq. (2.3.10). In this form the base properties are represented by the angular velocity terms, ω. To create the A matrix the partial derivative of the functions needs to be taken with respect to the variables in the state matrix, $\vec{x}$.

$$\mathrm{A} = \begin{bmatrix} \frac{\delta f_1}{\delta \omega_1} & \frac{\delta f_1}{\delta \omega_2} & \frac{\delta f_1}{\delta \omega_3} \\ \frac{\delta f_2}{\delta \omega_1} & \frac{\delta f_2}{\delta \omega_2} & \frac{\delta f_2}{\delta \omega_3} \\ \frac{\delta f_3}{\delta \omega_1} & \frac{\delta f_3}{\delta \omega_2} & \frac{\delta f_3}{\delta \omega_3} \end{bmatrix} \tag{2.4.2}$$

The application of this rule on Eq. (2.3.10) results in the following A matrix.

$$A = \begin{bmatrix} 0 & \frac{(J_2 - J_3)}{J_1}\omega_3 & \frac{(J_2 - J_3)}{J_1}\omega_2 \\ \frac{(J_3 - J_1)}{J_2}\omega_3 & 0 & \frac{(J_3 - J_1)}{J_2}\omega_1 \\ \frac{(J_1 - J_2)}{J_3}\omega_2 & \frac{(J_1 - J_2)}{J_3}\omega_1 & 0 \end{bmatrix} \tag{2.4.3}$$

To create the B matrix the partial derivative of the functions needs to be taken with respect to the variables in the input matrix, $\vec{u}$.

$$B = \begin{bmatrix} \frac{\delta f_1}{\delta \mathrm{M}_1} & \frac{\delta f_1}{\delta \mathrm{M}_2} & \frac{\delta f_1}{\delta \mathrm{M}_3} \\ \frac{\delta f_2}{\delta M_1} & \frac{\delta f_2}{\delta \mathrm{M}_2} & \frac{\delta f_2}{\delta \mathrm{M}_3} \\ \frac{\delta f_3}{\delta \mathrm{M}_1} & \frac{\delta f_3}{\delta \mathrm{M}_2} & \frac{\delta f_3}{\delta \mathrm{M}_3} \end{bmatrix} \tag{2.4.4}$$

Implementing this results in the following B matrix:

$$B = \begin{bmatrix} \frac{1}{J_1} & 0 & 0 \\ 0 & \frac{1}{J_2} & 0 \\ 0 & 0 & \frac{1}{J_3} \end{bmatrix} \tag{2.4.5}$$

The C matrix will depend on what sensors are onboard the spacecraft and the controllability of the system. An example C matrix where motion about the x-axis is measured would result in:

$$\mathrm{C} = [1 \quad 0 \quad 0] \tag{2.4.6}$$

CHAPTER 3: PARAMETER ISOLATION

3.1 Overview

Using an internal measurement unit, or IMU, located at known location in spacecraft A, will allow for all required measurements to be taken. From the IMU, the angular acceleration will be isolated and compared the torque created by a set of reaction wheels inside the primary spacecraft. In Figure 3.1, the known spacecraft is denoted in blue and by the tag 'A'. The unknown debris or spacecraft is denoted in red and by the tag 'B'. The reaction wheels inside of Spacecraft A have their potential influence on the spacecraft show in green about the primary axes of the craft.

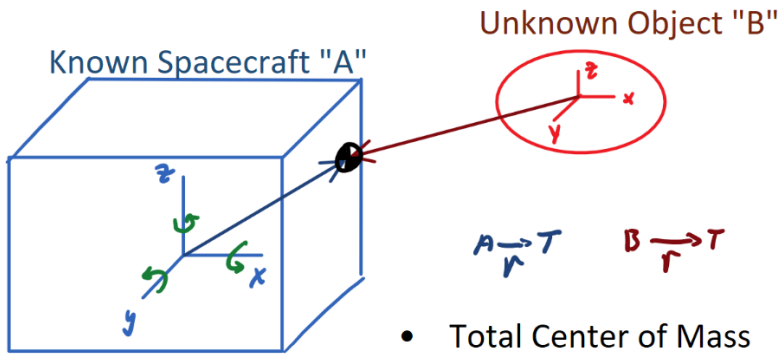

Figure 3.1: Diagram of Coupled Spacecraft

For testing and verification purposes the Cassini Inertia Tensor and mass properties [9] will be utilized to verify algorithms before using those algorithms to produce the final estimation of the system.

$$\boldsymbol{J_{Cassini}} = \begin{bmatrix} 8810.8 & -136.8 & 115.3 \\ -136.8 & 8157.3 & 156.4 \\ 115.3 & 156.4 & 4721.8 \end{bmatrix} [kg \cdot m^2] \tag{3.1.1}$$

3.2 Assumptions

For the following calculations in Sections 3.3 and 3.4 to be solved a few assumptions will be made about the known spacecraft. The primary assumption is that there are two main sets of sensors onboard, the accuracy requirement for both will be analyzed in Chapter 5 in depth. The first sensor that will be included is a rotation based sensor which can determine the rotation rate, ω, in radians per second as well as the acceleration of the rotation, α, in radians per second per second. The second sensor will be a linear acceleration sensor that will detect the linear acceleration in meters per second per second created due to the rotation of the space craft about the total center of mass with respect to the sensor itself, or the centripetal acceleration of the sensor.

The second assumption is that there is an arm that the spacecraft manipulates to attach itself to the unknown object. This will aide by allowing for the dynamics of the system to be changed in a known way and an analysis of the changes will provide further data for the calculation of the parameters to be estimated.

3.3 Inertia Equation

In this system, the designer knows the inertia matrix of Spacecraft A, $\boldsymbol{J}_A$, as well as the mass m_A. Using RLS the total rotational inertia matrix can be found about the primary axes of Spacecraft A. The inertial equation can be solved for using the sum of inertias and the parallel axis theorem. This derivation is shown below.

$$\boldsymbol{J}_T = \boldsymbol{J}_{A_T} + \boldsymbol{J}_{\mathrm{B_T}} \tag{3.3.1}$$

In Eq. (3.3.1), $\boldsymbol{J}_T$ represents the rotational inertia of the total system about the center of mass of the combined system. $\boldsymbol{J}_{A_T}$ and $\boldsymbol{J}_{B_T}$ represent the rotational inertia contribution of Spacecraft A and Spacecraft B respectively about the center of mass of the system.

Expanding Eq. (3.2.1) using the parallel axis theorem leads to the following:

$$\mathbf{J}_T = \mathbf{J}_\mathrm{A} + \mathrm{m}_\mathrm{A}\left(\vec{r}_{\mathrm{A_T}}\right)^2 + \mathbf{J}_\mathrm{B} + \mathrm{m}_\mathrm{B}\left(\vec{r}_{\mathrm{B_T}}\right)^2 \tag{3.3.2}$$

When using vector notation the scalar r^2 is invalid and it becomes the following:

$$\mathbf{J}_T = \mathbf{J}_\mathrm{A} + \mathrm{m}_\mathrm{A}\left(\vec{r}_{\mathrm{A_T}} \cdot \vec{r}_{\mathrm{A_T}}{}^T\right) + \mathbf{J}_\mathrm{B} + \mathrm{m}_\mathrm{B}\left(\vec{r}_{\mathrm{B_T}} \cdot \vec{r}_{\mathrm{B_T}}{}^T\right) \tag{3.3.3}$$

3.4 Solving for Unknowns

Using a linear acceleration sensor the perceived centripetal acceleration can be accessed from the sensor located in Spacecraft A. Using an IMU, the angular velocity can be obtained and together the radius from the center of mass of the total system to the center of mass of Spacecraft A, $\vec{r}_{T_A}$, can be solved for using Eq. (3.3.1).

$$\vec{a}_c = \boldsymbol{\omega}^2 \times \vec{r}_{T_{IMU}} \tag{3.4.1}$$

To convert between $\vec{r}_{T_{IMU}}$ and $\vec{r}_{T_A}$ subtraction of the $\vec{r}_{IMU}$ vector is required. The implementation of these are shown in Simulink in Figure 3.2 below.

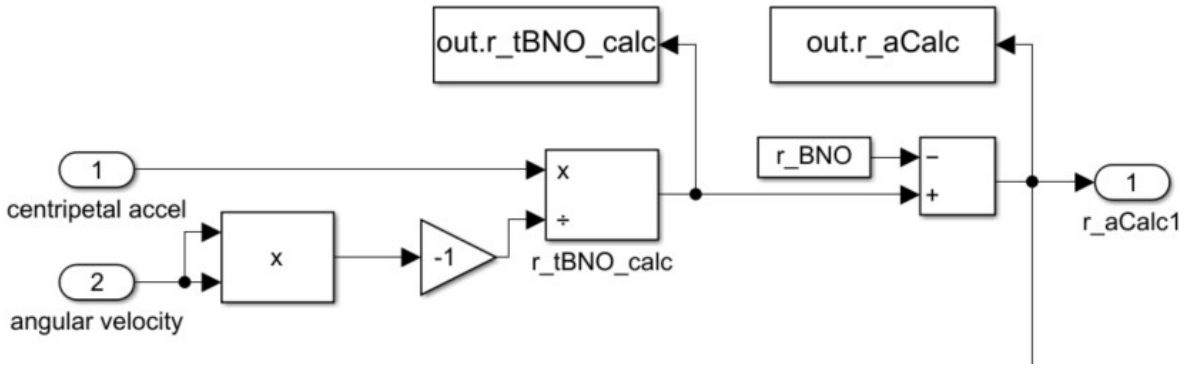

Figure 3.2 $\vec{r}_{T_A}$ Solution in Simulink

To calculate the vector of the center of mass of the total system to the center of mass of the unknown object, $\vec{r}_b$, it is possible to represent the vectors as a ratio of their masses.

$$\frac{m_b}{m_a} = \frac{r_b}{r_a} \tag{3.4.2}$$

Physically drawing the system together through the use of the retractable arm would allow for the dynamics of the system to change. This results in a r_{b_2} and r_{a_2} substitution using the change in the vectors d_{r_b} and d_{r_a} and the initial states. The ratio m_b:m_a can be ignored to result in the following equation.

$$\frac{r_{b_1}}{r_{a_1}} = \frac{r_{b_1} + d_{r_b}}{r_{a_2}} \tag{3.4.3}$$

Following the assumption laid out in Section 3.2 that the spacecraft knows the amount that it retracted in the operation d_{r_b} can be solved for as seen in Eq. (3.4.4). d_{r_a} is solved for by solving Eq. (3.4.1) post retraction.

$$d_{r_{total}} = d_{r_a} + d_{r_b} \tag{3.4.4}$$

The implementation for this in Simulink is shown below and the MATLAB function can be found in Apendix B.10.

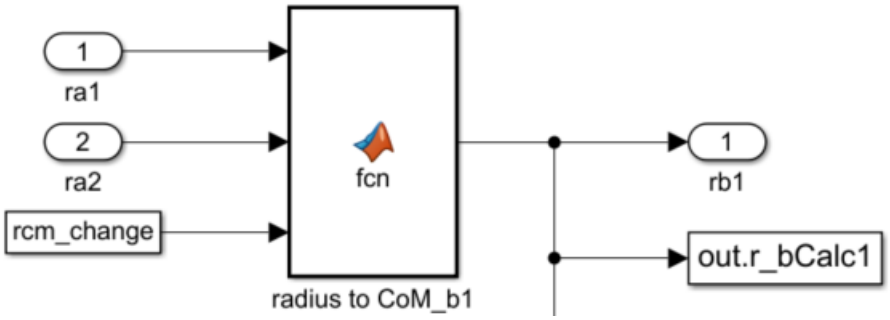

Figure 3.3 Simulink Implementation of the Solution for $\boldsymbol{r_b}$

The mass of the unknown object, m_b, can then be solved through rearranging Eq. (3.4.2) to form Eq. (3.4.5). See Figure 3.4 for the Simulink implementation and Apendix B.11 for the MATLAB function.

$$m_b = \frac{r_b * m_a}{r_a} \tag{3.4.5}$$

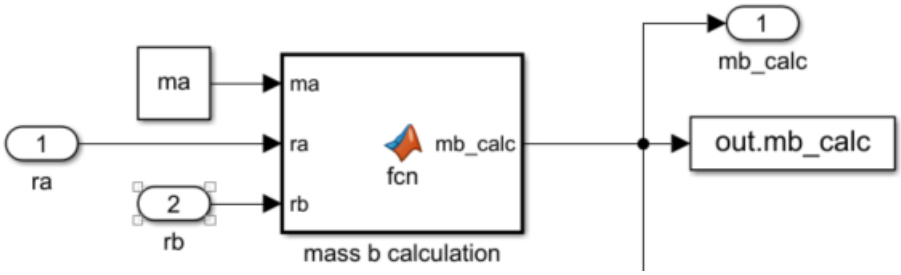

Figure 3.4 Simulink Mass of Unknown Object Calculation

To solve for the total inertia matrix the relation between torque applied and rotational acceleration can be used along with a Recursive Least Squares estimator to estimate the value of the total inertia matrix. This is discussed in Chapter 4 and implemented in Simulink as shown below in Figure 3.5.

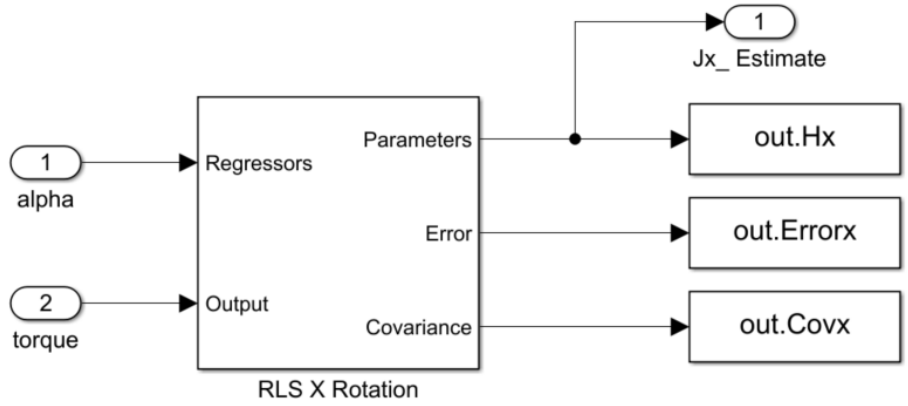

Figure 3.5 Recursive Least Square Implementation

This is then used in conjunction with the correction required to account for the products of inertia which is later discussed in Section 4.5, Eq. (4.5.1), and Figure 4.5. This results in the

following Simulink Implementation. The RLS estimation shown above in Figure 3.5 is contained within the three estimation blocks with the label “Fig. 3.5”.

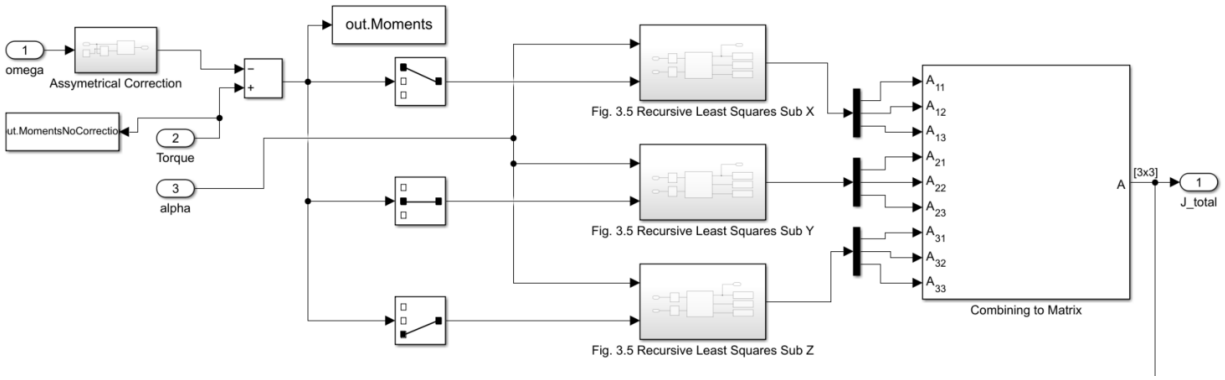

Figure 3.6: Total Inertia Matrix Estimation

The final calculation in the determination of the unknowns is to isolate the inertia matrix of the unknown spacecraft from Eq. (3.3.3). This is shown in the equation below.

$$\boldsymbol{J}_{\mathrm{B}} = \boldsymbol{J}_{\mathrm{T_A}} - \boldsymbol{J}_{\mathrm{A}} - m_a\left(\vec{r}_{T_a} \cdot \vec{r}_{T_a}{}^{T}\right) - m_b\left(\vec{r}_{T_b} \cdot \vec{r}_{T_b}{}^{T}\right) \tag{3.4.6}$$

It is important to note that if there is a future case where the radius vectors become matrices, [3x2] or similar, the outer product will need to be used instead of multiplying the transpose by the original vector. This is implemented using a MATLAB function block in the Simulink as can be seen in the image below and Appendix B.12

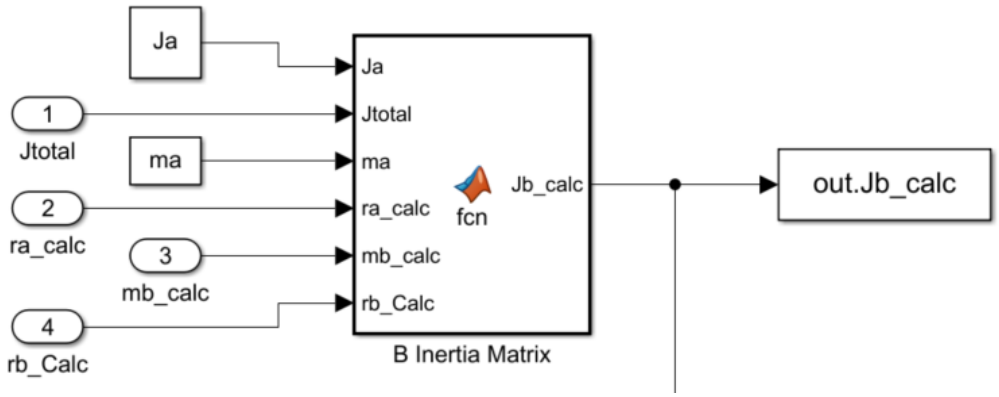

Figure 3.7: Unknown Inertia Matrix Simulink Implementation

The MATLAB file used to generate the data required for the calculations is available in Appendix B.13. The final Simulink file that was used to combine the algorithm described previously to obtain the desired unknown values can be found in Appendix B.16.

3.5 Results

In this simulation the Cassini spacecraft is assumed to be trying to identify the mass properties of an unknown object. The properties of which detailed below.

$$\boldsymbol{J_{Unknown}} = \begin{bmatrix} 2000 & 0 & 0 \\ 0 & 2000 & 0 \\ 0 & 0 & 2000 \end{bmatrix} [kg \cdot m^2] \tag{3.5.1}$$

$$m_{unknown} = 500\ [kg] \tag{3.5.2}$$

$$\vec{r}_{CoM} = \begin{bmatrix} 8 \\ 4 \\ \sqrt{20} \end{bmatrix} \tag{3.5.3}$$

The simulation is run resulting in an average error for $\boldsymbol{J_{Unknown}}$ of 10^{-8} as shown in the error plot in Figure 3.9. This value is calculated by using the average value of the estimated matrix and dividing that by 2000, the value given to the unknown object as its principal moments of inertia. The reasoning is that any adverse movements caused by the products of inertia during maneuvering will be competing with the effects of the principal moments of inertia.

