# Parameter Estimation of a Spacecraft Simulator Using Parameter-Adaptive Control

**Sam Wright**
**May 10, 2006**
**Aerospace and Ocean Engineering**
**Virginia Polytechnic and State University**

## Abstract

The focus of this paper is on Parameter-Adaptive Filtering (PAF) as it applies to locating the center of mass of a spacecraft simulator. The model is of a rigid body with an attached point mass used to excite the system. The equations of motion are developed for the model. The technique of PAF is introduced mathematically and an algorithm for implementation is provided. Three configurations are considered along with variations in the uncertainty and noise levels of the sensor measurements.

# Table of Contents

## List of Parameters

| | |
|---|---|
| $\vec{\omega}^{bi}$ | -angular velocity of the body frame with respect to the inertial frame |
| $\vec{v}_o$ | -velocity of the point O |
| $m_b$ | -mass of the body |
| $\vec{c}_b$ | -1st moment of inertia of the body |
| $\vec{r}_p$ | -position vector from O to the mass particle |
| $\vec{b}$ | -nominal location of the mass particle (x=0) |
| $\hat{n}$ | -direction of travel of the mass particle |
| $\delta$ | -displacement from the nominal position of the mass particle |
| $\underline{\underline{J}}$ | -2nd moment of inertia of the system |
| $\vec{h}^{o}$ | -angular momentum of the system about point O |
| $\underline{\underline{J}}_b$ | -2nd moment of inertia of the body |
| $\underline{\underline{J}}_p$ | -2nd moment of inertia of the point mass |
| $\vec{h}_b$ | -angular momentum of the body |
| $\vec{h}_p$ | -angular momentum of the mass particle |
| $\vec{f}_{ext}$ | -external forces applied to the system |
| $\vec{g}_{ext}$ | -external torques applied to the system |
| $m_p$ | -mass of the particle |
| g | -gravitational constant |
| $\underline{\theta}$ | -vector of Euler angles |
| $\mathbf{y}_{acc}$ | -output acceleration vector |
| $\mathbf{y}_{rg}$ | -output angular velocity vector |
| $\mathbf{x}$ | -state vector, $\mathbf{x} = [\underline{\theta} \quad \underline{\omega} \quad \delta]^T$ |
| $\mathbf{y}$ | -output vector, $\mathbf{y} = [\underline{\theta} \quad \underline{\omega} \quad \mathbf{y}_{acc} \quad \delta]$ |
| $\mathbf{S}^{-1}$ | -matrix relating angular velocity and angular rates |
| $\underline{f}_b$ | -force due to gravity of the body |
| $\underline{f}_p$ | -force due to gravity of the point mass |
| $\mathbf{p}$ | -unknown parameter vector |
| $\mathbf{G}$ | -observer gain |
| $\mathbf{A}_A$ | -linearized augmented input matrix |
| $\mathbf{C}_A$ | -linearized augmented output matrix |
| $\mathbf{V}$ | -noise spectral density matrix |
| $\mathbf{W}$ | -disturbance spectral density matrix |
| $\mathbf{P}$ | -covariance matrix |
| $\hat{\mathbf{x}}_A$ | -state estimate |
| $\mathbf{I}$ | -identity matrix |
| $\mathbf{R}^{bi}$ | -rotation matrix from the inertial frame to the body frame |
| $\mathbf{r}_y$ | -vector from point *O* to the rate gyro |

## Background Information

Prior research by the author led to a literature review on the topic of mass property estimation. The papers discussed below are concerned with on-orbit estimation algorithms of the dynamic parameters of a spacecraft. Dynamic parameters include the moments of inertia, thruster parameters, mass, etc.[1] The most important role of dynamic parameters is in the backup attitude control system which may be used in case of primary sensor failure.