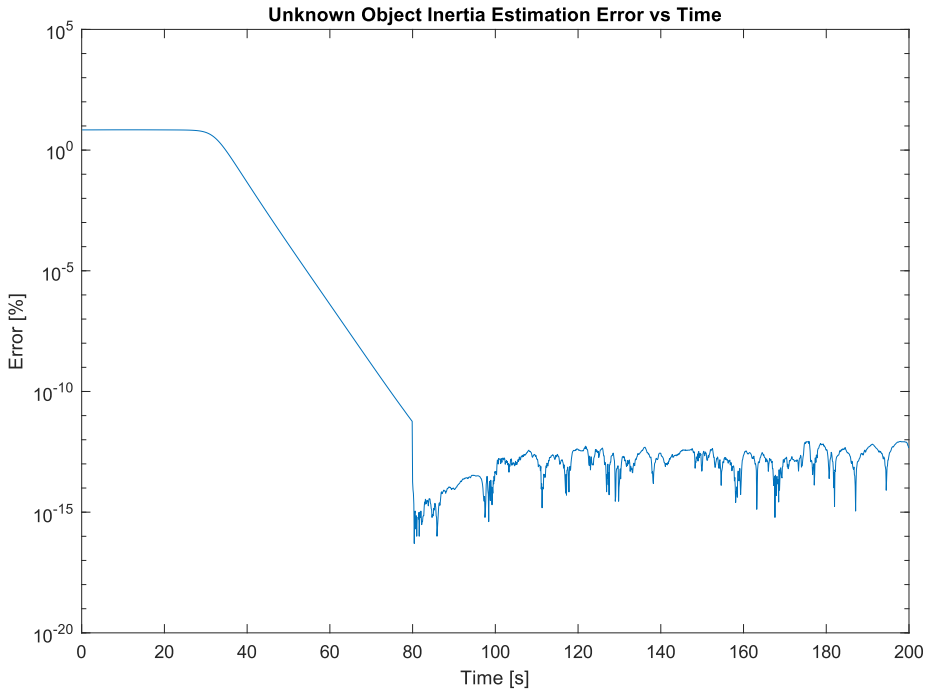

Figure 3.8: Unknown Object Inertia Estimation Error

When comparing Figure 3.9 to that of the errors resulting from the RLS estimation blocks, Figure 3.10, a very similar trend can be seen where at 80 seconds into the simulation there is a large dropoff and the estimation stabilizes afterwards. This time corresponds with the cutoff time for the reaction wheels simulated torque. This correlation was examined by reducing the cutoff time of the torque to 60 seconds to see if the estimation would be able to solve in less time due to having no external moment applied to the system. This resulted in the plots seen in Figure 3.11 and Figure 3.12. In both of these figures the cutoff can be visibly seen in the estimation error but neither shows an immediate settling of the residual error. This infers that the time that the simulation requires to solve is independent of the time that the torque is applied. Further time based questions will be proposed in Chapter 5.

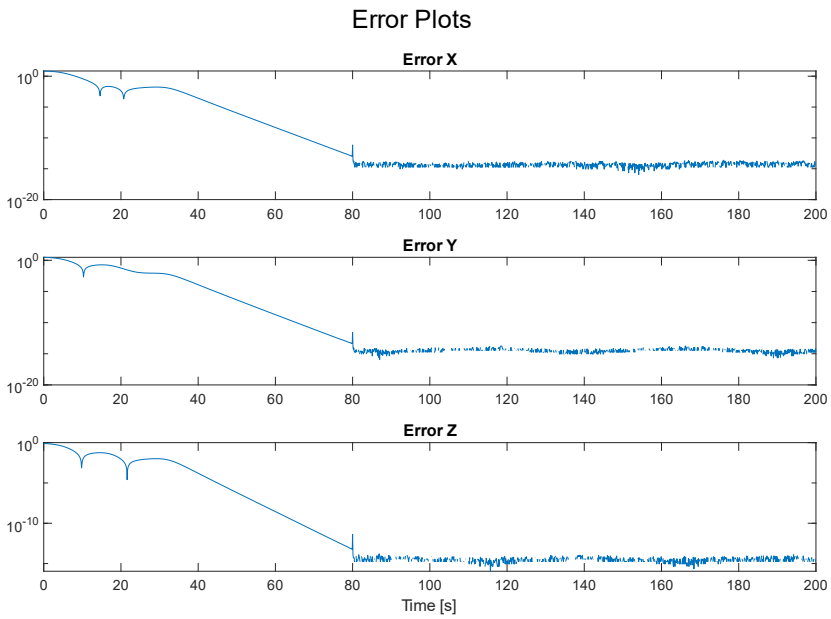

Figure 3.9: RLS Estimation Plots

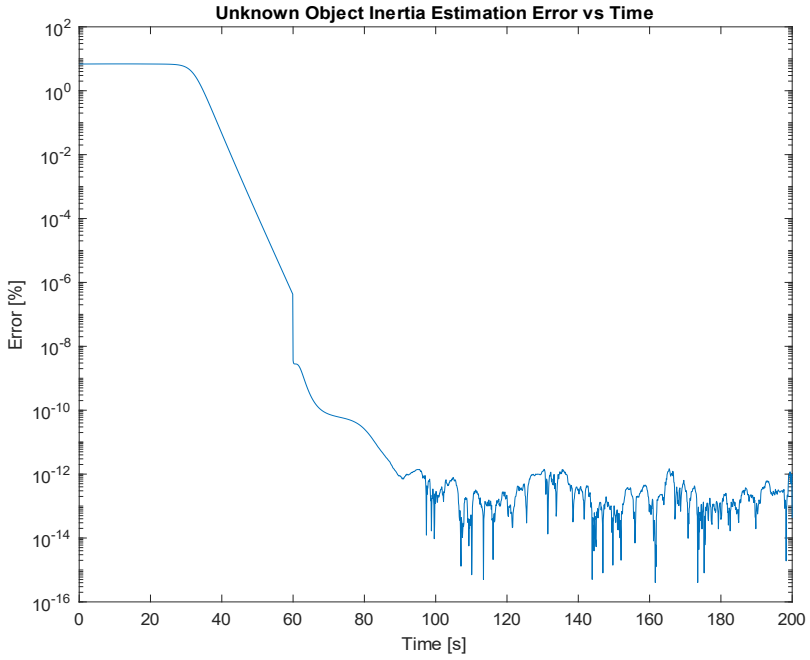

Figure 3.10: Unknown Inertia Estimate with Reaction Wheel Cutoff at 60s

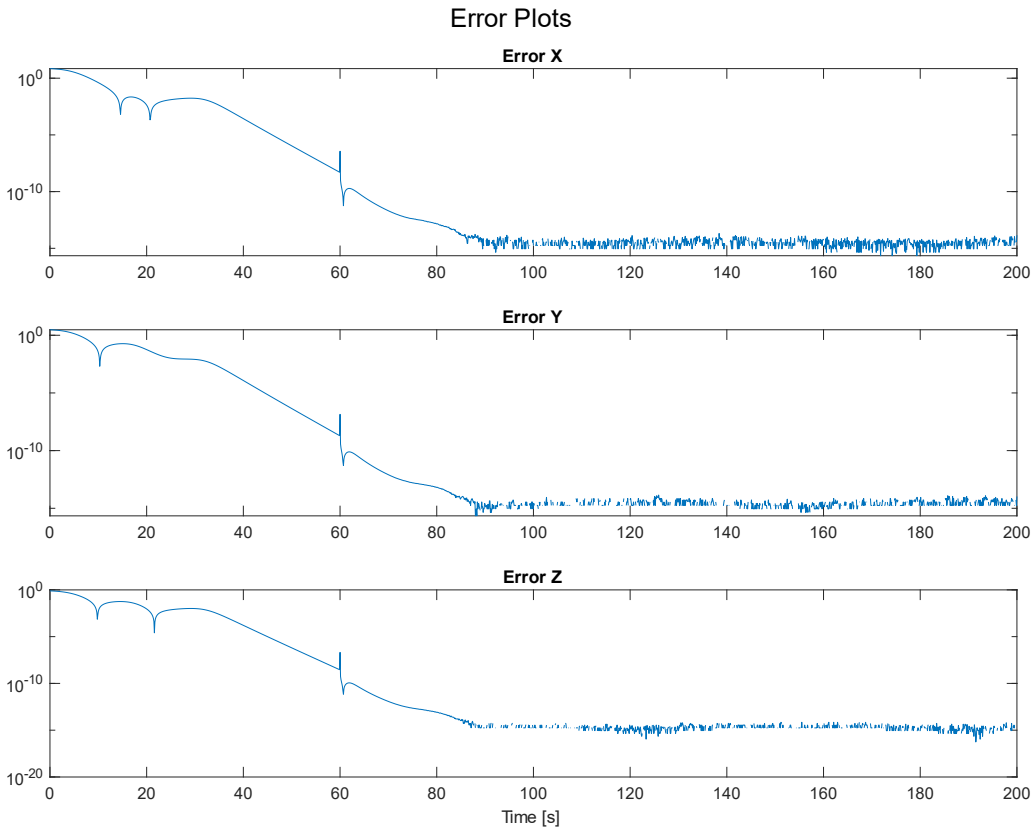

Figure 3.11: RLS Estimation Plots for 60s Cutoff

CHAPTER 4: ADAPTIVE ESTIMATION

4.1 Overview

An adaptive filter is an algorithm that takes a set of inputs and adjusts a guess for the coupling value between those inputs. This is best used when a system has a measurable input and output. The coupling value of the two measurements can generally be said to be solved for through parameter estimation schemes. For this project, parameter estimation will be used to estimate the inertia matrix from the moment input and angular acceleration output.

4.2 Least-Squares Estimation

Least-Squares Estimation (LSE) is a method of estimation based on the average deviation from the mean by a group of measured or given data. This estimation method can simply be compared to the estimation of a single point where multiple samples of noisy data are collected. To make an estimation of the true location of the data point the simplest method would be to take the average of all data points. This method is the optimal method as the result is the minimalization of the error values of each point squared giving the name least-squares estimate. Practically, this method works by solving for a value that minimizes the estimation error between each guess and the average of all the guesses, as seen in Eq. (4.2.1).

$$S = \sum_i^N (e_i)^2 \tag{4.2.1a}$$

$$S = \|e\|_2^2 \tag{4.2.1b}$$

The derivation of the Least-Squares Estimation algorithm can be found in Appendix A.3 page 73. The algorithm is summed up in Eq. (A.3.9), shown below, where $\boldsymbol{A_{n\times m}}$ represents the input matrix, $\boldsymbol{d_{n\times m}}$ represents the output matrix, and $\boldsymbol{h^*_{m\times 1}}$ represents the filter matrix. *n* represents the number of data points represented and *m* the number of variables being solved for.

$$\boldsymbol{h^* = (A^T A)^{-1} A^T d}$$

4.3 Recursive Least Squares

Recursive Least Squares, or RLS, is the natural discrete continuation on the theory behind Least-Squares Estimation. RLS is applied when the estimation algorithm is being used in a real time application instead of a batch process. The algorithm minimizes the weighted least squares cost function, assuming the errors are gaussian in nature, expanded from Eq. (4.2.1) results in the function below.

$$S(\beta) = \sum r_i(\beta)^2 \tag{4.3.1}$$

Summarizing the Recursive Least Squares estimation algorithm from Appendix A.4 page 74 leaves the following steps:

Solve for the gain vector in Eq. (A.4.12a)

$$\boldsymbol{K}_n = \frac{\boldsymbol{P}_{n-1}\boldsymbol{u}_n}{\boldsymbol{I}+\boldsymbol{u}_n^T\boldsymbol{P}_{n-1}\boldsymbol{u}_n}$$

Find the current error from the previous estimate in Eq. (A.4.19)

$$\boldsymbol{\zeta} = \boldsymbol{d}[n] - \boldsymbol{u}_n^T\boldsymbol{h}_{n-1}^*$$

Update the filter using Eq. (A.4.20)

$$\boldsymbol{h}_n^* = \boldsymbol{h}_{n-1}^* + \boldsymbol{K}_n\boldsymbol{\zeta}$$

Update the inverse matrix using Eq. (A.4.11)

$$\boldsymbol{P}_n = \boldsymbol{P}_{n-1} - \boldsymbol{K}_n\boldsymbol{u}_n^T\boldsymbol{P}_{n-1}$$

4.4 MATLAB Implementation

4.4.1 Test Data Creation

Implementation of the step by step process above first required the creation of data to test the code with. The code for creating the test data is shown in Appendix B.1 on page 78 under the MATLAB section "EOM Calculations". The Cassini Probe was used as this baseline for the proof of concept and its inertia matrix, as shown in Eq. (3.1.1), is used in the code. During the creation of test data, there will be two distinct timeframes representing different input conditions. Condition 1 from time $t = 1$ to $t = 150$ will use the input torque of 10 [Nm] about the x-axis.

$$\vec{\tau} = \begin{bmatrix} 10 \\ 0 \\ 0 \end{bmatrix} [Nm] \tag{4.4.1}$$

During the second condition, from $t = 150$ to $t = 200$, the input torque will be zero about all axes. This step input is being simulated to ensure that the functions developed can handle a change in input which adds another degree of difficulty to the problem.

$$\vec{\tau} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} [Nm] \tag{4.4.2}$$

The function in used to simulate the motion of the spacecraft is rotationalEOM.m, detailed in Appendix B.2 on page 80. This is based on the coupled rotational dynamics for a system where the products of inertia are non-zero. The relationship is detailed below.

$$\vec{\tau} = I\dot{\vec{\omega}} + \vec{\omega} \times (I\vec{\omega}) \tag{4.4.3}$$

This produced the data set seen below in Figure 4.1.

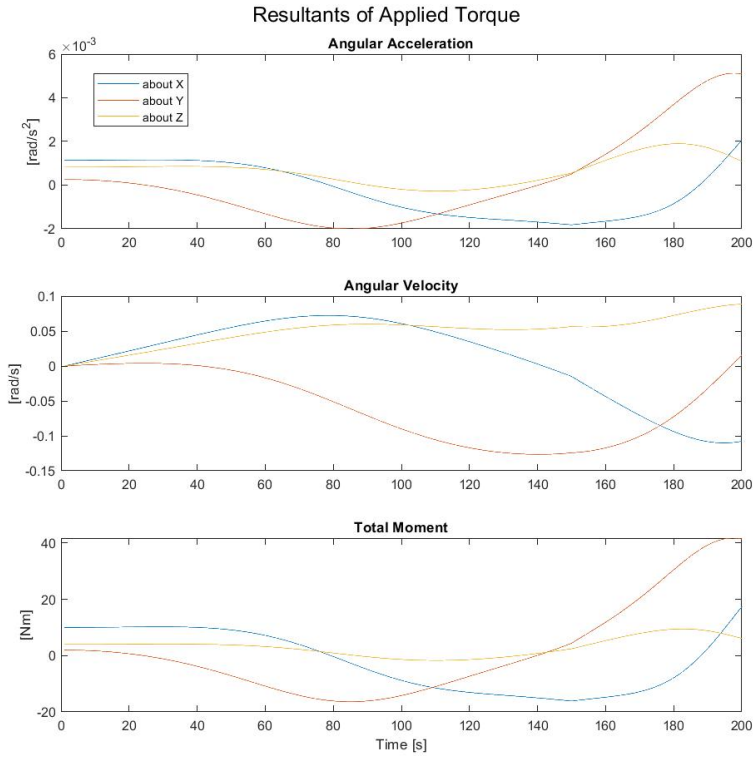

Figure 4.1: Motion of Cassini

4.4.2 Recursive Least Squares Estimation

The estimation algorithm described at the end of Chapter 5 Section 4.3 was used to create the MATLAB function RLSStep.m as seen in Appendix B.3 on page 81. The function was then applied to the code to create an estimate of the relationship between the total moment and angular acceleration data. From Eq. (2.3.4) and Eq. (4.4.3) we know this relationship is defined by the inertia matrix. The output of the RLSStep.m function is a [3x1] vector matrix which can describe the relationships between the angular acceleration and the moment about one axis. Thus the function needs to be run three times in order to obtain column Ix_, column Iy_, and column Iz_.

The error estimated for each estimate versus time is also output which is displayed in the graph below.

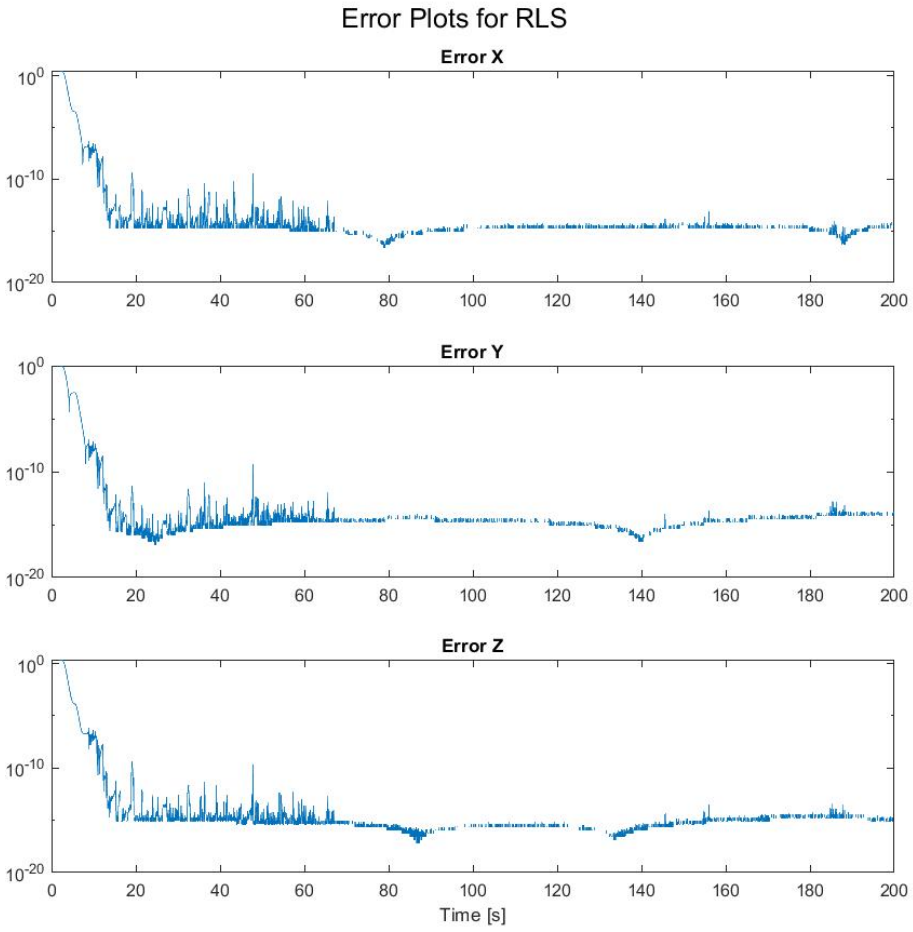

Figure 4.2: Error Plots for RLS

As seen in Figure 4.2, the error for each of the estimates quickly goes below an acceptable threshold of 10^{-06} within the first 20 seconds. This allows for a high degree of confidence in the simulation which output $J_{estimate}$ detailed below.

$$J_{estimate} = \begin{bmatrix} 8810.8 & -136.8 & 115.3 \\ -136.8 & 8157.3 & 156.4 \\ 115.3 & 156.4 & 4721.8 \end{bmatrix} [kg \cdot m^2] \tag{4.4.4}$$

When compared to the Cassini inertia matrix, Eq. (3.1.1), the matrices are identical.

4.5 Simulink Implementation

Verification of the MATLAB implementation is needed to use the code for potential determination of the mass properties of an unknown system. For this a Simulink block diagram was set up to utilize the built in RLS block provided in the Aerospace Blockset of MATLAB. The RLS block was implemented along with the 6 degree-of-freedom block to analyze the movement and determination of the inertia tensor of the Cassini Probe. The inertia tensor will be solved for by first imparting a moment about the x-axis, then the y-axis, and finally the z-axis. This is done as a result of the output of the RLS block which must be a vector. The output of the RLS block, when using moment about x, represents J_{xx}, J_{xy}, and J_{xz}. These values will be directly compared to the same values from the Cassini inertia matrix.

The first iteration of the moment about the x-axis block diagram, without a Recursive Least Squares block, can be seen in Figure 4.3.

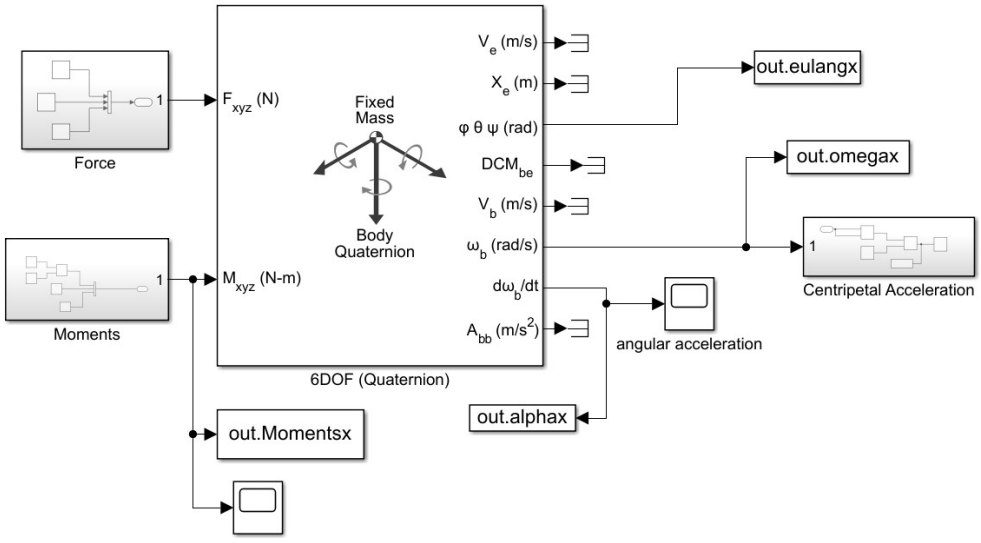

Figure 4.3: 6DOF Block Diagram

The 6DOF block outputs variables that describe the motion of the object being simulated in both the Body Frame and the Earth Centered Inertia Frame. The output is shown in Figure 4.4

where the outputs of the angular acceleration, angular velocity, angular position (Euler Angle), and the input moment are shown. Also shown is the centripetal acceleration which is calculated using Eq. (3.4.1).

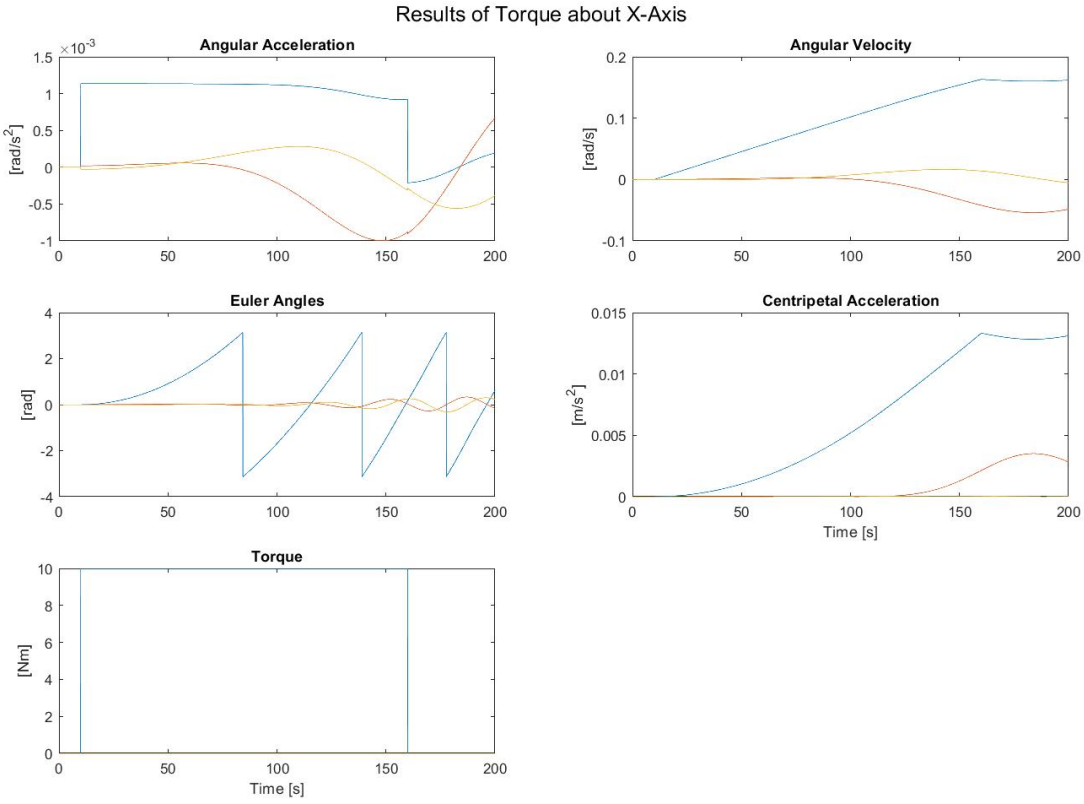

Figure 4.4: 6DOF Output

The output from this system clearly shows an acceleration due to the input torque but it does not tell the whole story. To see what is going on the estimate of the inertia tensor is solved for as shown in Figure 4.5. The x-direction estimated inertia values are displayed above the block diagram. When the values are compared with the values in Eq. (3.1.1), they are incorrect.

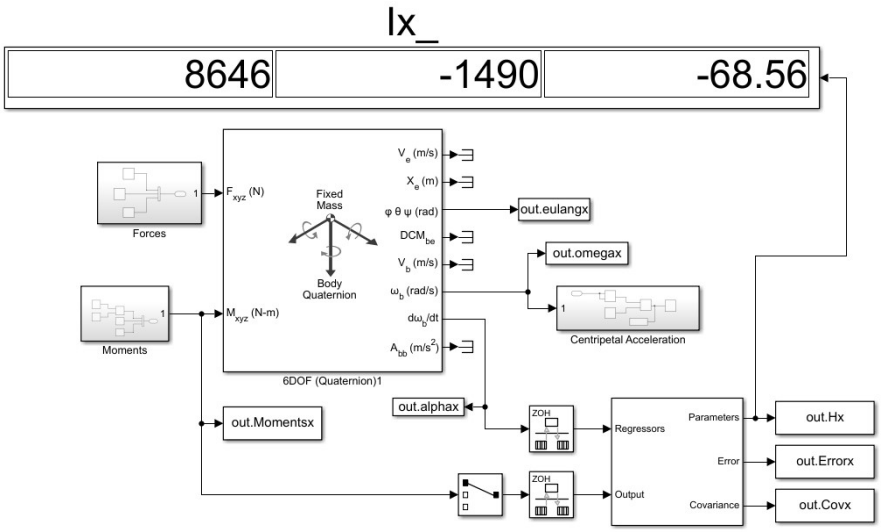

Figure 4.5: Simulink with RLS Block

Investigating the incorrect values lead to comparing the graphs in Figure 4.4 and a disparity is seen in the graphs of moment and angular acceleration. This disparity is due to the expected linear relationship between the two based on Eq. (2.2.4). The error output from the RLS block was also graphed to determine how confident the algorithm is in its estimate. As seen in Figure 4.6, the error value never drops below an acceptable threshold of 10^{-6}.

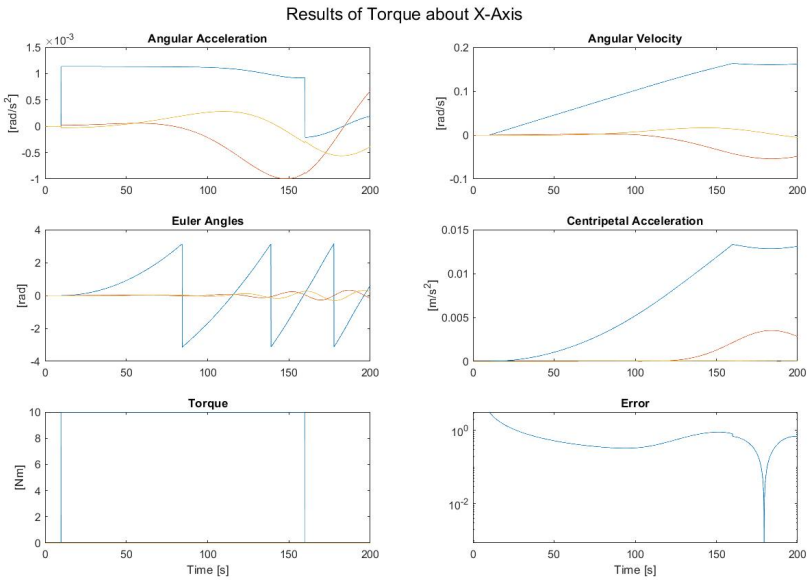

Figure 4.6: RLS Output with Error

To determine why the incorrect values were the result of this simulation it is necessary to look at the inertia equation given in Eq. (2.3.4) and expand it further. When dealing with an inertia matrix that contains non-zero products of inertia, the effects of this asymmetry must be included. Derived from Eq. (4.4.3), the term is shown below.

$$\tau = \boldsymbol{I}\vec{\alpha} + \vec{\omega} \times (\boldsymbol{I}\vec{\omega}) \tag{4.5.1}$$

This is implemented into the Simulink through the subsystem shown in Figure 4.7 and input into the system as shown in Figure 4.8. The RLS section of the diagram has been included in the subsystem shown in Figure 4.9 for future diagrams.

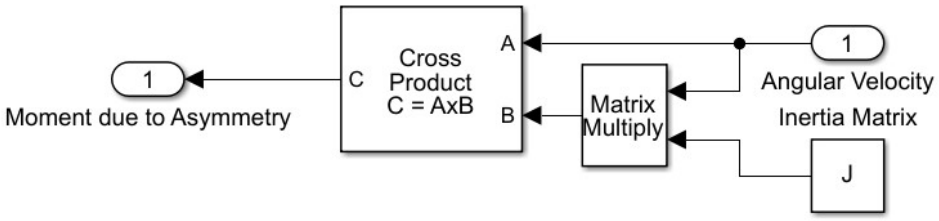

Figure 4.7: Correction Subsystem for Asymmetry

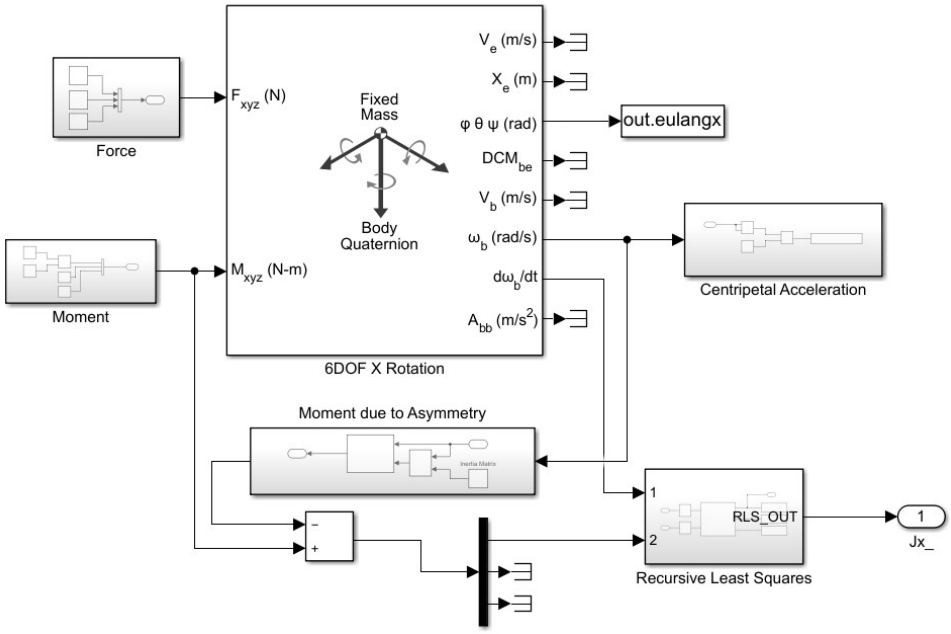

Figure 4.8: RLS with Asymmetry Correction

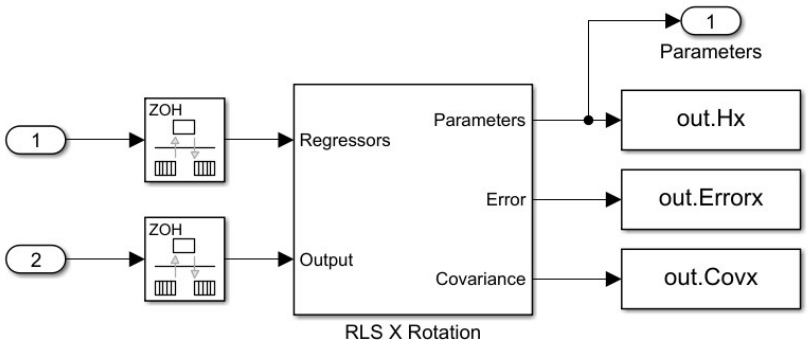

Figure 4.9: RLS Subsystem

Outputting the same graph as before results in the angular acceleration graph and the moment graph looking like each other in a linear way such that Eq. (2.3.4) would predict. The error from the output also goes well past conventional stopping points, 10^{-6}, and drops below 10^{-14}.

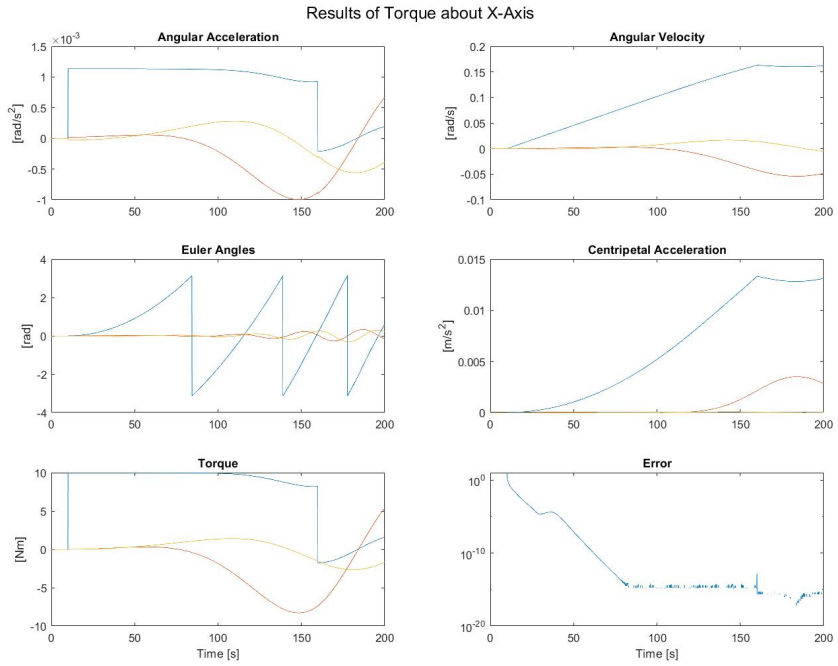

Figure 4.10: Output Corrected for Asymmetrical System

Once the Ix_ solver was complete, Figure 4.11, the same was constructed for Y, Figure 4.12, and Z, Figure 4.13, with the outputs of all being used to construct the total inertia matrix as seen in Figure 4.14.

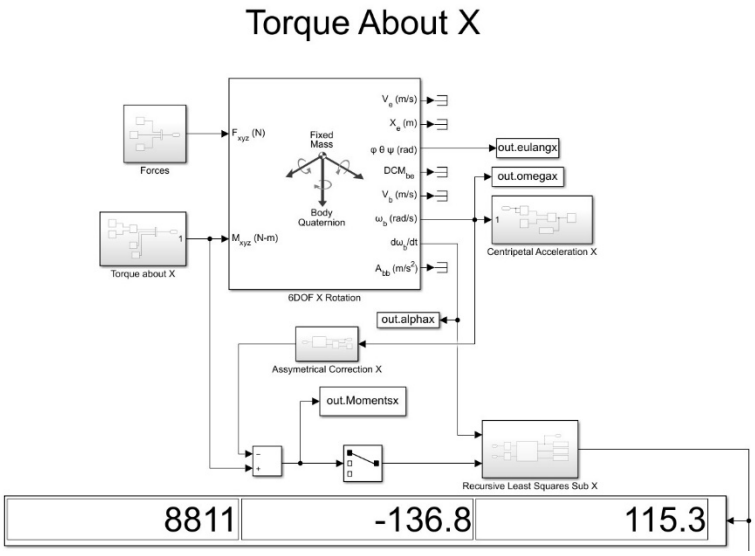

Figure 4.11: Torque about X

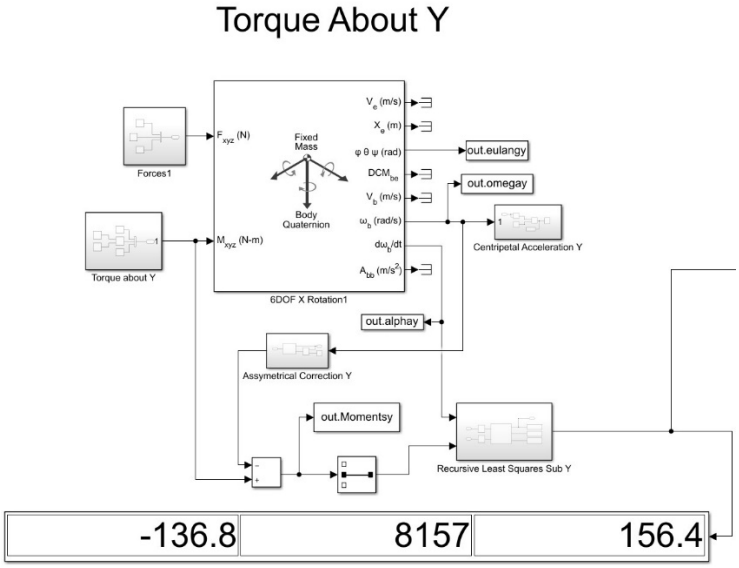

Figure 4.12: Torque about Y

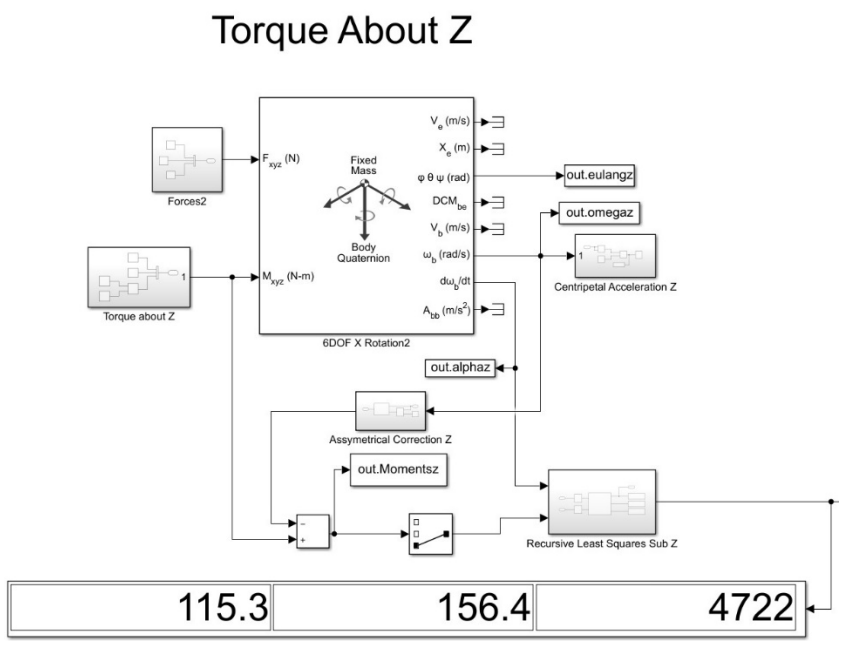

Figure 4.13: Torque about Z

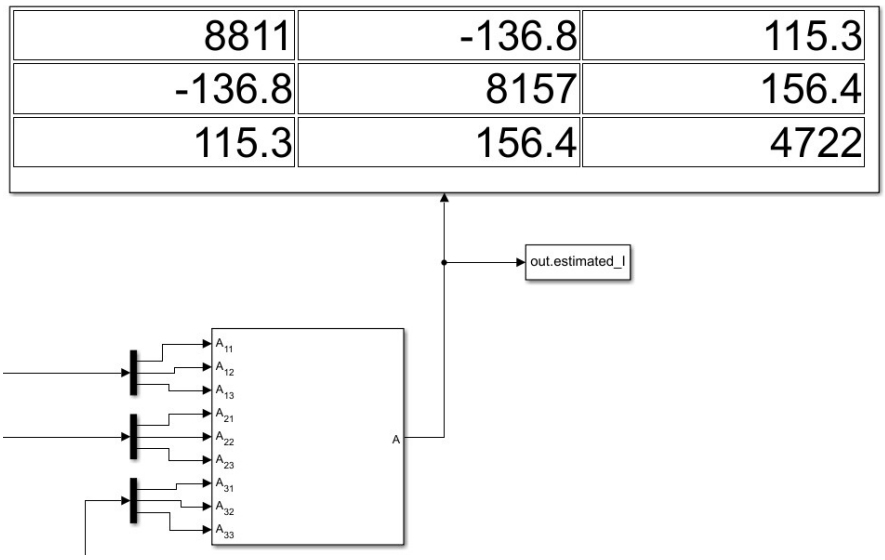

Figure 4.14: Total Inertia Matrix Construction

4.6 Results

Both the Simulink and RLSStep.m algorithm are able to reliably return the true inertia mattrix given multiple variations of input torque. When comparing the error values of the code written for RLSStep.m with that of the Simulink block diagram it can be seen that both surpass 10^{-6}, a

common cutoff threshold. This shows that the code that was written for this paper performs as well as that given by Mathworks. RLSStep.m does maintain several advantages when the time step size, input torque, and inertia matrix are all controlled for. These advantages are that the error values drop past 10^{-6}within 7 seconds, hit a lower floor than the simulink algorithm (10^{-15} vs 10^{-14}), and they reach the floor within 13 seconds while the Simulink errors require at least 75 seconds.