The attitude control subsystem is important for the success of a spacecraft mission. Many of the other subsystems are dependent upon the accuracy of the orientation of the spacecraft. The thermal and power budgets are heavily dependent upon the trajectory and orientation. Precise control of the attitude is also necessary for scientific objectives. Continued success of a mission even after primary sensor failure relies on accurate knowledge of the parameters in the equations of motion. Further motivation for on-orbit estimation of parameters is the increasing complexity of spacecraft. The space station receives additional modules, which change the mass properties. Also, an extra-vehicular activity astronaut maneuvering while manipulating massive components requires knowledge of the inertia matrix.[3]

Many different models and algorithms have been developed to solve the problem of parameter estimation. Each solution has a unique method, but most employ the use of rate and attitude sensors. A conservation of energy method is used by Tanygin and Williams[3] to relate the inertia and angular velocity during a maneuver. Lee and Wertz[4] use the principal of conservation of angular momentum to determine the inertia tensor of the Cassini spacecraft. Once the dynamic equations are solved, a numerical technique is often necessary to optimize the answer. Extended Kalman filters, batch least squares estimation, or other cost function minimization techniques are used to estimate parameters.[1,5,8]

A paper by Mark Psiaki[2] presents an approach based on satisfying Euler's equations. The unknowns are constituted by the parameters being estimated while the attitude and rates are known. Linear and non-linear optimization techniques are used in conjunction with each other to yield the best guess at the parameters. Recursive linear techniques are used to obtain the estimates of the error. The iterative non-linear method is used to estimate the parameters. The physical model of the system includes gravity gradient torques, magnetic dipole moments, and an inertial impulse due to unmodeled torques. Psiaki's model is more complete than the others, but along with the completeness comes complexity.

A simpler approach is taken by Tanygin and Williams.[3] The equations of motion are used as the physical model.

$$I\dot{\omega} + \omega \times I\omega = M + \sum_i \rho_i \times F_i$$

Motion is excited by applying external torques, *M*, and forces, *F*. The equation must be manipulated into a form suitable for a batch least squares estimation. The standard form consists of a regressor matrix, Φ, a parameter vector, θ, and the output vector, Y.

$$\Phi\theta = Y$$

The time rate of change of rotational kinetic energy involves the inertia matrix and is used to convert the equations of motion into the necessary form. A vector of the unknown parameters, inertia matrix and location of the center of mass, is formed. A batch least squares estimation is used to minimize the quadratic cost function based on all the measurements available.  Noise and disturbances are not estimated in this method resulting in a simpler, less reliable result.  An alternative cost function is suggested by Clemen, where $y$ is the measured value, $\hat{y}$ is the simulated value, and $\bar{y}$ is the mean of the measured values.[1]

$$J = \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y}_i)^2}$$

An example of actual parameter estimation was done on-board the Cassini spacecraft.  The inertia tensor is used by the attitude-control fault protection algorithms, attitude estimator, thruster vector control algorithms, and the reaction wheel actuator.[2] Prior to launch an estimation of the inertia tensor was made.  The inertia of each component was determined and the location of the system center of mass was estimated. The overall system inertia tensor was found to be $\begin{bmatrix} 8810.8 & -136.8 & 115.3 \\ -136.8 & 8157.3 & 156.4 \\ 115.3 & 156.4 & 4721.8 \end{bmatrix}$ kg-m$^2$.[2]

An on-board algorithm involving the conservation of angular momentum was used to make further estimations of the inertia tensor.  A maneuver consisting of a Y-axis slew, followed by a X-axis slew, another Y-axis slew, a Z-axis slew, and finally another Y-axis slew was performed.  The angular rates, reaction wheel spin rates, quaternions, reaction wheel inertias, and location of the reaction wheels are all assumed to be accurately known.  Equating the initial angular momentum to the current angular momentum at each time step provides an explicit equation for the inertia tensor.  The angular momentum is conserved by neglecting the effects of external torques for the duration of the maneuver.

The resulting estimate for the inertia tensor was $\begin{bmatrix} 8655.2 & -144 & 132.1 \\ -144 & 7922.7 & 192.1 \\ 132.1 & 192.1 & 4586.2 \end{bmatrix}$ kg-m$^2$.[2]

Many of the algorithms are developed for space missions; however, the application to spacecraft simulators is of interest. Spacecraft simulators are typically used by small satellite builders. Small satellite programs are limited in time, financial resources, and manpower resources.[6] Simulators are used to test the attitude determination and control system before sending the satellite into orbit. The size and shape of spacecraft simulators limit the possibilities of directly determining the mass properties. Therefore, similar methods as those used for on-board parameter estimation must be used on simulators. The major difference between on-board and simulator estimation is the gravity effect due to the inevitable misalignment of the center of mass with the center of rotation.[6] An algorithm based solely on rate gyro measurements was developed by Kim and Lee[6]. The conservation of angular momentum is used in the same way as the method developed by Lee and Wertz. However, the gravity effect must be accounted for since it is an external torque. A function is developed to allow for the external torque, $T_g$.

$$f = h - h_0 + \int (\omega \times h - T_g)$$

Imbedded within the torque are the Euler angles between the body frame and inertial frame. If attitude sensors are not available, the rate data must be integrated to yield values for the angles. The numerical integration to find the Euler angles and $f$ leads to erroneous results.[6]

Another method commonly used is parameter-adaptive filtering. Here, a Kalman filter is used on a linear model for the system. The approach is to augment the system variables to include the unknown parameters as states. The unknown parameters are then estimated using measurements from sensors. One drawback is that there are no guarantees of performance with this method.[9] The rule of thumb, given by Stengel[9], is that the number of unknown parameters should not exceed the number of states in the original system.

Techniques have been developed for on-orbit parameter estimation. There are advantages and disadvantages associated with each algorithm. The complete models are able to provide better estimates, but at the cost of more computation time. Computation time onboard spacecraft is typically limited. The simpler algorithms are able to provide useful data if information such as the attitude *and* rates are known. The technique chosen for is unique for each spacecraft and should be based upon the available sensors, computation time, and accuracy needed.

# 1. Introduction

Spacecraft simulators are used as tools in the design process of satellites as well as in education. Simulators are used to test attitude determination and control algorithms. Knowledge of mass properties such as the inertia matrix and location of the center of mass are necessary when testing an algorithm. The properties are always present as parameters in the equations of motion. The size and shape of the simulators often prohibit direct measurement of these properties. Determination of these properties by another means is therefore necessary.

Parameter-Adaptive Filtering (PAF) provides estimates of unknown mass properties through the use of an extended Kalman filter. This paper implements PAF in order to estimate the location of the center of mass of a modeled spacecraft simulator. The equations of motion for a rigid body with an attached point mass are developed. The model includes the force of gravity along with the associated torques. The PAF algorithm is then presented along with the implementation for this particular model. Results are given for the actuated mass being used as the excitation. Results for three nominal configurations are compared.

# 2. Equations of Motion

Included in this section is the development of the necessary equations of motion for the spacecraft simulator to be modeled. First, the attitude representation is considered. The appropriate kinematic equations are found. The derivation of the dynamic equations follow, namely the angular velocity rate. The system modeled is a rigid body with an attached point mass.

## 2.1 Kinematic Equations

The equations of motion for the attitude and angular velocities must be determined. To represent the attitude, we choose a 1-2-3 Euler angle sequence to avoid the singularity associated with symmetric sequences during the linearization. The Euler angle rates take the form

$$\dot{\theta} = \mathbf{S}^{-1}\omega \tag{1}$$

Where

$$\mathbf{S}^{-1}(\theta) = \frac{1}{c\theta_2}\begin{bmatrix} c\theta_3 & -s\theta_3 & 0 \\ c\theta_2 s\theta_3 & c\theta_2 c\theta_3 & 0 \\ -s\theta_2 c\theta_3 & s\theta_2 s\theta_3 & c\theta_2 \end{bmatrix} \tag{2}$$

Equation 2 was taken from Schaub & Junkins.[10]

## 2.2 Dynamical Equations

In order to find the equations of motion for the angular velocity we follow a similar process and notation as Hughes.[12] We start by finding the angular momentum of a rigid body and then add the contribution of the point mass. The derivative is then taken in order to solve for the desired quantity, $\dot{\omega}$. See figure 1 for diagram.