CHAPTER 5: DESIGN VALIDATION

5.1 Overview

The Monte Carlo method is a category of algorithms which use brute force to random sampling to determine the quality or optimal setup of a simulation as is desired by the designer. For control systems this method is generally used for two distinct circumstances being optimization and validation. A Monte Carlo simulation would be considered optimization if a system were being designed and an optimal controller were needed. The method would randomize control parameters many times and the optimal combination would be selected by the constraints put in place by the designer. For a PID controller this would be realized by setting the proportional, integral, and derivative gains (K_P, K_I, and K_D) to be randomly generated within a predetermined range.

Monte Carlo Validation assumes that the designer has a set of gain values that are to be validated and parameters outside of the domain of the controller are varied to create a wide array of conditions under which the desired gain values are tested. Using the PID controller in a mass, spring, and damper scenario, the designer can incorporate noise in the measurements to simulate how well the chosen K_P, K_I, and K_D.

Due to the complexity of some systems, a simulation will be run multiple times to see what effect the variation of the noise level of the variable, or combinations of potential noise, have on a system. This will give the designer a better idea of where money or time should be spent to better optimize the system they are simulating. Another benefit of running smaller, focused, simulations is that the simulation will solve faster and the simulation can be refined for further study at a more rapid pace. After the refinement has been completed larger simulations can be run for final verification of assumptions made to simplify the initial batch of simulations.

5.2 Implementation Example

In order to implement the Monte Carlo Validation in the algorithm described in Chapter 3, an example using a pendulum and cart will first be demonstrated where the varying noise level will be filtered out and the results scrutinized. The importance of the Kalman Filter and its function will be discussed in Chapter 6. For this simulation noise in the distance sensor is used as the changing varible and working from an unknown, instead of known, noise level will challenge the ability of the filter to accurately provide clean data. In this circumstance, the designer of the controller wants to estimate the maximum allowable noise before the assumed covariance, R, is no longer valid. A Monte Carlo Validation algorithm will be implemented to vary the noise level, v_n, and determine the allowable operating range. The code demonstrating this can be found in Appendix B.9. The Simulink is shown below in Figure 5.1.

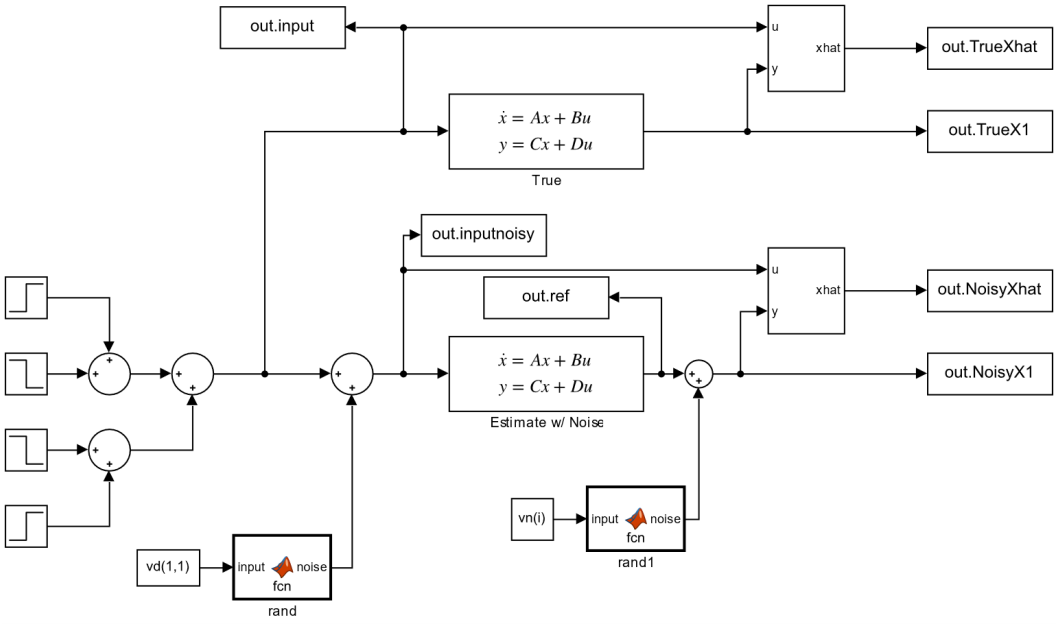

Figure 5.1 Monte Carlo Example

The simulation was setup with two separate state space simulations. The first, labeled "True" contains the pure A, B, C, and D matrices and zero noise input. The second, labeled "Estimate w/ Noise" uses the same matrices but has noise aded to the system both on the input and output in

order to test the Kalman Filter estimator. The effect on the estimation due to noise can be seen below in Figure 5.2. The graph shows the disparity between the true state (red) and the estimates (blue) grow as the magnitude of the noise grows larger.

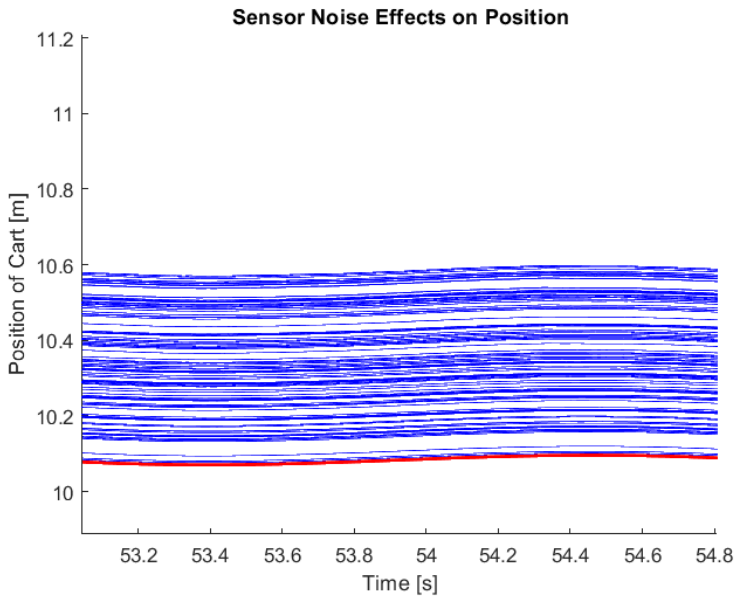

Figure 5.2: Sensor Noise Effects on Position

In the above figure, the minimum noise level, and most accurate estimation pass, was 0.0037 [m]. The designer will determine the threshold by which they can certify the designed system verified or inadequate. For a Kalman Estimation situation the estimate will not be based on what was directly measured (x-position in this example), but by what was estimated (velocity, angle of the pendulum, and the angular velocity of the pendulum. For this example the angle of the pendulum will be considered the most important factor that is to be estimated. Figure 5.3 shows the estimated positions (blue) vs the true position (red) for a assigned covariance of 0.5 and max noise of 1.

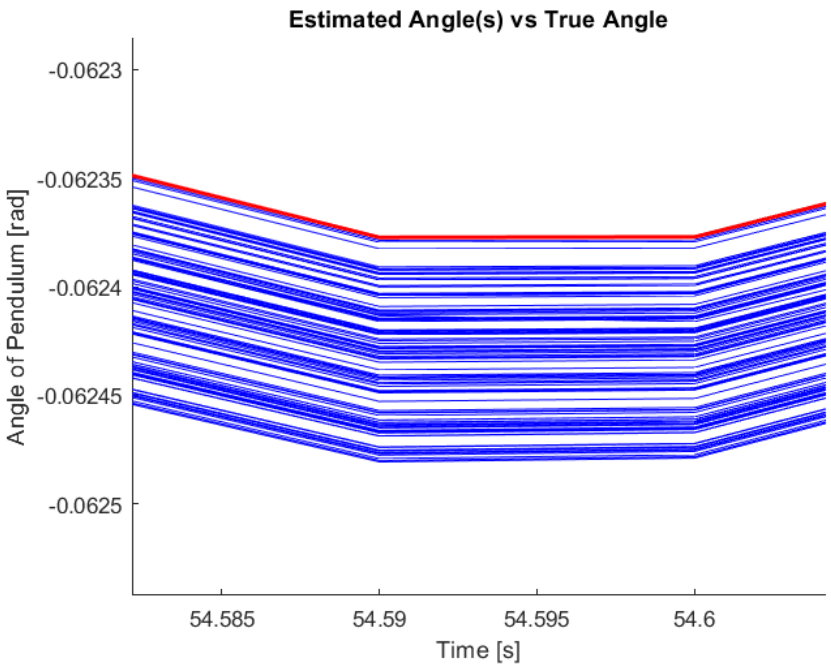

Figure 5.3: Estimated Angle(s) vs True Angle

When comparing the range of the estimations to the true angle it can be seen that there is aproximately a 0.0001 radian diffence in the worst case. This comes out to 0.0057 degrees, which is two orders of magnitude less than the value being estimated at this timestep.

Expanding the noise threshold to $v_n = 5$ the maximum error that would be observed at the same timestep is approximately 0.0003 radians, which is still in the range of 2 orders of magnitude less than the value being estimated. This is shown in Figure 5.4.

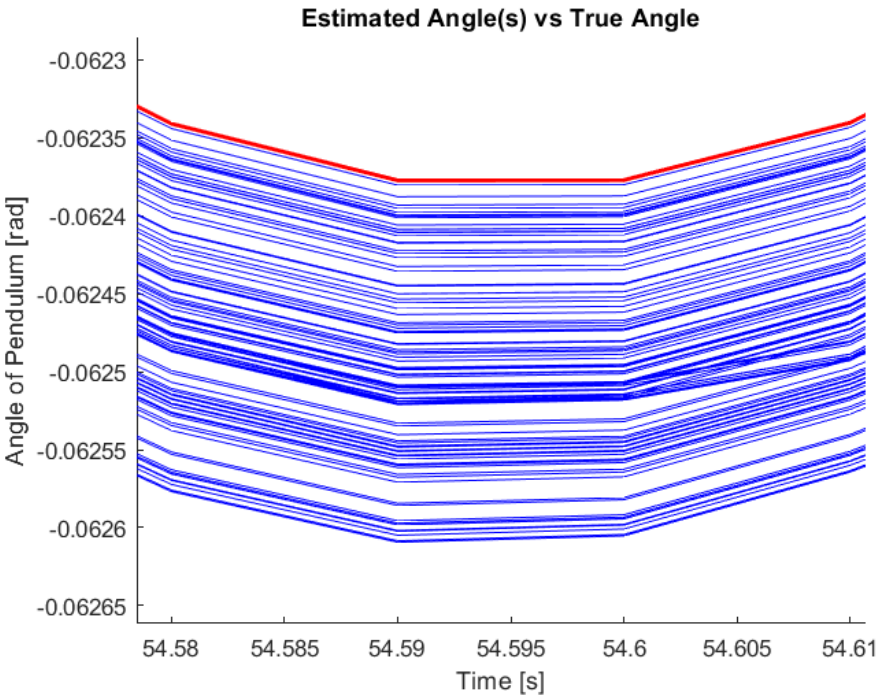

Figure 5.4: Estimated Angle(s) for $\boldsymbol{v_n = 5}$, $\boldsymbol{R = 1}$

5.3 Parameter Identification with Sensor Noise

Random noise was implemented into the Parameter Identification scheme detailed in Chapter 3. This introduction of noise required the use of the built in MATLAB randn() function that creates Gaussian white noise based off of the inputs that the designer uses. Gaussian white noise is generated in a fashion that creates a distrobution similar to the noise that is seen in sensors and other electronic devices. The level of noise trends towards the average, zero in this particular use case, and can be seen in the figure below.

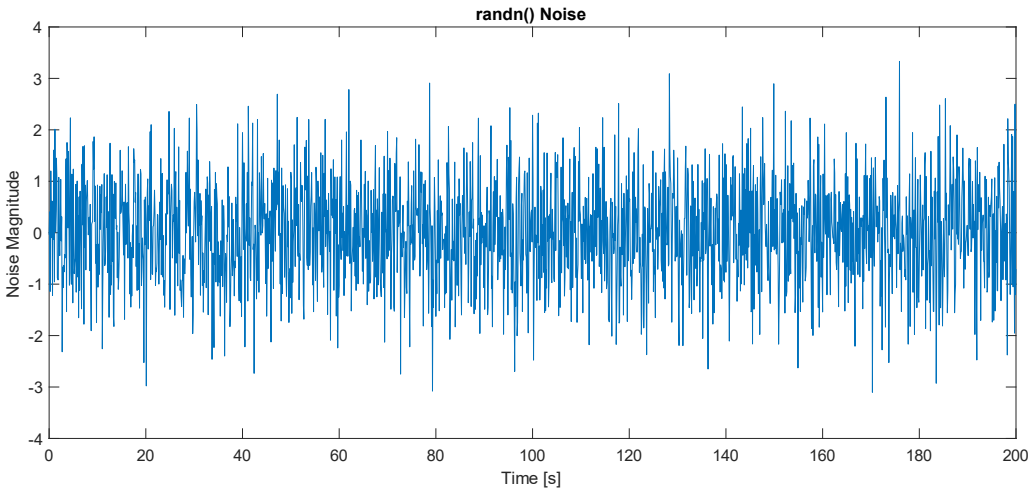

Figure 5.5: Noise Generated by randn() function

In the implementation in Simulink the noise will be generated using the Eq. (5.3.1)

$$Signal_{Noise} = Signal_{Original} + \frac{randn(3,1)}{3} * Noise_{Magnitude} \tag{5.3.1}$$

The randn() signal is divided by 3 to create a unit distribution then multiplied by the desired magnitude and added to the original signal. This is implemented for each of the readings that an IMU would make as shown in the figure below enclosed by the blue box. The MATLAB code for the function can be found in Appendix B.14.

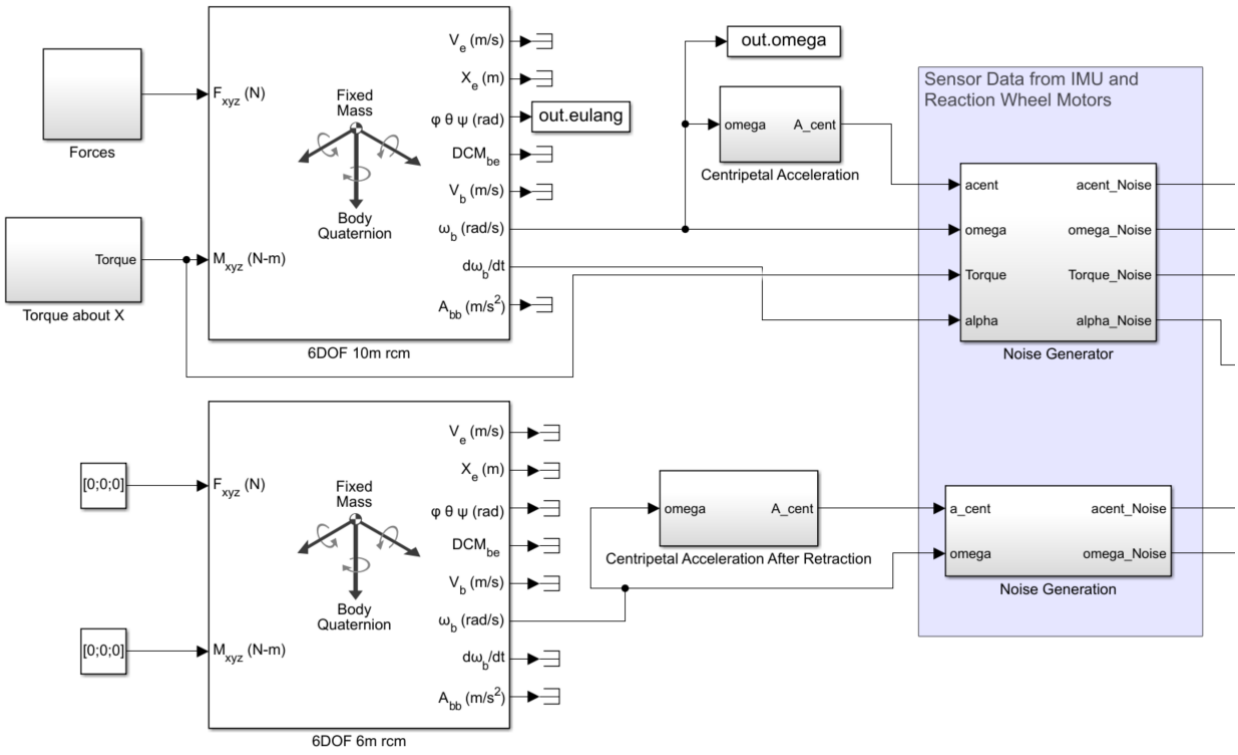

Figure 5.6: Simulink Implementation of Sensor Noise

The code encompasing the loop and noise magnitude can be found in Appendix B.15. This code does not contain the graphing or initial conditions due to them being near identical to the code in Appendix B.13.

5.4 Noise Level Analysis

A focused study was completed that looked at a max of two variables at a time in order to understand the effects that each variable has on the estimation of the desired parameters. To get a wide view of what the potential error values look like due to variations in omega noise and linear acceleration noise a simulation of 100 iterations was run with the noise magnitude for both types being between 10^{-1} and 10^{-8}. The results are shown below in Figure 5.7.

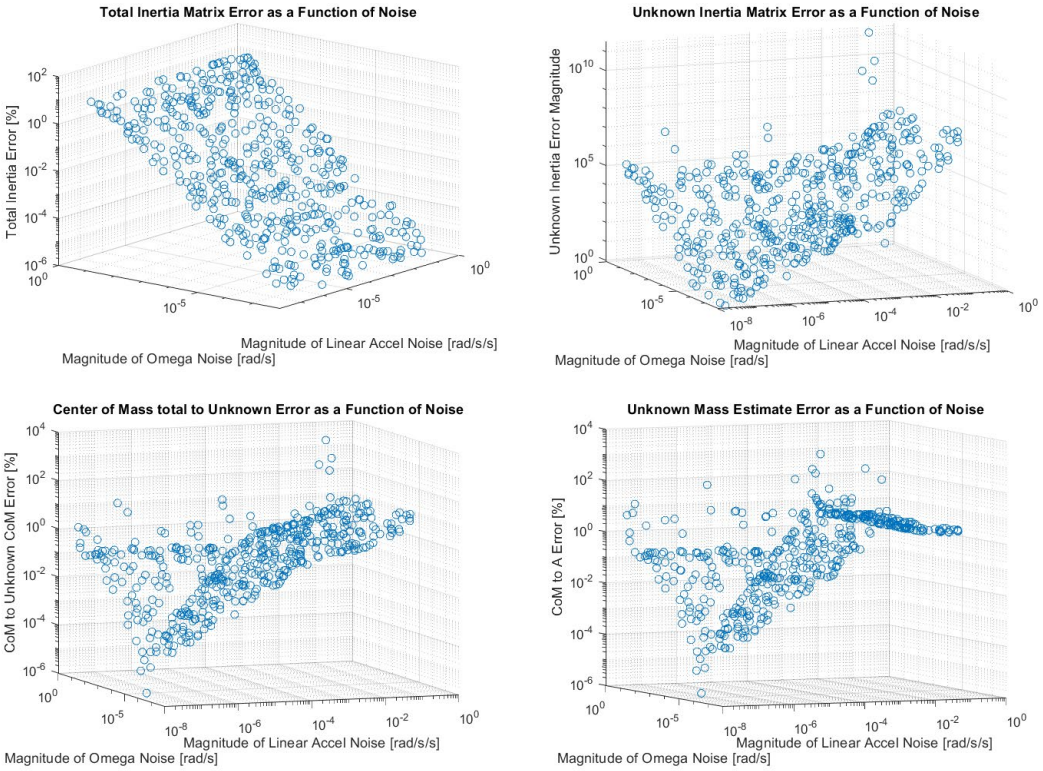

Figure 5.7: Monte Carlo Results of Omega and Linear Acceleration Noise

In the figure about the Total Inertia Matrix Error it can be seen that the error is only affected by the change in omega noise. This matches with what should be expected by the formulas given in Section 2.1 and as can be seen in the Simulink diagram given by Figure 3.6 where the linear acceleration is not seen in the inputs to the estimation scheme. The other plots, when moved in the 3-D space, show a trend in the form of a trough where the optimal sensor selection can be made in relation to its noise floor.