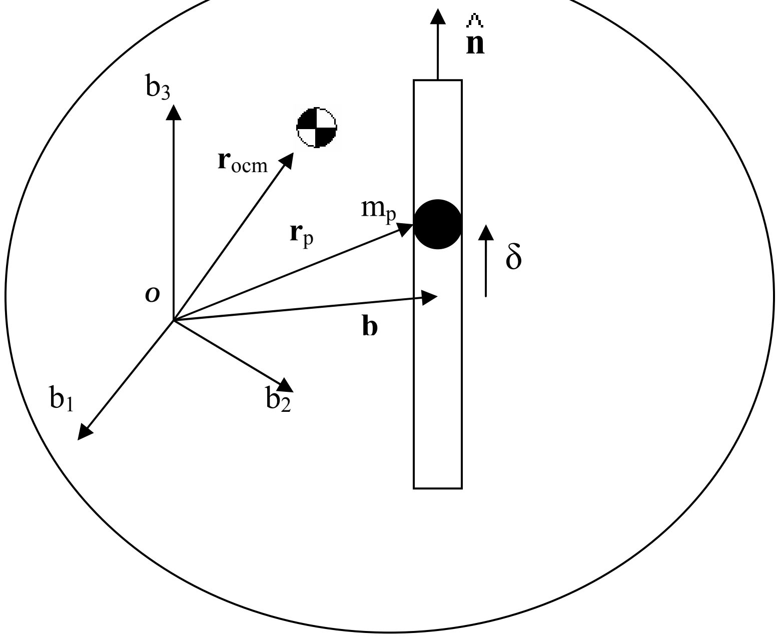

**Figure 1** Diagram of rigid body and point mass

### 2.2.1 Angular Momentum of the Body

The angular momentum of the body is taken about the point *O*. By definition the angular momentum is the cross product of the position vector from *O* to the mass element with the inertial velocity.

$$\vec{h}_b = \int_B \vec{r} \times \vec{v}\, dm = \int_B \vec{r} \times \left(\vec{v}_o + \vec{\omega}^{bi} \times \vec{r}\right) dm \tag{3}$$

Using a vector identity and the definition of the unit dyadic, we rewrite this as

$$\int_B \vec{r} \times \left(\vec{\omega}^{bi} \times \vec{r}\right) dm = \int_B \left(\vec{r} \cdot \vec{r}\right)\vec{\omega}^{bi} - \left(\vec{r} \cdot \vec{\omega}^{bi}\right)\vec{r}\, dm = \int_B r^2\, \bar{1} \cdot \vec{\omega}^{bi} - \vec{r}\vec{r} \cdot \vec{\omega}^{bi}\, dm \tag{4}$$

which leads to the final form of the body angular momentum

$$\vec{h}_b = \vec{c}_b \times \vec{v}_o + \underline{\underline{J}}_b \vec{\omega}^{bi} \tag{5}$$

where $\underline{\underline{J}}_b$ is the second moment of inertia of the body about point *O*

$$\underline{\underline{J}}_b = m_b \left(r^2 \underline{\underline{1}} - \vec{r}\,\vec{r}^T\right) \tag{6}$$

### 2.2.2 Angular Momentum of the Point Mass

The angular momentum about *O* for the point mass is found by applying the definition

$$\begin{aligned}\vec{h}_p &= \int_B \vec{r}\times\vec{v}\,dm \\ &= \int_B \vec{r}_p \times\left(\vec{v}_o + \vec{\omega}^{bi}\times\vec{r}_p + \dot{\delta}\hat{n}\right)dm \qquad (7) \\ &= m_p\vec{r}_p\times\vec{v}_o + \underline{\underline{J}}_p\vec{\omega}^{bi} + m_p\dot{\delta}\left(\vec{r}_p\times\hat{n}\right)\end{aligned}$$

where $\underline{\underline{J}}_p$ is the second moment of inertia of the point mass about point *O*

$$\underline{\underline{J}}_p = m_p\left(r_p^2\underline{\underline{1}} - \vec{r}_p\vec{r}_p^T\right) \tag{8}$$

The total angular momentum is then found by summation

$$\vec{h}^o = m_p\vec{r}_p\times\vec{v}_o + \vec{c}_b\times v_o + \underline{\underline{J}}\vec{\omega}^{bi} + m_p\dot{\delta}\left(\vec{r}_p\times\hat{n}\right) \tag{9}$$

with $\underline{\underline{J}} = \underline{\underline{J}}_b + \underline{\underline{J}}_p$.

### 2.2.3 Derivative of Angular Momentum

The time rate of change of angular momentum is equal to the external torques.

$$\dot{\vec{h}}^i = \vec{g}_{ext_i} \tag{10}$$

The fact that point *O* is not inertial must be taken into account. The resulting equation is

$$\dot{\vec{h}}^o = \dot{\vec{h}}^i - \frac{d}{dt}\left(\vec{r}_o\times\vec{p}\right) \tag{11}$$

Substituting and taking the derivative yields

$$\dot{\vec{h}}^o = \vec{g}_{ext_i} - \vec{v}_o\times\vec{p} - \vec{r}_o\times\dot{\vec{p}} \tag{12}$$

The torque about the inertial point *i* is related to the torque about *O* by

$$\vec{g}_{ext_i} = \vec{g}_{ext_o} + \vec{r}_o\times\dot{\vec{p}} \tag{13}$$

Using this relationship we find the angular momentum rate to be

$$\dot{\vec{h}}^{o} = \vec{g}_{ext_o} + \vec{r}_o \times \dot{\vec{p}} - \vec{v}_o \times \vec{p} - \vec{r}_o \times \dot{\vec{p}} = \vec{g}_{ext_o} - \vec{v}_o \times \vec{p} \tag{14}$$

To obtain the angular momentum rate in the body frame the transport theorem must be used

$${}^{B}\dot{\underline{h}}^{o} = {}^{I}\dot{\underline{h}}^{o} - \vec{\omega}^{bi} \times {}^{B}\underline{h}^{o} \tag{15}$$

Finally, the body frame time rate of change of angular momentum is found to be

$${}^{B}\dot{\underline{h}}^{o} = {}^{B}\underline{g}_{ext_o} - \underline{v}_o \times \underline{p} - {}^{B}\underline{\omega}^{bi} \times {}^{B}\underline{h}^{o} \tag{16}$$

### 2.2.4 Angular Velocity Rate

With equations 9 and 16 we are now ready to solve for $\dot{\omega}^{bi}$ explicitly. To simplify the equations, with loss of generality, we choose point *O* to be the stationary center of rotation of the simulator. All the terms involving the velocity of point *O* drop out. All calculations are done in the body frame unless otherwise notated; therefore the superscript notation will be dropped. Also, unless otherwise stated, **ω** refers to $\omega^{bi}$**.** We substitute the definition of total angular momentum from equation 9 into the left and right side of equation 16. The result is

$$\dot{\underline{\underline{J}}}\underline{\omega} + \underline{\underline{J}}\dot{\underline{\omega}} + m_p \ddot{\delta}\left(\underline{r}_p \times \underline{\hat{n}}\right) = \underline{g}_{ext} - \underline{\omega}^{x}\left(\underline{\underline{J}}\underline{\omega} + m_p \dot{\delta}\left(\underline{r}_p \times \underline{\hat{n}}\right)\right) \tag{17}$$

where we have made use of the skew symmetric matrix defined by

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \Rightarrow \mathbf{v}^{x} = \begin{bmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{bmatrix}$$

rearranging eq. 17 for $\dot{\omega}$ yields

$$\underline{\dot{\omega}} = \underline{\underline{J}}^{-1}\left\lfloor \underline{g}_{ext} - \dot{\underline{\underline{J}}}\underline{\omega} - m_p \dot{\delta}\left(\underline{r}_p \times \underline{\hat{n}}\right)\ddot{\delta} - \underline{\omega}^{x}\left(\underline{\underline{J}}\underline{\omega} + m_p \dot{\delta}\left(\underline{r}_p \times \underline{\hat{n}}\right)\right)\right\rfloor \tag{18}$$

Care must be taken at this point not dismiss the derivative of the inertia matrix. The movement of the point mass causes a change with time in the system inertia matrix.