After running the simulation for 500 iterations the trough can be seen more clearly by looking at Figure 5.8. This trough in the data represents the ideal ratio between the noise level seen by the sensor in rotational velocity and linear acceleration. When chosing a sensor and the noise ratio skews towards one sensor or the other that will degrade the performance regardless of how

sensitive the other sensor is. This can be seen by looking at the graph from the XZ or YZ frame as with the example below, in Figure 5.9, showing the Unknown Inertia Matrix vs rotational velocity.

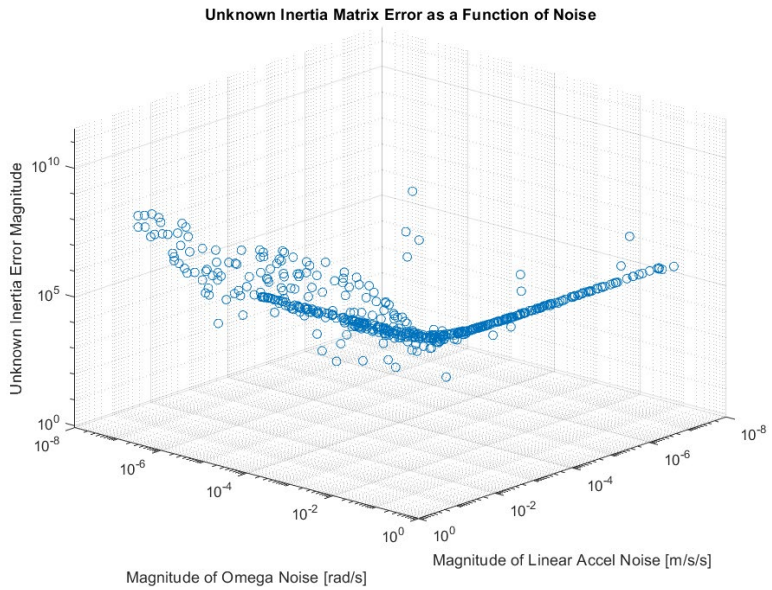

Figure 5.8: 500 Iteration varying rotational velocity and linear acceleration looking down the trough

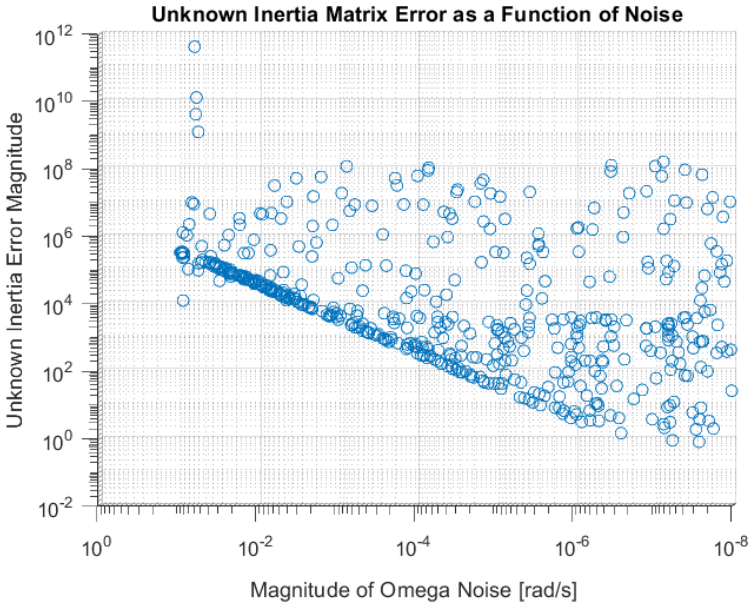

Figure 5.9: Side view looking at rotational velocity

In the area where a clear line is seen, the data points are linear and soley dependent on rotational velocity until the accuracy ratio approaches that of the trough showing that the noise in angular velocity is too great and overwhelms the noise in linear acceleration. When the accuracy ratio surpasses the trough the opposite is true.

For this circumstance where the unknown inertia matrix is given by Eq. (3.5.1) a reasonable error magnitude would be less than 20 [kg m^2] and a good error magnitude would be less than 2 [kg m^2]. For the reasonable estimate the omega noise magnitude would need to be less than $5x10^{-7}\left[\frac{rad}{s}\right]$ and the linear acceleration noise magnitude would need to be less than $4x10^{-7}\left[\frac{m}{s^2}\right]$. This would produce the following figures showing the original signal versus the max allowable noisy signal.

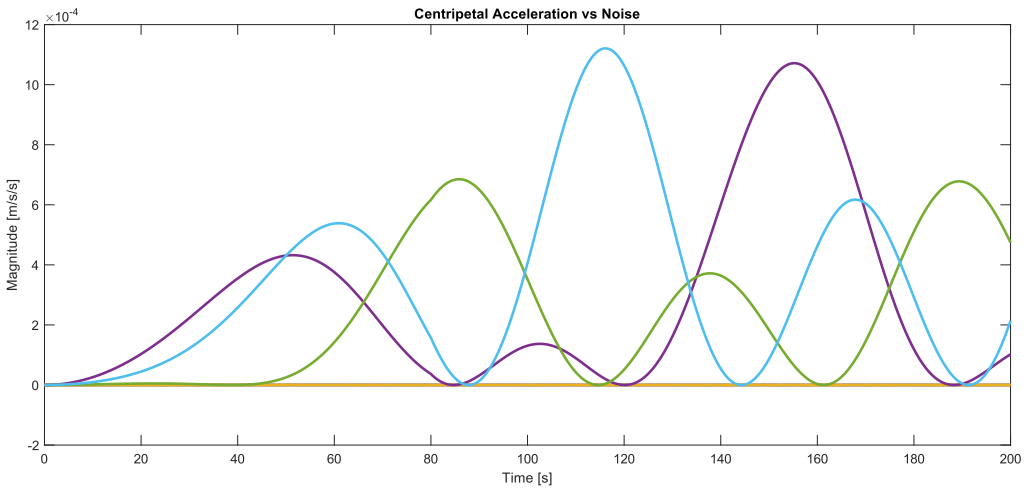

Figure 5.10: Centripetal Acceleration vs Noise

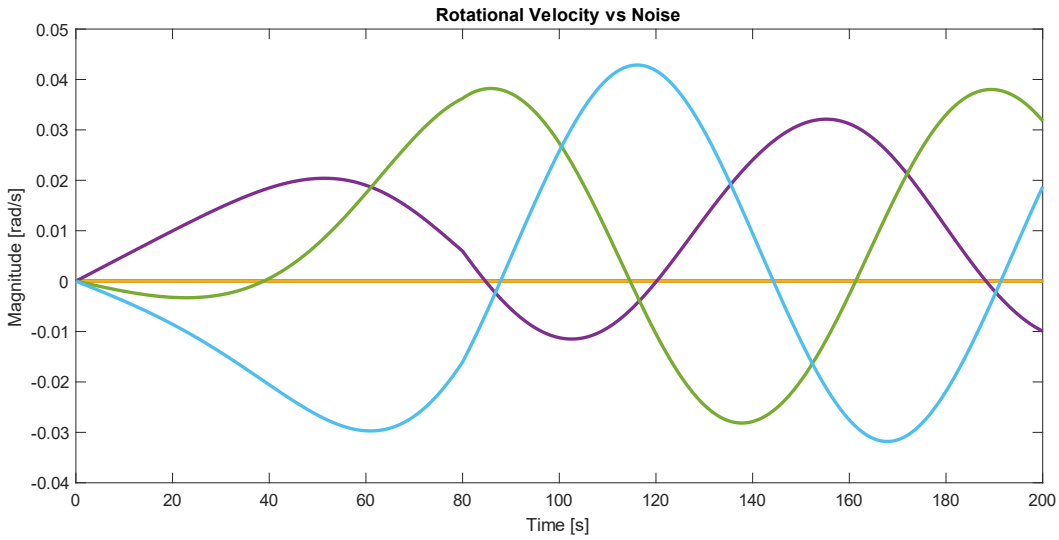

Figure 5.11: Rotational Velocity vs Noise

In both Figure 5.10 and Figure 5.11 the original signals are blue, green, and purple while the noise is represented by the orange line. The magnitude of the noise is so much less than the magnitude of the original signals that the plot of the noise appears to be constantly zero. This noise to signal ratio is unacceptable and the noise must be filtered out at higher magnitudes through a noise filter.

5.5 Recursive Least Squares Filter

5.5.1 Setup

Similar to the RLS Estimation in Chapter 4, a Recursive Least Squares Filter will seek to minimize the difference between the mean estimate and the current time step measurement, or error, squared. This is implemented in Simulink in order to clear the noise generated due to imperfect sensors as shown in Figure 5.12.

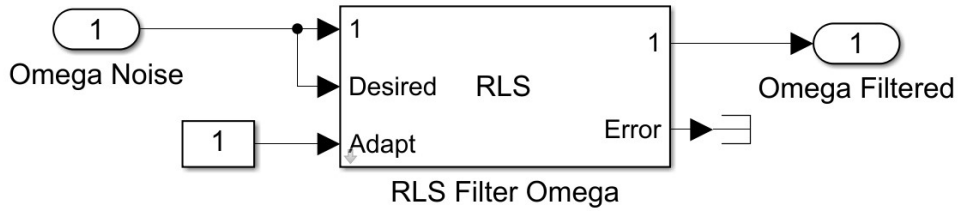

Figure 5.12: RLS Filter

When the RLS filter is run with an omega noise magnitude of $10^{-2.5}[rad/s]$ a portion of the noise can be filtered out, as shown in Figure 5.13. The lower graph contains just the noisy signal while the upper contains the true the filtered signal. The magnitude of the noise is less in the filtered signal and this allows for more accurate results when isolating the desired parameters. The same filtering can be seen for the linear acceleration sensor with a noise magnitude of $10^{-4}[rad/s]$ in Figure 5.14.

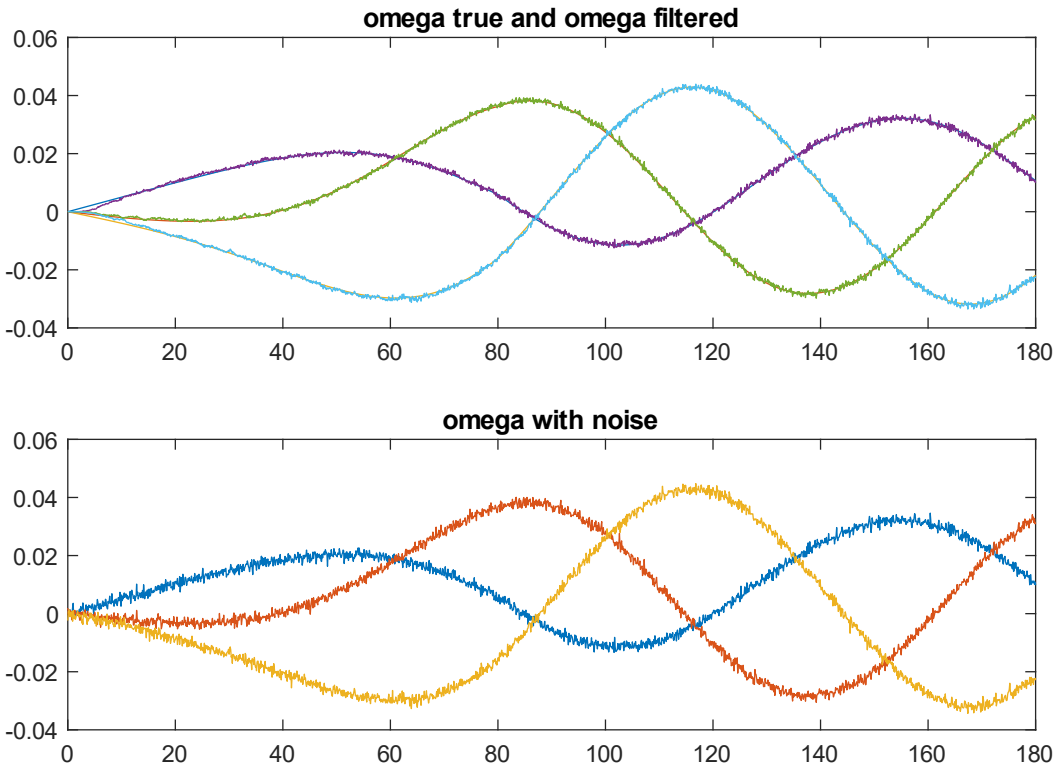

Figure 5.13: RLS Filtering of Omega Noise

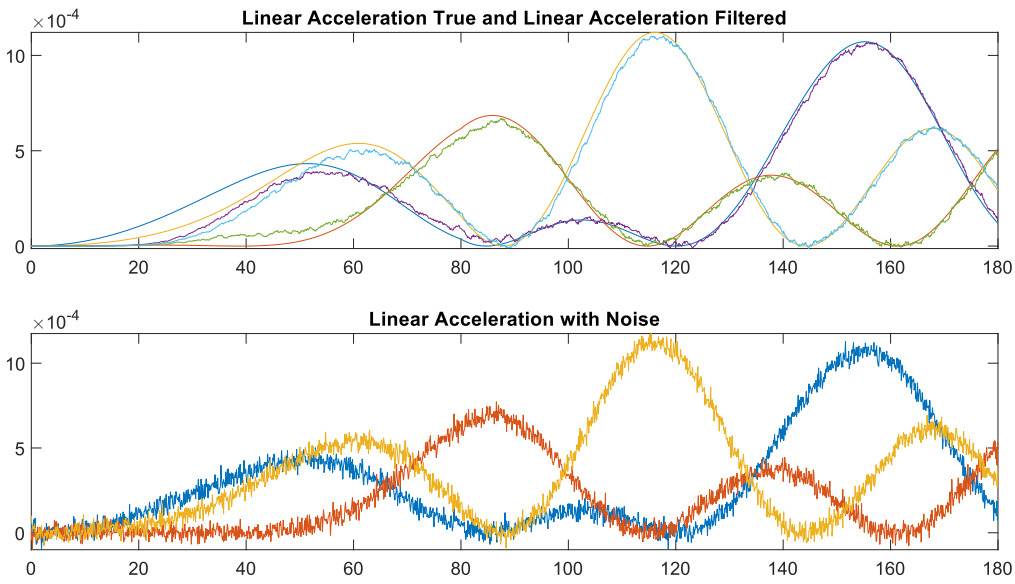

Figure 5.14: RLS Filtering of Linear Acceleration Noise

Using the RLS Filtering blocks between the noise generation and the first set of calculations completed represents the flight computer apllying the filtering method to the data before attempting to calculate the required unknowns. This setup can be seen below in Figure 5.15.

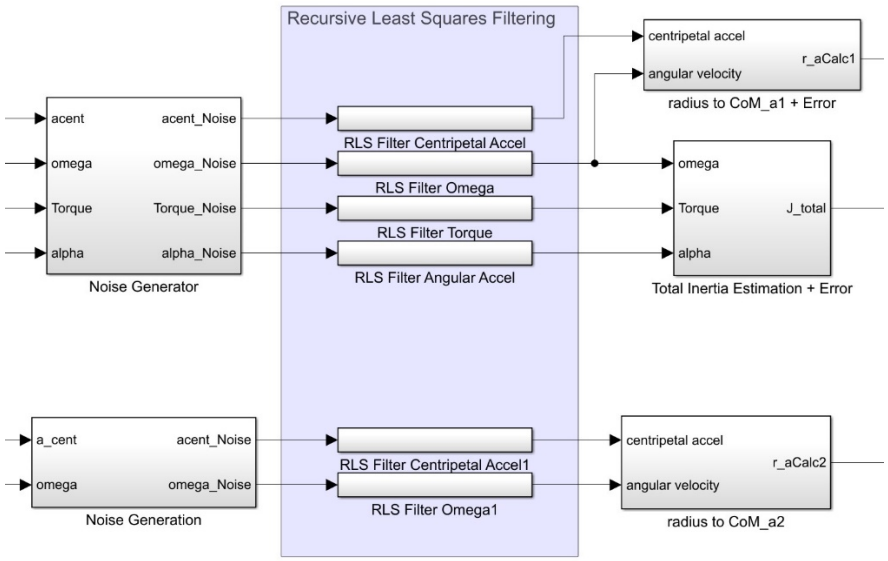

Figure 5.15: RLS Filtering in Simulink

5.5.2 Results

The implementation of the RLS Filter was partially successful due to the damping effect the filter had on the noise magnitude. It was also not successful due to the poor estimation qualities that it displayed. This is likely due to the filter not incoporating the dynamics of the system into the estimation effort and soley relying on the data that it was provided then attempting to reduce the squared error. The use of a Kalman Filter or similar filter type that adapts to the state space matrices would improve the estimation.

Using the graphs in Figure 5.16, it can be seen that in order to reach a 1% error the rotational velocity noise must have a noise magnitude of less than 10^{-4} $[rad/s]$ for the Unknown Inertia Matrix, Center of Mass vector, and the Unknown Mass Estimate. When comparing this to Figure 5.13 using the magnitude of peak rotation rate of 0.04 $[rad/s]$ this equates to a noise to signal ratio of 0.25%. The formula can be seen in Eq. (5.5.1).

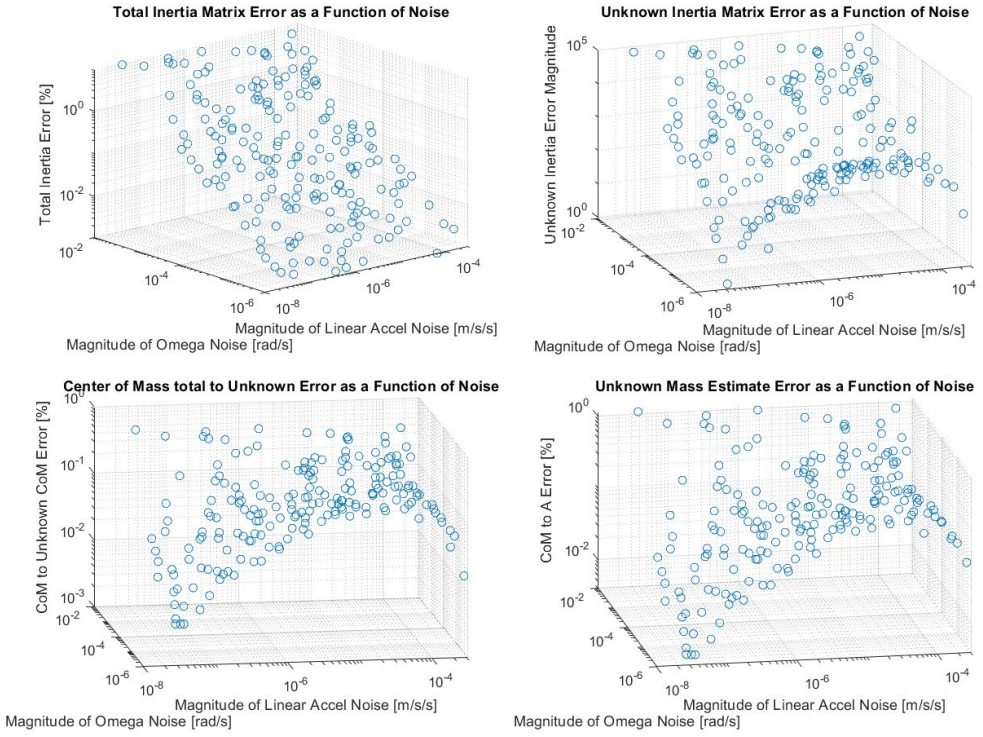

Figure 5.16: Parameter Estimation with Noise Reduction by RLS Filter

$$\frac{NoiseMagnitude}{SignalMagnitude} = Noise{:}Signal \tag{5.5.1}$$

Again using Eq. (5.5.1), the linear acceleration noise magnitude required to reach a 1% error, 10^{-7} $[m/s/s]$, yields an allowable noise to signal ratio of 0.1%. Both the allowable error for the rotational velocity and linear acceleration are well below what would be considered an acceptable threshold.