## 2.3 Manipulation of Equations of Motion

Parameter-Adaptive Filtering can be simplified by manipulating the equations of motion to isolate the unknown parameters. Also, for ease of calculations during the PAF some limiting assumptions are made. We start with the kinematics followed by the dynamics.

### 2.3.1 Linearization of the Kinematics

Starting with equation 2 we make the limiting small angle assumption resulting in the following $\mathbf{S}^{-1}$ matrix

$$\mathbf{S}^{-1}(\theta) = \begin{bmatrix} 1 & -\theta_3 & 0 \\ \theta_3 & 1 & 0 \\ -\theta_2 & 0 & 1 \end{bmatrix} \tag{19}$$

Substituting back into equation 1 and carrying out the multiplication yields

$$\dot{\theta} = \mathbf{S}^{-1}(\theta)\omega = \begin{bmatrix} 1 & -\theta_3 & 0 \\ \theta_3 & 1 & 0 \\ -\theta_2 & 0 & 1 \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = \begin{bmatrix} \omega_1 - \theta_3\omega_2 \\ \theta_3\omega_1 + \omega_2 \\ -\theta_2\omega_1 + \omega_3 \end{bmatrix} \tag{20}$$

Another useful approximation is for the force due to gravity for the point mass and the body. The gravity always points in the $-\hat{k}$ direction in the inertial frame. Using the small angle assumption and the rotation matrix from the inertial frame to the body frame, the force due to gravity is given as [13]

$$\underline{f}_b = m_b g[\theta_2 \quad -\theta_1 \quad -1]^T \tag{21}$$

$$\underline{f}_p = m_p g[\theta_2 \quad -\theta_1 \quad -1]^T \tag{22}$$

## 2.3.2 Manipulation of the Dynamics

The motivation for this section is to isolate the location of the center of mass of the body relative to the center of rotation, $\underline{r}_{ocm}$. To assist in the isolation of $\underline{r}_{ocm}$, the inertia matrix, its derivative, and the external torque from equation 18 are separated into the contributions from the body and the point mass. Also the mass is constrained to move at a constant velocity along the path, resulting in $\ddot{\delta} = 0$. Performing these operations we arrive at

$$\underline{\dot{\omega}} = \left(\underline{\underline{J}}_b + \underline{\underline{J}}_p\right)^{-1}\left[\underline{g}_{ext,b} + \underline{g}_{ext,p} - \left(\underline{\underline{\dot{J}}}_b + \underline{\underline{\dot{J}}}_p\right)\underline{\omega} - \underline{\omega}^x\left(\left(\underline{\underline{J}}_b + \underline{\underline{J}}_p\right)\underline{\omega} + m_p\dot{\delta}\left(\underline{r}_p \times \underline{\hat{n}}\right)\right)\right] \quad (23)$$

Since the inertia of the body is constant, the derivative of the body inertia is equal to zero.

$$\underline{\underline{\dot{J}}}_b = \underline{\underline{0}} \quad (24)$$

The following equalities are easily obtained from the definitions of the vectors involved:

$$\underline{r}_p = \underline{b} + \delta\underline{\hat{n}} \quad (25)$$

$$\underline{\dot{r}}_p = \dot{\delta}\underline{\hat{n}} \quad (26)$$

$$\underline{\underline{J}}_p = m_p\left(\underline{r}_p^{\,x}\right)\left(\underline{r}_p^{\,x}\right) \quad (27)$$

$$\underline{\underline{\dot{J}}}_p = m_p\left(\underline{\dot{r}}_p^{\,x}\right)\left(\underline{r}_p^{\,x}\right) + m_p\left(\underline{r}_p^{\,x}\right)\left(\underline{\dot{r}}_p^{\,x}\right) \quad (28)$$

Using equations 21-22 , 23-27 the angular velocity rate is completely in terms of system constants and the state vector given below

$$\mathbf{x} = \begin{bmatrix} \underline{\theta} \\ \underline{\omega} \\ \delta \end{bmatrix} \qquad \text{with} \quad \underline{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} \qquad \text{and} \quad \underline{\omega} = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \quad (29)$$

# 3. Parameter Adaptive Filtering

The following is an outline of the process of Parameter Adaptive Filtering as adapted from [8]. The equations previously derived are then implemented into the process. The goal in this application is to estimate the location of the center of mass of the simulator.