CHAPTER 6: CONCLUSIONS AND FUTURE WORK

6.1 Completed Work

The equations of rotational motion and spacecraft dynamics from Chapter 2, were used to separate the inertia matrix, mass, and center of mass of Spacecraft B from a theoretically coupled spacecraft duo, as seen in Chapter 3. The output of which was seen to be extremely accurate assuming no noise in the system. To combat a noise level representing a real world system a recursive least squares filter was proposed in Chapter 4. A design validation scheme was implemented in the form of a Monte Carlo method in Chapter 5 to test a variety of noise levels against the designed filter. The recursive least squares filter was determined to be inaccurate enough that it, in the current form, does not represent a usable noise filtering method. In Chapter 6, a state estimation algorithm in the form of the Kalman Filter was described and implemented, in a well studied example, to show its ability to filter noise and estimate all observable states.

6.2 Challenges Faced

Through the first several iterations of the recursive least squares algorithm the inertia matrix output was routinely incorrect by greater than 20% when compared to the inertia matrix used to calculate the movement of the spacecraft. The first major correction was taking into account the product of inertia influence described in Eq. (4.4.3). This correction brought the error to within 5% of the correct inertia value for the principal axes but the error was still large for the products of inertia. The inclusion of a forgetting factor λ brought the error value output by the algorithm to well below an acceptable threshold of 10^{-6} as well as matching the input matrix exactly.

When developing the design validation algorithm an incorrect Monte Carlo method was initially implemented that utilized a static, yet random, grid for the tested noise levels creating a semi-uniform plot. This was due to the use of preset random noise values that were then used in

a nested double for-loop. A single for-loop with the random values generated inside the loop was used instead which resolved the issue. Following the implementation of the recursive least squares filter a significant error rate was observed which was eventually resolved when the angular acceleration estimate was seen to be multiple orders of magnitude less than the true value for all three axes. This was solved by introducing a more accurate initial noise estimate for the filter.

During the application of the Extended Kalman Filter the states would not settle and the filter never reached an acceptable error determined to be an average of $< 5\%$ across all states measured given the ideal circumstances used. This issue is not resolved at this time but will be documented as future work.

6.3 Future Work

6.3.1 Physical Attributes vs Noise Level Investigation

The investigation between how changing specific physical attributes of the spacecraft system affect the allowable noise level is an important part of the design process and will impact the choice of sensors for such a spacecraft. The recommended relationships are:

1. Mass ratio of the unknown and known spacecraft
2. Ratio of total inertia to the known ability of the spacecraft to impart a torque on the system
3. What parameter is the most important for accurate estimation

6.3.2 State Estimation

6.3.2.1 Overview

All systems have some level of uncertainty, which is created due to measurement error, inaccuracies of production, or the inability to measure specific states. Because of this an

inaccurate model of the system is utilized by the flight computer which results in imprecise calculations which cannot be corrected using a controller. State estimation attempts to circumvent this issue through the manipulation of the known equations of motion and the sensor data being measured. This method determines the best guess of the true states for the system. Once true sensor data is attained, unknown states can be estimated and utilized by the controller. The result is an invaluable resource for systems without a human in the loop (for example, self-driving vehicles, aircraft autopilot, spacecraft) that depend on having accurate data with the effects of noise negated.

To estimate the states of a system, that system must be fully observable. If a system is fully observable, then all the state variables can be determined through measurement of the system output. The observability of a system can be determined by finding the rank of the observability matrix shown below.

$$\boldsymbol{Ob} = \begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{CA} \\ \dots \\ \boldsymbol{CA}^{n-1} \end{bmatrix} \tag{6.3.1}$$

The matrix uses the $\boldsymbol{A}$ and $\boldsymbol{C}$ matrices from the state space matrix of the system where n is the number of states being observed. If the rank of the matrix is equal to n then the system is fully observable. If the rank, r, is less than n then the system only has r states that can be observed.

As the quality of the estimation increases, the controllability of the system also increases due to the true state matrix being approximated. During the discussion of state estimation, the test case of a pendulum, Figure 6.1, and an inverted pendulum, Figure 6.2, balanced on a moving cart will be used.

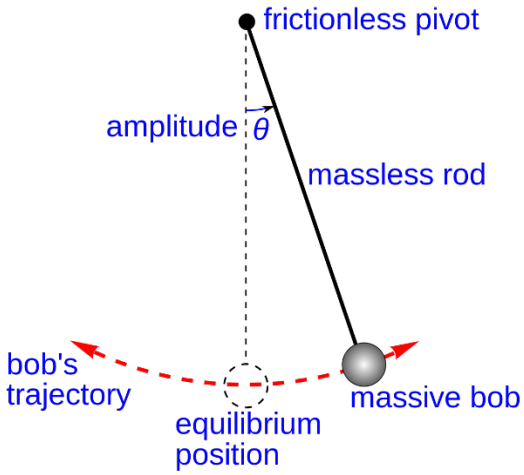

Figure 6.1: Pendulum [25]

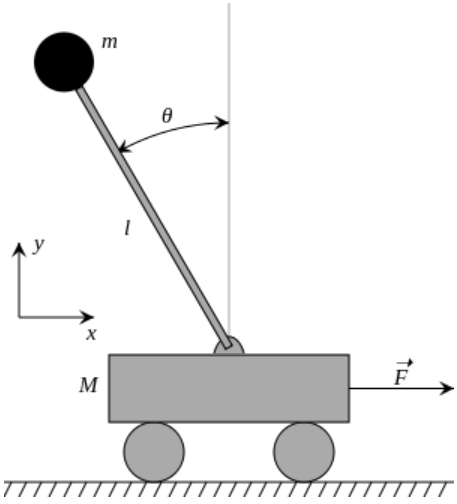

Figure 6.2: Inverted Pendulum on Cart [26]

6.3.2.2 Kalman Filter

The Kalman Filter represents the optimal full state estimator utilizing the inputs, usually in the form of a force or moment, and the output state of the system. In a physical control system, the output state most commonly represents location, velocity, or acceleration, all of which can be linear or rotational. The Kalman Filter makes several assumptions about the system being modeled.

The first of these is that the system is linear in nature and the general kinematics do not change over time. Practically, this means that the matrices in the State Space system must be consistent in any state. An object that is moving on a consistent surface would be an ideal use case for a Kalman Filter. The second assumption is that there is a level of noise that must be accounted for being created by the sensor and the system being modeled. This noise, for a Kalman Filter is assumed to be Gaussian in nature. Gaussian noise has the characteristics of a normal distribution with a mean of the true signal, or zero mean. An example of Gaussian zero mean sensor noise is shown in Figure 6.3.

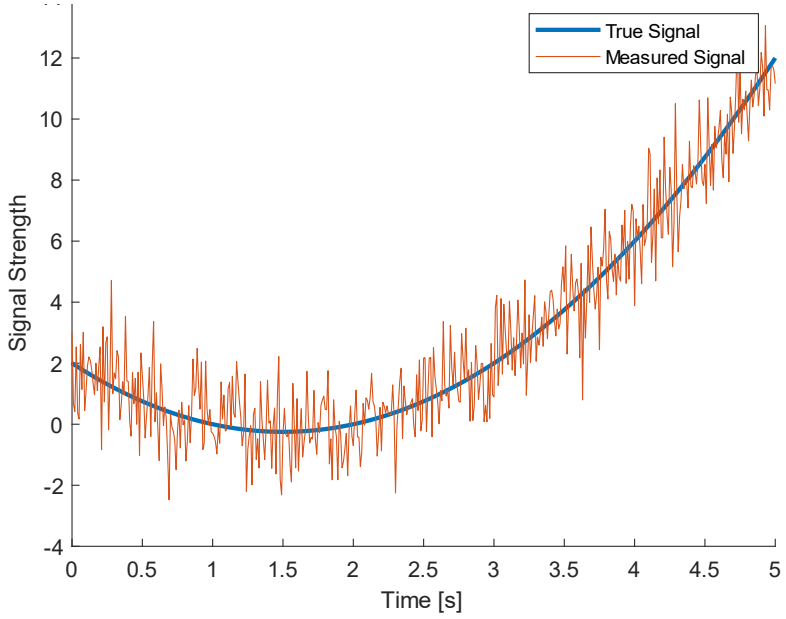

Figure 6.3: Zero Mean Signal Noise Representation

Summarizing the Kalman Filter from Appendix A.1 on page 69 gives the following set of equations:

Calculate the Kalman Gain using Eq. (A.1.15):

$$\boldsymbol{K}_k = \boldsymbol{P}_{k-1}\boldsymbol{C}^T(\boldsymbol{C}\boldsymbol{P}_{k-1}\boldsymbol{C}^T + \boldsymbol{R})^{-1}$$

Update the State Estimate using Eq. (A.1.5):

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_{k-1} + \boldsymbol{K}_k(\boldsymbol{y}_k - \boldsymbol{C}\hat{\boldsymbol{x}}_{k-1})$$

Update the error Covariance in response to the Kalman Gain from Eq. (A.1.16):

$$\boldsymbol{P}_k = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C})\boldsymbol{P}'_k$$

Project to next step using Eq. (A.1.17) and Eq. (A.1.21):

$$\hat{\boldsymbol{x}}'_{k+1} = \boldsymbol{A}\hat{\boldsymbol{x}}_k$$

$$\boldsymbol{P}_{k+1} = \boldsymbol{A}\boldsymbol{P}_k\boldsymbol{A}^T + \boldsymbol{Q}$$

6.3.2.3 Extended Kalman Filter

An Extended Kalman Filter, EKF, is applied to a particular system when the system being analyzed is non-linear in nature. The most common example used in academics is the inverted pendulum supported by a cart, as shown in Figure 6.2. The dynamics of an inverted pendulum are well documented and exceptionally well linked to the movement of the cart allowing for discussion about many topics.

With any non-linear system, the equations of motion are initially linearized about time $t = 0$. Due to the characteristics of a non-linear system, the equations of motion will have to be linearized at some time interval. This interval can be set at a predetermined number of iterations, after the error value exceeds a limit, or after each successive estimation. This decision is ultimately left up to the designer.

The Extended Kalman Filter works around the state space equations varying with time which changes the Kalman Filter algorithm to the following form:

Calculate the Kalman Gain:

$$\boldsymbol{K_k = P_{k|k-1} C_k^T \left(C_k P_{k|k-1} C_k^T + R_k \right)^{-1}}$$

Update the State Estimate:

$$\boldsymbol{\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \left(y_k - C_k \hat{x}_{k|k-1} \right)}$$

Update the error Covariance in response to the Kalman Gain:

$$\boldsymbol{P_{k|k} = (I - K_k C_k) P'_{k|k-1}}$$

Project to next step using:

$$\boldsymbol{\hat{x}'_{k+1|k} = A_k \hat{x}_{k|k}}$$

$$\boldsymbol{P_{k+1|k} = A_k P_{k|k} A_k^T + Q_k}$$

6.3.2.4 MATLAB Example Implementation

The results of such a filter is demonstrated by Brunton & Kutz [27] through the use of the MATLAB function $lqe()$. The algorithm from the book can be found in Appendix B.8. $lqe()$ outputs a gain value that will correct for the sensor noise in the data. This result can be seen in Figure 6.4: X-Position for Pendulum on Cart where the noisy data from a distance sensor on a pendulum carrying cart is run through a Kalman Filter to estimate the true value.

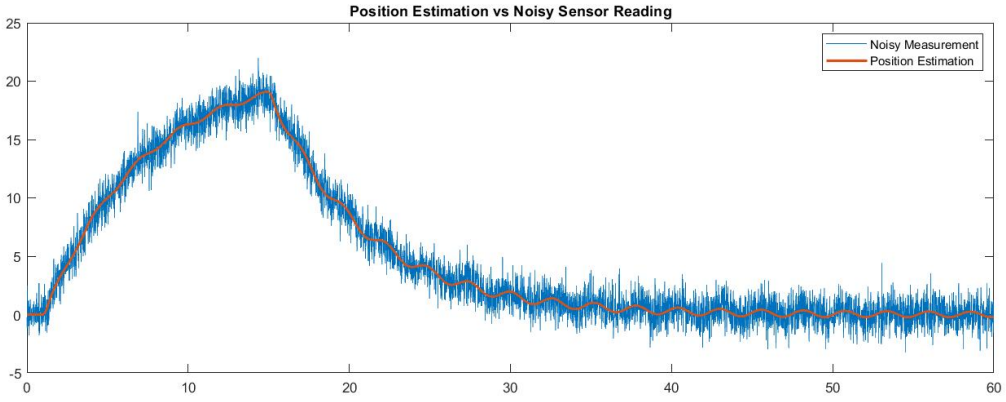

Figure 6.4: X-Position for Pendulum on Cart

Full state estimation can also be implemented with the $lqe()$ function. Doing this when only measuring the state x-position provides the graphs shown in Figure 6.5 and Figure 6.6. For both graphs the solid line indicates the true state while the dashed line indicates the estimated states being calculated from the x values only. Figure 6.6 is a close up view of Figure 6.5 as the estimates are accurate enough that they are very difficult to discern when looking at the full graph.

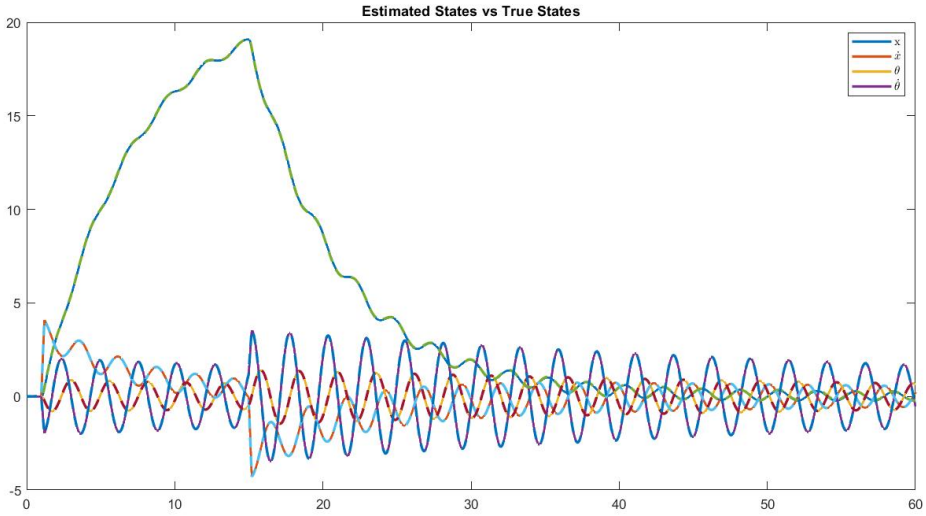

Figure 6.5: Estimated vs True States for Pendulum on a Cart

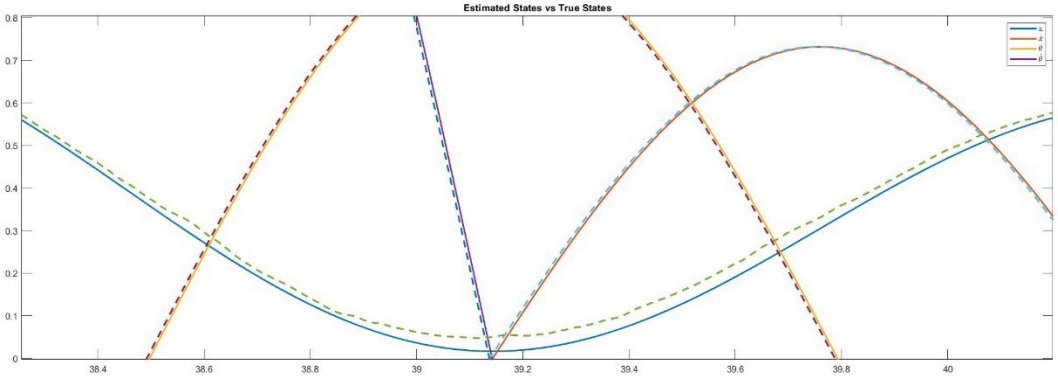

Figure 6.6: Close up view of State vs Estimated States

REFERENCES

[1] "Space debris by the numbers," *ESA* Available: https://www.esa.int/Safety_Security/Space_Debris/Space_debris_by_the_numbers.

[2] Shan, M., Guo, J., and Gill, E., "Review and comparison of active space debris capturing and removal methods," *Progress in Aerospace Sciences*, vol. 80, 2016, pp. 18–32.

[3] Leomanni, M., Bianchini, G., Garulli, A., Giannitrapani, A., and Quartullo, R., "Orbit Control Techniques for Space Debris Removal Missions Using Electric Propulsion," *Journal of Guidance, Control, and Dynamics*, vol. 43, 2020, pp. 1259–1268.

[4] Nishida, S.-I., and Kawamoto, S., "Strategy for capturing of a tumbling space debris," *Acta Astronautica*, vol. 68, 2011, pp. 113–120.

[5] Aslanov, V., and Yudintsev, V., "Dynamics of large space debris removal using tethered space tug," *Acta Astronautica*, vol. 91, 2013, pp. 149–156.

[6] Bergmann, E., Walker, B., and Levy, D., "Mass property estimation for control of asymmetrical satellites," *Guidance, Navigation and Control Conference*, 1985.

[7] Ma, O., Dang, H., and Pham, K., "On-Orbit Identification of Inertia Properties of Spacecraft Using a Robotic Arm," Journal of Guidance, Control, and Dynamics, vol. 31, 2008, pp. 1761–1771.

[8] Sharifi, E., and Damaren, C. J., "Nonlinear Optimal Approach to Spacecraft Attitude Control Using Magnetic and Impulsive Actuations," *Journal of Guidance, Control, and Dynamics*, vol. 43, 2020, pp. 1154–1163.

[9] Bergmann, E., and Dzielski, J., "Spacecraft mass property identification with torque-generating control," *Journal of Guidance, Control, and Dynamics*, vol. 13, 1990, pp. 99–103.

[10] Lee, A. Y., and Wertz, J. A., "In-Flight Estimation of the Cassini Spacecrafts Inertia Tensor," *Journal of Spacecraft and Rockets*, vol. 39, 2002, pp. 153–155.

[11] Norman, M. C., Peck, M. A., and Oshaughnessy, D. J., "In-Orbit Estimation of Inertia and Momentum-Actuator Alignment Parameters," *Journal of Guidance, Control, and Dynamics*, vol. 34, 2011, pp. 1798–1814.

[12] Krstic, M., and Tsiotras, P., "Inverse optimal stabilization of a rigid spacecraft," *IEEE Transactions on Automatic Control*, vol. 44, 1999, pp. 1042–1049.

[13] Salcudean, S. "A Globally Convergent Angular Velocity Observer for Rigid Body Motion," IEEE Transactions on Automatic Control, 36 (12), 1991, pp. 1493–1497.

[14] Khosravian, A. and Namvar, M. "Globally Exponential Estimationof Satellite Attitude using a Single Vector Measurement and Gyro," *Proceedings of the 49th IEEE Conference on Decision and Control*, December 15-17, 2010.

[15] Ruiter, A. H. J. D., "Observer-based spacecraft attitude tracking with guaranteed performance bounds," *2015 American Control Conference (ACC)*, 2015

[16] Kim, D. H., Yang, S., Cheon, D.-I., Lee, S., and Oh, H.-S., "Combined estimation method for inertia properties of STSAT-3," *Journal of Mechanical Science and Technology*, vol. 24, 2010, pp. 1737–1741

[17] Ren, L., Ban, X., Huang, X., and Ding, S., "Parameter Identification of Gyro Using Recursive Subspace Algorithm," *IFAC-PapersOnLine*, vol. 48, 2015, pp. 662–667.

[18] Ni, Z., Liu, J., Shen, X., and Chang, C., "On-orbit identification of spacecraft time-varying moment of inertia using an improved recursive subspace method," *2017 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, 2017.