## 3.1 Mathematical Formulation

We start with a nonlinear set of equations as previously derived, $\mathbf{f}_1$. In addition we are also adding a nonlinear output, **h**, representing the measurements taken from the simulator. The input equations have the unknown parameter, **p**, and could also have unknown disturbances, **w**. The output *is* subject to random noise. In summary,

$$\dot{\mathbf{x}} = \mathbf{f}_1(\mathbf{x}, \mathbf{p}, \mathbf{w}, t) \tag{30}$$

$$\mathbf{y} = \mathbf{h}(\mathbf{x}, \mathbf{v}, t) \tag{31}$$

The technique is now to add the unknown parameter vector, **p**, to the state vector. The augmented state is defined as

$$\mathbf{x}_A = \begin{pmatrix} \mathbf{x} \\ \mathbf{p} \end{pmatrix} \tag{32}$$

The result is a new system, which is more nonlinear in nature

$$\mathbf{f}_A = \dot{\mathbf{x}}_A = \begin{pmatrix} \mathbf{f}_1(\mathbf{x}, \mathbf{p}, \mathbf{w}, t) \\ \mathbf{f}_2(\mathbf{p}, \mathbf{w}, t) \end{pmatrix} \tag{33}$$

The form of the equation 23 allows for the location of the center of mass to remain constant. Since the parameter is constant, trivial dynamics, i.e. $\mathbf{f}_2 = \mathbf{0}$, are used here. An important note is that the parameter itself will not vary; only the estimate of the parameter will vary.[8] The augmented system output is given as

$$\mathbf{y}_A = \begin{pmatrix}\mathbf{h}(\mathbf{x},\mathbf{v},t) & \mathbf{0}\end{pmatrix} \tag{34}$$

A hybrid extended Kalman filter is implemented on the augmented system defined by the following equations:

$$\dot{\hat{\mathbf{x}}}_A = \mathbf{f}_A(\hat{\mathbf{x}}_A,\mathbf{u},t) + \mathbf{G}(\mathbf{y}_A - \mathbf{h}(\hat{\mathbf{x}}_A,t)) \qquad \hat{\mathbf{x}}(t_0) = \hat{\mathbf{x}}_0 \tag{35}$$

$$\mathbf{G} = \mathbf{P}\,\mathbf{C}_A(t)^T\,\mathbf{V}^{-1} \tag{36}$$

$$\dot{\mathbf{P}} = \mathbf{A}_A\mathbf{P} + \mathbf{P}\mathbf{A}_A^T - \mathbf{P}\mathbf{C}_A^T\mathbf{V}^{-1}\mathbf{C}_A\mathbf{P} + \mathbf{W} \qquad \mathbf{P}(t_0) = \mathbf{P}_0 \tag{37}$$

In equations 35-37 $\mathbf{A}_A(t)$ and $\mathbf{C}_A(t)$ are defined as the Jacobians of $\mathbf{f}_A$ and $\mathbf{y}_A$, respectively, with the noise set to zero in both cases.

## 3.2 Computational Implementation

The focus of this section is to set up the algorithm and equations needed to implement the discrete/continuous time *hybrid* extended Kalman filter used in PAF. Again the method used in [8] is followed here. In the following steps the subscript denoting the augmented system, *A,* is dropped to clarify notation. No disturbance input was assumed in order to simplify the algorithm.