[19] Byeon, S., Lee, H., and Jeong, Y.-H., "Estimator for Spacecraft Mass Property and Momentum Actuator Alignment under Influence of External Torque," *SpaceOps 2016 Conference*, 2016.

[20] Xu, Z., Qi, N., and Chen, Y., "Parameter estimation of a three-axis spacecraft simulator using recursive least-squares approach with tracking differentiator and Extended Kalman Filter," *Acta Astronautica*, vol. 117, Dec. 2015, pp. 254–262.

[21] Townsend, K., "Adafruit BNO055 Absolute Orientation Sensor," *Adafruit Learning System* Available: https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor.

[22] "Earth Centred Inertial Frame," *ADCS For Beginners* Available: https://adcsforbeginners.wordpress.com/tag/earth-centred-inertial-frame/

[23] Farissi, M. S., Carletta, S., Nascetti, A., and Teofilatto, P., "Implementation and Hardware-In-The-Loop Simulation of a Magnetic Detumbling and Pointing Control Based on Three-Axis Magnetometer Data," *Aerospace*, Dec. 2019

[24] "Euler angles," *Euler angles - Knowino* Available: https://www.tau.ac.il/~tsirel/dump/Static/knowino.org/wiki/Euler_angles.html

[25] "Pendulum," *Wikipedia* Available: https://en.wikipedia.org/wiki/Pendulum

[26] "Inverted pendulum," *Wikipedia* Available: https://en.wikipedia.org/wiki/Inverted_pendulum

[27] Brunton, S. L., Data Driven Science & Engineering, Cambridge University Press, 2017.

APPENDICES

A. Derivations

A.1 Kalman Filter

The mathematical representation of the noise in the system is represented in Eq. (A.1.1a) & (A.1.1b).

$$\dot{\vec{x}} = \boldsymbol{A}\vec{\mathrm{x}} + \mathbf{B}\vec{\mathrm{u}} + w_d \tag{A.1.1a}$$

$$y = \boldsymbol{C}\vec{\mathrm{x}} + \mathbf{D}\vec{\mathrm{u}} + w_n \tag{A.1.1b}$$

The process noise, or disturbance, is captured in the matrix $\mathbf{w}_d$ while the sensor noise is captured by the matrix $\mathbf{w}_n$. Within a standard Kalman Filter, any noise added to the system is assumed to be both Gaussian in nature and zero-mean. If noise is to be considered zero-mean, then the noise pattern must maintain a rolling average of the true signal. Types of Kalman Filters are available depending on if the noise skews in a particular direction or is unbalanced.

The covariance of the gaussian noise can be given by Eq (A.1.2).

$$\boldsymbol{Q} = E[\boldsymbol{w}_d \boldsymbol{w}_d^T] \tag{A.1.2a}$$

$$\mathbf{R} = \mathrm{E}[\boldsymbol{w}_n \boldsymbol{w}_n^T] \tag{A.1.2b}$$

Solving for the mean squared value results in the following set of equations with $\hat{\boldsymbol{x}}_k$ denoting the estimation of states:

$$\boldsymbol{P}_k = E[\boldsymbol{e}_k \boldsymbol{e}_k^T] = E[(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k)(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k)^T] \tag{A.1.3}$$

When estimating the current state the new guess can be made through the use of a Kalman gain, $\mathbf{K}_f$, and the application of matrix algebra where the guessing function looks like Eq. (A.1.4).

$$\hat{\boldsymbol{x}}_{k+1} = \hat{\boldsymbol{x}}_k + \boldsymbol{K}_k(\boldsymbol{y}_{k+1} - \hat{\boldsymbol{y}}_k) \tag{A.1.4}$$

Expand using Eq. (A.1.1b):

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_{k-1} + \boldsymbol{K}_k(\boldsymbol{y}_k - \boldsymbol{C}\hat{\boldsymbol{x}}_{k-1}) \tag{A.1.5}$$

The term inside the parenthesis can be called the measurement residual and redefined as:

$$\boldsymbol{m}_k = \boldsymbol{z}_k - \boldsymbol{C}\hat{\boldsymbol{x}}_{k-1} \tag{A.1.6}$$

Combine Eq. (A.1.1) with Eq. (A.1.5):

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_{k-1} + \boldsymbol{K}_k(\boldsymbol{C}\boldsymbol{x}_k + \boldsymbol{w}_d - \boldsymbol{C}\hat{\boldsymbol{x}}_{k-1}) \tag{A,1.7}$$

Using Eq. (A.1.7) in Eq. (A.1.3):

$$\boldsymbol{P}_{\mathrm{k}} = \mathrm{E}\left[\left[(\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C})(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k-1}) - \boldsymbol{K}_{kw_n}\right] \quad [(\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C})(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k-1}) - \boldsymbol{K}_k\boldsymbol{w}_n]^T\right] \tag{A.1.8}$$

Simplifying Eq. (A.1.8) by isolating the error term:

$$\boldsymbol{P}_k = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C})\mathrm{E}[(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k-1})(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k-1})^T](\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C}) + \boldsymbol{K}_k\mathrm{E}[\boldsymbol{w}_n\boldsymbol{w}_n^T]\boldsymbol{K}_{k^T} \tag{A.1.9}$$

Simplify the Error functions according to Eq. (A.1.3) and Eq. (A.1.2b):

$$\boldsymbol{P}_k = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C})\boldsymbol{P}_{k-1}(\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C}) + \boldsymbol{K}_k\boldsymbol{R}\boldsymbol{K}_k^T \tag{A.1.10}$$

Expanding the previous equation:

$$\boldsymbol{P}_k = \boldsymbol{P}_{k-1} - \boldsymbol{K}_k\boldsymbol{C}\boldsymbol{P}_{k-1} - \boldsymbol{P}_{k-1}\boldsymbol{C}^T\boldsymbol{K}_k^T + \boldsymbol{K}_k(\boldsymbol{C}\boldsymbol{P}_{k-1}\boldsymbol{C}^T + \boldsymbol{R})\boldsymbol{K}_k^T \tag{A.1.11}$$

Using linear the linear algebra convention for transpose it can be said that the following relationship is true and that Eq. (A.1.11) simplifies into Eq. (A.1.12).

$$\boldsymbol{K}_k\boldsymbol{C}\boldsymbol{P}_{k-1} = \boldsymbol{P}_{k-1}\boldsymbol{C}^T\boldsymbol{K}_k^T$$

$$\boldsymbol{P}_k = \boldsymbol{P}_{k-1} - 2\boldsymbol{K}_k\boldsymbol{C}\boldsymbol{P}_{k-1} + \boldsymbol{K}_k(\boldsymbol{C}\boldsymbol{P}_{k-1}\boldsymbol{C}^T + \boldsymbol{R})\boldsymbol{K}_k^T \tag{A.1.12}$$

In order to minimize $\mathbf{P}_k$ with respect to $\mathbf{K}_k$ taking the derivative of Eq. (A.1.12) is necessary. The resulting equation becomes:

$$\frac{\boldsymbol{P}_k}{\boldsymbol{K}_k} = -2(\boldsymbol{C}\boldsymbol{P}_{k-1})^T + 2\boldsymbol{K}_k(\boldsymbol{C}\boldsymbol{P}_{k-1}\boldsymbol{H}^T + \boldsymbol{R}) \tag{A.1.13}$$

Setting Eq. (A.1.13) equal to zero results in:

$$(\boldsymbol{H}\boldsymbol{P}_{k-1})^T = \boldsymbol{K}_k(\boldsymbol{C}\boldsymbol{P}_{k-1}\boldsymbol{C}^T + \boldsymbol{R}) \tag{A.1.14}$$

Solving for the Kalman gain results in:

$$\boldsymbol{K}_k = \boldsymbol{P}_{k-1}\boldsymbol{C}^T(\boldsymbol{C}\boldsymbol{P}_{k-1}\boldsymbol{C}^T + \boldsymbol{R})^{-1} \tag{A.1.15}$$

Using the Kalman gain to simplify the equation for $\mathbf{P}_k$ leads to:

$$\boldsymbol{P}_k = \boldsymbol{P}_{k-1} - \boldsymbol{P}_{k-1}\boldsymbol{C}^T(\boldsymbol{C}\boldsymbol{P}_{k-1}\boldsymbol{C}^T + \boldsymbol{R})^{-1}\boldsymbol{C}\boldsymbol{P}_{k-1}$$

$$\boldsymbol{P}_k = \boldsymbol{P}_{k-1} - \boldsymbol{K}_k\boldsymbol{C}\boldsymbol{P}_{k-1}$$

$$\boldsymbol{P}_k = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C})\boldsymbol{P}_{k-1} \tag{A.1.16}$$

The future state can be predicted using the current estimate and the transition matrix **A:**

$$\hat{\boldsymbol{x}}_{k+1} = \boldsymbol{A}\hat{\boldsymbol{x}}_k \tag{A.1.17}$$

Solving for the error using the state estimate and the next set of measurements:

$$\boldsymbol{e}_{k+1} = \boldsymbol{x}_{k+1} - \hat{\boldsymbol{x}}_{k+1}$$

$$= (\boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{w}_d) - \boldsymbol{A}\hat{\boldsymbol{x}}_k$$

$$\boldsymbol{e}_{k+1} = \boldsymbol{A}\boldsymbol{e}_k + \boldsymbol{w}_k \tag{A.1.18}$$

Using Eq. (A.1.3) at time k+1 gives:

$$\boldsymbol{P}_{k+1} = E[(\boldsymbol{A}\boldsymbol{e}_k + \boldsymbol{w}_d)(\boldsymbol{A}\boldsymbol{e}_k + \boldsymbol{w}_d)^T] \tag{A.1.19}$$

Expanding results in:

$$\boldsymbol{P}_{k+1} = \mathrm{E}[(\boldsymbol{A}\boldsymbol{e}_k)(\boldsymbol{A}\boldsymbol{e}_k)^T] + \mathrm{E}[\mathbf{w}_\mathrm{k}\mathbf{w}_\mathrm{k}{}^T] \tag{A.1.20}$$

Simplifying leaves:

$$\boldsymbol{P}_{k+1} = \boldsymbol{A}\boldsymbol{P}_k\boldsymbol{A}^T + \boldsymbol{Q} \tag{A.1.21}$$

A.2 Extended Kalman Filter

The Kalman Filter is easy to implement when considering the algorithm shown in Section 6.4. The adaptation comes when considering that the C & A matrices need to be altered to conform with the new state of the system with each successive itteration. The application of this can be solved by linearizing the partial differentiation of the Equations of Motion at each time step. The algorithms partDiff.m and linPart.m, on pages 83 & 84 respectively, solve this issue for and can be implemented in the overall EKF algorithm. The algorithm will look as follows:

1. Solve for the partial differential of the equations of motion:

   [Apd,Bpd] = partDiff(fcn,store,point)

2. Linearize about the current states:

   [ALin,BLin] = linPoint(a,b,X,point)

3. Calculate the Kalman Gain:

$$\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1}\boldsymbol{C}_k^T\left(\boldsymbol{C}_k\boldsymbol{P}_{k|k-1}\boldsymbol{C}_k^T + \boldsymbol{R}_k\right)^{-1} \tag{A.2.1}$$

4. Update the State Estimate:

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k\left(\boldsymbol{y}_k - \boldsymbol{C}_k\hat{\boldsymbol{x}}_{k|k-1}\right) \tag{A.2.2}$$

5. Update the error Covariance in response to the Kalman Gain:

$$\boldsymbol{P}_{k|k} = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{C}_k)\boldsymbol{P}'_{k|k-1} \tag{A.2.3}$$

6. Project to next step using:

$$\widehat{\boldsymbol{x}}'_{k+1|k} = \boldsymbol{A}_k \widehat{\boldsymbol{x}}_{k|k} \tag{A.2.4}$$

$$\boldsymbol{P}_{k+1|k} = \boldsymbol{A}_k \boldsymbol{P}_{k|k} \boldsymbol{A}_k^T + \boldsymbol{Q}_k \tag{A.2.5}$$

7. Return to step 2

A.3 Least-Squares Estimation

The minimalization of the cost function in Eq. (5.3.1) is accomplished using a desired output sequence, **d**[n], and an input sequence, **x**[n], to create a filter or gain value **h**[n] that corrects for the error in the system. The relationship is shown in Eq. (A.3.1).

$$\boldsymbol{d}[n] = \sum_{i=0}^{n} (\boldsymbol{h}_i \boldsymbol{x}_{i-1}) + \boldsymbol{e}_i \tag{A.3.1}$$

Solving for the error:

$$\boldsymbol{d}[n] - \sum_{i=0}^{n} (\boldsymbol{h}_i \boldsymbol{x}_{i-1}) = \boldsymbol{e}_i \tag{A.3.2}$$

Rewriting this in pure vector notation leaves the following and simplifying the inputs into matrix **A**.

$$\boldsymbol{e} = \boldsymbol{d} - \boldsymbol{A}\boldsymbol{h} \tag{A.3.3}$$

Rewriting the cost function in Eq. (5.3.1b):

$$\boldsymbol{S} = \|\boldsymbol{d} - \boldsymbol{A}\boldsymbol{h}\|_2^2 \tag{A.3.4}$$

$$\boldsymbol{S} = (\boldsymbol{d} - \boldsymbol{A}\boldsymbol{h})^T (\boldsymbol{d} - \boldsymbol{A}\boldsymbol{h}) \tag{A.3.5}$$

Expanding Eq. (A.3.5) results in the following:

$$\boldsymbol{S} = \boldsymbol{d}^T\boldsymbol{d} - \boldsymbol{h}^T\boldsymbol{A}^T\boldsymbol{d} - \boldsymbol{d}^T\boldsymbol{A}\boldsymbol{h} + \boldsymbol{h}^T\boldsymbol{A}^T\boldsymbol{A}^T\boldsymbol{h} \tag{A.3.6}$$

Using matrix properties of $(AB)^T = B^T A^T$ the equation can be reduced to:

$$\boldsymbol{S} = \boldsymbol{d}^T\boldsymbol{d} - 2\boldsymbol{d}^T\boldsymbol{A}\boldsymbol{h} + \boldsymbol{A}^T\boldsymbol{A}\boldsymbol{h}^2 \tag{A.3.7}$$

The minimum of a function can be solved for by setting the derivative of said function to zero:

$$\frac{\delta \boldsymbol{S}}{\delta \boldsymbol{h}} = 0 = -2\boldsymbol{A}^T\boldsymbol{d} + 2\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{h} \tag{A.3.8}$$

Solving Eq. (A.3.8) for **h** or the optimal filter gain results in the following:

$$\boldsymbol{h}^* = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{d} \tag{A.3.9}$$

Solving for the filter in Eq. (A.3.9) can also be solved using the variance of the input data with respect to the covariance of the input and output data. This is accomplished in Eq. (A.3.10).

$$\hat{\beta} = \frac{Covariance(x_i, \mathrm{d}_i)}{Variance(x_i)} \tag{A.3.10}$$

Expanding Eq. (A.3.10) shows the math behind the covariance and variance functions.

$$\hat{\beta} = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(d_i - \bar{d})}{\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{A.3.11}$$

A.4 Recursive Least Squares Estimation

The recursive algorithm obtains the optimal filter $\mathbf{h}^*$ by beginning with measurements of the output, $\boldsymbol{z}_k$, where k represents the current iteration and the given inputs, $\boldsymbol{H}_k$. In the algorithm, the current error statistics are held by the error covariance matrix, $\boldsymbol{R}_k$, state residuals are held by a covariance matrix, $\boldsymbol{P}_k$, and the estimate filter represented by $\hat{\mathbf{x}}_\mathrm{k}$. The model equation is represented in Eq. (A.4.1), where **n** is the error value for the specified iteration.

The optimal future filter $\boldsymbol{h}_{n+1}^*$ is given by Eq. (A.4.1).

$$\boldsymbol{h}_{n+1}^* = (\boldsymbol{A}_{n+1}^T\boldsymbol{A}_{n+1})^{-1}\boldsymbol{A}_{n+1}^T\boldsymbol{d}_{n+1} \tag{A.4.1}$$

Combining all the inputs into one row vector:

$$\boldsymbol{u}_i = \begin{bmatrix} x_i & x_{i-1} \dots x_{i-(m-1)} \end{bmatrix} \tag{A.4.2}$$

Eq. (A.4.1) can be rewritten using Eq. (A.4.2) and the following terms:

$$\boldsymbol{\phi}_n = \sum_{i=0}^{n} \boldsymbol{u}_i \, \boldsymbol{u}_i^T \tag{A.4.3}$$

$$\mathbf{z}_n = \sum_{i=0}^{n} \boldsymbol{u}_i \, \boldsymbol{d}[i] \tag{A.4.4}$$

$$\boldsymbol{h}^* = \boldsymbol{\phi}_n^{-1} \boldsymbol{z}_n \tag{A.4.5}$$

$\boldsymbol{\phi}_n$ can be expanded to form the following expression:

$$\boldsymbol{\phi}_n = \sum_{i=0}^{n-1} \boldsymbol{u}_i \, \boldsymbol{u}_i^T + \boldsymbol{u}_n \boldsymbol{u}_n^T \tag{A.4.6}$$

Rewriting by substituting in $\boldsymbol{\phi}_{n-1}$ and adding an Identity matrix of size *m*:

$$\boldsymbol{\phi}_n = \boldsymbol{\phi}_{n-1} + \boldsymbol{u}_n \boldsymbol{I} \boldsymbol{u}_n^T \tag{A.4.7}$$

The Matrix Inversion Lemma says that Eq. (A.4.9) is the inverse of Eq. (A.4.8).

$$\boldsymbol{A} = \boldsymbol{B}^{-1} + \boldsymbol{C}\boldsymbol{D}^{-1}\boldsymbol{C}^T \tag{A.4.8}$$

$$\boldsymbol{A}^{-1} = \boldsymbol{B} - \boldsymbol{B}\boldsymbol{C}(\boldsymbol{D} + \boldsymbol{C}^T\boldsymbol{B}\boldsymbol{C})^{-1}\boldsymbol{C}^T\boldsymbol{B} \tag{A.4.9}$$

Expand Eq. (A.4.7) using the Matrix Inversion Lemma:

$$\boldsymbol{\phi}_n^{-1} = \boldsymbol{\phi}_{n-1}^{-1} - \frac{\boldsymbol{\phi}_{n-1}^{-1} \boldsymbol{u}_n \boldsymbol{u}_n^T \boldsymbol{\phi}_{n-1}^{-1}}{\boldsymbol{I} + \boldsymbol{u}_n^T \boldsymbol{\phi}_{n-1}^{-1} \boldsymbol{u}_n} \tag{A.4.10}$$

Saying $\boldsymbol{P}_n = \boldsymbol{\phi}_n^{-1}$ and $\boldsymbol{P}_{n-1} = \boldsymbol{\phi}_{n-1}^{-1}$ and simplifying Eq. (A.4.10) with a gain vector:

$$\boldsymbol{P}_n = \boldsymbol{P}_{n-1} - \boldsymbol{K}_n \boldsymbol{u}_n^T \boldsymbol{P}_{n-1} \tag{A.4.11}$$

Writing out $\mathbf{K}_n$ results in:

$$\boldsymbol{K}_n = \frac{\boldsymbol{P}_{n-1}\boldsymbol{u}_n}{\boldsymbol{I}+\boldsymbol{u}_n^T\boldsymbol{P}_{n-1}\boldsymbol{u}_n} \tag{A.4.12a}$$

$$\boldsymbol{K}_n = (\boldsymbol{P}_{n-1} - \boldsymbol{K}_n\boldsymbol{u}_n^T\boldsymbol{P}_{n-1})\boldsymbol{u}_n = \boldsymbol{P}_n\boldsymbol{u}_n \tag{A.4.12b}$$

Similarly, the operation performed for Eq. (A.4.6) and Eq. (A.4.7), $\mathbf{z}_{n-1}$ can be separated out from $\mathbf{z}_n$ resulting in a new equation:

$$\boldsymbol{z}_n = \boldsymbol{z}_{n-1} + \boldsymbol{u}_n\boldsymbol{d}[n] \tag{A.4.13}$$

Plugging Eq. (A.4.13) into Eq. (A.4.5):

$$\boldsymbol{h}_n^* = \boldsymbol{P}_n\boldsymbol{z}_n$$

$$\boldsymbol{h}_n^* = \boldsymbol{P}_n(\boldsymbol{z}_{n-1} + \boldsymbol{u}_n\boldsymbol{d}[n]) \tag{A.4.14}$$

Expand the first term in Eq. (A.4.14) using Eq. (A.4.11) in order to add the previous estimate to the current estimate error as seen in Eq. (A.4.17).

$$\boldsymbol{h}_n^* = (\boldsymbol{P}_{n-1} - \boldsymbol{K}_n\boldsymbol{u}_n^T\boldsymbol{P}_{n-1})\boldsymbol{z}_{n-1} + \boldsymbol{P}_n\boldsymbol{u}_n\boldsymbol{d}[n] \tag{A.4.15}$$

Distribute the $\mathbf{z}_{n-1}$ and expand:

$$\boldsymbol{h}_n^* = \boldsymbol{P}_{n-1}\boldsymbol{z}_{n-1} - \boldsymbol{K}_n\boldsymbol{u}_n^T\boldsymbol{P}_{n-1}\boldsymbol{z}_{n-1} + \boldsymbol{P}_n\boldsymbol{u}_n\boldsymbol{d}[n] \tag{A.4.16}$$

Use Eq. (A.4.5) and Eq. (A.4.12b) to simplify the equation:

$$\boldsymbol{h}_n^* = \boldsymbol{h}_{n-1}^* - \boldsymbol{K}_n\boldsymbol{u}_n^T\boldsymbol{h}_{n-1}^* + \boldsymbol{K}_n\boldsymbol{d}[n] \tag{A.4.17}$$

Factor out $\mathbf{K}_n$ from the right-hand section of the equation:

$$\boldsymbol{h}_n^* = \boldsymbol{h}_{n-1}^* - \boldsymbol{K}_n(\boldsymbol{u}_n^T\boldsymbol{h}_{n-1}^* - \boldsymbol{d}[n]) \tag{A.4.18}$$

The section inside the parenthesis is the error of the algorithm. Use a new term, ζ, to describe the error and simplify Eq. (A.4.18).

$$\boldsymbol{\zeta} = \boldsymbol{d}[n] - \boldsymbol{u}_n^T \boldsymbol{h}_{n-1}^* \tag{A.4.19}$$

$$\boldsymbol{h}_n^* = \boldsymbol{h}_{n-1}^* + \boldsymbol{K}_n \boldsymbol{\zeta} \tag{A.4.20}$$

B. MATLAB Code Files

B.1 Recursive Least Squares for Cassini

```
%% EOM Calculations %%
% ================== %
clc, close all, clear all

% Set initial angular velocity vector
w0 = [0
   0
   0];

% Declare Cassini Inertia Matrix
J = [8810.8 -136.8 115.3;
   -136.8 8157.3 156.4;
   115.3 156.4 4721.8];

dt = 0.01;              % Set timestep for simulation
t1 = [1:dt:150];        % Time range for torque input
t2 = [150+dt:dt:200];   % Time range for zero torque

tau1 = [10              % 10Nm torque about x
       2
       4];
tau2 = [0               % Zero torque input
       0
       0];


% set max timestep to the same as Simulink
options = odeset('MaxStep',1/50);

% ODE45 is used to simulate the differential equation
% represented by rotationalEOM.m
```

```
[T1,om1] = ode45(@rotationalEOM,t1,w0,options,tau1,J)
[T2,om2] = ode45(@rotationalEOM,t2,om1(size(om1,1),:),options,tau2,J)

% Combine the data from the datasets with different torques to create one
% continuous dataset
omega = [om1
   om2];
t = [T1
   T2];

% Calculate rotational inertia and the realized input moment from the omega
% output from ODE45.
Jinv = inv(J);
for i = 1:size(omega,1)
   alpha(i,:) = Jinv*(tau1-cross(omega(i,:),omega(i,:)*J)');
   moment(i,:) = alpha(i,:)*J;
end

%% Calculate J using RLS based on Jguess %%
% ========================================= %

lamda = 0.95;
Linv = 1/lamda;

Jguess = [6000     0   0;
          0   4000    0;
          0   0   2000];

Px = eye(3);
hx = Jguess(:,1);
for i = 1:size(omega,1)

   [hx,errx,Px] = RLSStep(alpha(i,:)',moment(i,1),Px,hx,Linv);
```

```
    erx(:,i) = abs(errx);
end


Py = eye(3);
hy = Jguess(:,2);
for i = 1:size(omega,1)

    [hy,erry,Py] = RLSStep(alpha(i,:)',moment(i,2),Py,hy,Linv);
    ery(:,i) = abs(erry);
end


Pz = eye(3);
hz = Jguess(:,3);
for i = 1:size(omega,1)

    [hz,errz,Pz] = RLSStep(alpha(i,:)',moment(i,3),Pz,hz,Linv);
    er(:,i) = abs(err);
end

J_RLS = [hx hy hz]
```

B.2 Rotational Equations of Motion

```
function dydt = rotationalEOM(t,omega,torque,inertia)

% Function to solve for the rotational motion of an object

% t        - time array                        [s]
% omega    - angular velocity                  [rad/s]
% torque   - input torque to the system        [Nm]
% inertia  - [3x3] inertia matrix of the system    [kg*m^2]
```

```
dydt = inv(inertia)*(torque-cross(omega,inertia*omega));
```

B.3 Recursive Least Squares Step

```
function [h,err,P] = RLSStep(u,d,P,h,Linv)

% Function for 1 step of an RLS algorithm
%
% ==== input ====
% u - the input matrix nx1
% b - the output matrix nx1
% P - the inverse matrix
% h - the filter matrix
% L - lamda or forgetting factor 0<lamda<1
%
% ==== output ====
% K - gain vector
% err - error value

K = (P*u.*Linv)/(eye(size(d,2))+Linv.*u'*P*u);
err = d-u'*h;
h = h+K*err;
P = Linv.*(P-K*u'*P);
end
```

B.4 Kalman Filter

```
function [xn,P,K] = KalmanFilter(Alin,Blin,x,Yk,C,P,Qk,Rk)

% Continuous Time Kalman Filter Function
%
% Alin  - Linearized A Matrix
```

```
% Blin  - Linearized B Matrix
% x     - incomming states
% C     - measurement matrix
% P     - previous process matrix
% Qk & Rk - noise covariances

Pkp = Alin*P*Alin'+Qk;            % calculate the prediction matrix
Pkn = Pkp*C';

K = Pkp*C' ./ (C*Pkp*C' + Rk);      % calculate Kalman Gain
Yk = C*x;                         % current observation
xn = x + K*(Yk - C*x);            % estimate state matrix
P = (eye(size(Alin,1)) - K*C)*Pkp;  % update process covariance matrix
```

B.5 Extended Kalman Filter

```
function [X,P,K] = ExtendedKalmanFilter(Apd,Bpd,X_var,x,Yk,C,P,Qk,Rk)

% Continuous Time Kalman Filter Function
%
[Alin,Blin] = linPoint(Apd,Bpd,X_var,x');




Yn = Alin*C + sqrt(Rk);

P = Alin*P*Alin'+Qk;
Pn = P*C';            % calculate the prediction matrix
Y = xn*C;             % estimated output
K = Pn*inv(H*Pn+Rk);    % Kalman Gain

x = xn+K*(Yk-Y);        % state update
P = P-K*Pn';          % covariance update
```

B.6 Partial Differentiation

```
function [Apd,Bpd] = partDiff(fcn,store,point)

% Create the A and B matrices for a system of governing equations

% fcn is all of the equations of motion in a [n x 1] matrix
% store is all of the symbolic variables in the EoM in a [m x 1] matrix

a = size(fcn,1);        % pull the number of functions
b = size(store,2);      % pull the number of variables

% b should usually be larger than a by the numver of inputs

i=1;j=1;
for i = 1:b

    der(:,i) = diff(fcn(:,1),store(i));          % partial diff of all fcn with respect to the store of symbols

    if i <= a                   % once we move past 'a' number of partial diffs we are in the inputs
        Apd(:,i) = der(:,i);        % storing current differentiation for A

    else
        Bpd(:,j) = der(:,i);        % store the rest of the determination as B
        j=j+1;
    end

    i=i+1;
end
```

B.7 Linearization of Differential

```
function [ALin,BLin] = linPoint(a,b,X,point)

% Linearize a system of equations already in their A & B matrices about a
% specific point.
%
% 'a' - symbolic A matrix created using partial differentials
%
% 'b' - symbolic B matrix created using partial differentials
%
% 'X' - storage location of the symbolic variables of the state matrix.
% Ex: X = [x v theta omega]
%
% 'point' - The values of the state variables that are to be linearized
% about. These variables must be in the same order as the state variables
% they are associated with.
%

ALin = double(subs(a,X,point));      % substitute points to be linearized about
BLin = double(subs(b,X,point));      % substitute points to be linearized about
```

B.8 Kalman Filter Code from Brunton & Kutz [4]

```
%% Matlab Function lqe Kalman Gain Determination
%
% This filter will be based around the linear pendulum down
% position.
%

clc, close all, clear all



% ========== Setup ==========
```

```
M = 5;
m = 1;
L = 2;
g = -9.81;
d = 1;

P = -1;         % Pendulum up = 1

A = [0 1 0 0;
   0 -d/M -m*g/M 0;
   0 0 0 1;
   0 -P*d/(M*L) -P*(m+M)*g/(M*L) 0];

B = [0 1/M 0 P/(M*L)]';

C = [1 0 0 0];

D = zeros(1,1);



% ========== Defining Kalman Filter Parameters ==========

% In the typical inverted pendulum model we can expect that % the system will not experience any major disturbances,
% such as physically being bumped
% into, so the 'vd' variable will be kept low when compared % to the 'vn' variable.

vd = 0.1*eye(4);        % System "trustworthyness"
vn = 1;             % Sensor "trustworthyness"

BK = [B vd 0*B];    % adding distrubance to the inputs

sysXMeasured = ss(A,BK,C,[0 0 0 0 0 vn]);       % State Space
```

```
    sysAllMeasured = ss(A,BK,eye(4),zeros(size(BK,1),size(BK,2)));       % State Space for TRUE
system

    [K,P,E] = lqe(A,vd,C,vd,vn);    % Kalman Filter Gain determination

    sysKalmanFilter = ss(A-K*C,[B K],eye(4),0*[B K]);         % Kalman filter State Space



    % ========== Estimate System in Down Position ==========

    dt = 0.01;
    t = 0:dt:60;

    disturbance = randn(4,size(t,2));
    noise = randn(size(t));                % this could be simplified to randn(size(t)) but I would prefer
to have easy access to the strength of the noise in the first position

    input = 0*t;
    input(100:120) = 100;
    input(1500:1520) = -100;

    input_total = [input; vd*vd*disturbance; noise];

    [y,t] = lsim(sysXMeasured,input_total,t);
```

B.9 Monte Carlo Example

```
%% Monte Carlo Simulation

clc, close all, clear all

M = 5;
m = 1;
L = 2;
g = -9.81;
d = 1;

P = -1;        % Pendulum up = 1

A = [0 1 0 0;
   0 -d/M -m*g/M 0;
   0 0 0 1;
   0 -P*d/(M*L) -P*(m+M)*g/(M*L) 0];

B = [0 1/M 0 P/(M*L)]';

C = [1 0 0 0];

D = zeros(1,1);

vd = 0*eye(4);        % System "trustworthyness"
vd2 = vd*vd;
vn = 1;           % Sensor "trustworthyness"

BK = [B vd 0*B];    % adding distrubance to the inputs

IC = [0;0;0;0];

dt = 0.01;
time = 0:dt:60;

imp1 = 10;
imp2 = 8;

R = 1;
%%
figure
hold on
tic
for i = 1:100
   vn(i) = rand*5;
   sim('SIM_MC_003',time);
```

```
    errorx(:,i) = (ans.TrueXhat(:,1)-ans.NoisyXhat(:,1));
    errorv(:,i) = (ans.TrueXhat(:,2)-ans.NoisyXhat(:,2));
    errortheta(:,i) = (ans.TrueXhat(:,3)-ans.NoisyXhat(:,3));
    erroromega(:,i) = (ans.TrueXhat(:,4)-ans.NoisyXhat(:,4));

    NoisyStates(:,:,i) = ans.NoisyXhat;

    i
end
toc
TrueStates = ans.TrueXhat;

%% Error Values

for i = 1:100
    Merrorx(i) = mean(errorx(:,i));
    Merrorv(i) = mean(errorv(:,i));
    Merrortheta(i) = mean(errortheta(:,i));
    Merroromega(i) = mean(erroromega(:,i));

    stdx(i) = std(errorx(:,i))/sqrt(length(errorx(:,i)));
    stdv(i) = std(errorv(:,i))/sqrt(length(errorv(:,i)));
    stdtheta(i) = std(errortheta(:,i))/sqrt(length(errortheta(:,i)));
    stdomega(i) = std(erroromega(:,i))/sqrt(length(erroromega(:,i)));
end

Data = [vn' Merrorx' stdx' Merrorv' stdv' Merrortheta' stdtheta' Merroromega' stdomega'];
varnames = {'Noise','Mean X','std X','Mean V','std V','Mean \theta','std \theta','Mean \omega','std
\omega'};
Error =
table(Data(:,1),Data(:,2),Data(:,3),Data(:,4),Data(:,5),Data(:,6),Data(:,7),Data(:,8),Data(:,9),'Vari
ableNames',varnames)

%% Plots
figure
hold on
for i=1:100
    plot(time,NoisyStates(:,1,i),'b')
end
plot(time,TrueStates(:,1),'r','linewidth',2)
hold off
title('Sensor Noise Effects on Position')
ylabel('Position of Cart [m]')
xlabel('Time [s]')
%%
figure
```

```
hold on
for i=1:100
   plot(time,NoisyStates(:,3,i),'b')
end
plot(time,TrueStates(:,3),'r','linewidth',2)
hold off
title('Estimated Angle(s) vs True Angle')
ylabel('Angle of Pendulum [rad]')
xlabel('Time [s]')
```

B.10 Radius to CoM_b1

```
function rb1 = fcn(ra1,ra2,rcm_change)
dra = norm(ra1-ra2);
drb = -rcm_change+dra;
rb1 = ra1*drb/dra;
```

B.11 Unknown Mass Calculation

```
function mb_calc = fcn(ma,ra,rb)

mb_calc = ma*norm(ra)/norm(rb);
```

B.12 Unknown Inertia Calculation

```
function Jb_calc  = fcn(Ja,Jtotal,ma,ra_calc,mb_calc,rb_Calc)

Ra = ra_calc*ra_calc';
Rb = rb_Calc*rb_Calc';

Jb_calc = Jtotal-Ja-ma*Ra-mb_calc*Rb;
```

B.13 Parameter Isolation MATLAB

```
%%
clc, close all, clear all

%% Constants and Calculations of Values to be used
J_cassini = [8810.8 -136.8 115.3;
   -136.8 8157.3 156.4;
   115.3 156.4 4721.8];       % Cassini Inertia Matrix

Ja = J_cassini;

J_rwa_spin = 0.161;      % cassini reaction wheel inertia 10.2514/6.2005-6271
ma = 2100;
```

```
mrwa = 5.152;              % assuming the radius of the rwa's are 0.25m
rrwa = 0.25;               % arbitrarily assigned
lrwa = 0.1;                % arbitrarily assigned

J_rwa_offspin = 0.25*mrwa*rrwa^2+1/12*mrwa*lrwa^2;  % calculate spin about y and z axis

J_rwa = diag([J_rwa_spin;J_rwa_offspin;J_rwa_offspin]);


Jb = [2000  0   0;
   0   2000    0;
   0   0   2000];             % object inertia matrix

Jxguess = [6000     0   0];
Jyguess = [0    4000    0];
Jzguess = [0    0   2000];

mb = 500;

torque_input = 10;      % Torque applied
timeUp = 0;
timeDown = 60;

% ==== distance from CoM_A to CoM_B ====
rcm1 = [8;
        4;
        sqrt(20)];
rcm_change = 4;
rcm2 = (norm(rcm1)-rcm_change)*rcm1/norm(rcm1);

ra1 = -mb.*rcm1./(ma+mb);
rb1 = ra1+rcm1;

ra2 = -mb.*rcm2./(ma+mb);
rb2 = ra2+rcm2;

r_BNO = [0.5
   0.3
   0.25];

r_tBNO = ra1+r_BNO;
r_tBNO2 = ra2+r_BNO;

% ==== Total Inertia Matrix ====
J_calc = Ja+ma.*ra1*ra1'+Jb+mb.*rb1*rb1';
```

```
J_calc2 = Ja+ma.*ra2*ra2'+Jb+mb.*rb2*rb2';

%% Simulation and Initialization
dt = 0.1;
t = [0:dt:200];
tstep = 10;

% ==================
%    Covariance
% ==================
P = eye(3);
paw = 1e4;
Px = paw*eye(3);
Py = paw*eye(3);
Pz = paw*eye(3);

% ==================
%   Noise for Sim
% ==================

aN = 0;       % alpha order of mag
tN = 0;       % torque order of mag
omN = 0;      % omega order of mag
% open_system('SIM_MST_003')   Run if opening Simulink is desired
sim('SIM_MST_003_Images',t)

%% Outputs


t = ans.tout;
ac = ans.a_cent;
ac2 = ans.a_cent2;
alpha = ans.alpha;
alpha_Noise = ans.alpha_Noise;
omega = ans.omega;
omega_Noise = ans.omega_Noise;
eulang = ans.eulang;
r_aCalc = ans.r_aCalc;
r_aCalc2 = ans.r_aCalc2;
r_tBNO_calc = ans.r_tBNO_calc;
r_tBNO_calc2 = ans.r_tBNO_calc2;

r_bCalc1 = ans.r_bCalc1;
mb_calc = ans.mb_calc;

moments = ans.Moments;
```

```
momentsNC = ans.MomentsNoCorrection;
Torque_Noise = ans.Torque_Noise;


% ===============
%   X Variables
% ===============

Hx = ans.Hx;
Errorx = ans.Errorx;
Covx = ans.Covx;

% ===============
%   Y Variables
% ===============

Hy = ans.Hy;
Errory = ans.Errory;
Covy = ans.Covy;

% ===============
%   Z Variables
% ===============

Hz = ans.Hz;
Errorz = ans.Errorz;
Covz = ans.Covz;
```

B.14 Noise Generation

```
function [omega_Noise,Torque_Noise,alpha_Noise,acent_Noise] =
fcn(omega,Torque,alpha,acent,omN,tN,aN,acN)

omega_Noise = omega+randn(3,1)/3.*omN;
Torque_Noise = Torque+randn(3,1)/3.*tN;
alpha_Noise = alpha+randn(3,1)/3.*aN;
acent_Noise = acent+randn(3,1)/3.*acN;
```

B.15 Loop Code for Noise Iterations

```
%% Initialize
dt = 0.1;
```

```
t = [0:dt:180];
tstep = 10;

% ==================
%    Covariance
% ==================
P = eye(3);
paw = 1e4;
Px = paw*eye(3);
Py = paw*eye(3);
Pz = paw*eye(3);


iterations = 10;



%% Sim
% ==================
%   Noise for Sim
% ==================
tic
for i = 1:iterations

      aN = 0;            % alpha order of magnitude
      tN = 0;            % torque order of magnitude
      omN = 10^(-(3.8+rand*2));   % omega noise order of magnitude
      acN = 10^(-(3.0+rand*1.5)); % centripetal acceleration noise OoM

      sim('SIM_MST_103',t);
      Jest=ans.estimated_I;
      Jtotal_calc(i,:,:) = Jest(size(Jest,1)-2:size(Jest,1),:);
      mb_calc(i,:) = ans.mb_calc;
      r_bCalc1(i,:,:) = ans.r_bCalc1;
      JTerr_avg(i) = abs(ans.Jerr_avg(size(t,2)));
      Jb_calc(i,:,:) = ans.Jb_calc(:,:,size(t,2));
      Jberr_avg(i,:) = [acN,omN,ans.JBerr_avg(size(t,2))];


      mb_error(i,:) = [acN,omN,ans.mb_error(size(t,2))];
      raErr_avg(i,:) = [acN,omN,ans.raErr_avg(size(t,2))];
      JTotal_Avg(i,:)=[acN,omN,ans.Jerr_avg(size(t,2))];

   i
end
toc
```

B.16 Parameter Identification Simulink File