An initial state estimate and covariance are needed to initialize the filter. The covariance of the parameter being estimated should be as small as possible to assure the filter in converges correctly. We start the algorithm by propagating the state estimate with the full nonlinear model:

$$\hat{\mathbf{x}}_k(-) = \hat{\mathbf{x}}_{k-1}(+) + \int_{t_{k-1}}^{t_k} \mathbf{f}(\hat{\mathbf{x}}(\tau),\mathbf{u}(\tau),\tau)\,d\tau \tag{38}$$

Second, the error covariance is propagated using the locally linearized model:

$$\mathbf{P}_k(-) = \mathbf{P}_{k-1}(+) + \int_{t_{k-1}}^{t_k} \mathbf{A}(\tau)\mathbf{P}(\tau) + \mathbf{P}(\tau)\mathbf{A}(\tau)^T \, d\tau \qquad (39)$$

Third, the observer gain of equation 35 is obtained using the locally linearized model:

$$\mathbf{G}_k = \mathbf{P}_k(-)\mathbf{C}_k^T\left(\mathbf{C}_k\mathbf{P}_k(-)\mathbf{C}_k^T + \mathbf{V}_k\right)^{-1} \qquad (40)$$

Next the nonlinear output is used along with the observer gain to update the state and covariance estimates:

$$\hat{\mathbf{x}}_k(+) = \hat{\mathbf{x}}_k(-) + \mathbf{G}_k\left(\mathbf{y}_k - \mathbf{h}(\hat{\mathbf{x}}_k(-), \mathbf{0}, t_k)\right) \qquad (41)$$

$$\mathbf{P}_k(+) = (\mathbf{I} - \mathbf{G}_k\mathbf{C}_k)\mathbf{P}_{k-1}(-) \qquad (42)$$

These steps are repeated for the duration of the simulation. With the augmented system the updated state estimate includes an approximation for the unknown parameter, **p**, at each time step.

# 4. Results

Now that we have the equations of motion developed and suitable for the parameter-adaptive filter routine, three nominal configurations are investigated. The choice for each is based upon the actual configuration onboard the spacecraft simulator, Whorl-I. There is a linear actuator, controlled moving mass, for each axis on the simulator corresponding to the chosen configurations. The system constants not dependent on configuration are the mass of the body, mass of the point mass, inertia of the body, position of the rate gyro, and speed of the mass. All of these parameters were assigned values corresponding to the physical system. Also, the initial conditions for the states were set to zero for all simulations.

The inputs which distinguish between the test cases, aside from the linear actuator being used, are the noise covariance, initial uncertainty, and time between measurements. Each simulation propagates the full nonlinear model with knowledge of the location of the center of mass. Gaussian white noise is added to the true model at each measurement step to simulate sensor noise. The effects of these variables on the convergence of the PAF are discussed throughout the remainder of this section.

## 4.1 Noise Covariance

The measurements taken at each time step were modeled from the sensors available on the simulator. Currently the only sensors available are a three-axis rate gyro and a three axis accelerometer. However, attitude knowledge was also modeled as sensors will be available in the future. The rate gyros and accelerometers provide two vector outputs, namely

$$\mathbf{y}_{rg} = \omega \tag{43}$$

$$\mathbf{y}_{acc} = \dot{\omega}^x \mathbf{r}_y + \omega^x \omega^x \mathbf{r}_y - g\mathbf{R}^{bi}\hat{\mathbf{k}} \tag{44}$$

Based upon the manual for the MotionPak II and calibration conducted in [14], the accuracy for both of these outputs was approximately ±5º/s and ±0.5m/s^2. The values for measurement noise were chosen accordingly. Due to the lack of attitude sensors at present, a number of uncertainty levels were investigated. The location of the point mass is given by a step motor with negligible noise; therefore the noise was set to zero for this output. The noise amplitude, **v,** and covariance matrices, **V**, took the following form for the various outputs.

$$\mathbf{v}_\theta = n * \sigma \quad (45a) \qquad \mathbf{V}_\theta = (n * \sigma)^2 * \mathbf{I}_{3x3} \quad (45b)$$

$$\mathbf{v}_{\omega,yacc} = \sigma \quad (46a) \qquad \mathbf{V}_{\omega,yacc} = \sigma^2 * \mathbf{I}_{6x6} \quad (46b)$$

## 4.2 Parameter Uncertainty

The *a priori* knowledge of the unknown parameters is known to have a large impact on the effectiveness of PAF.[8] The robustness of this technique was tested by varying the covariance of the initial guess, i.e. $\mathbf{P}_o$. The initial angular velocity was taken to be known exactly. The initial attitude error was varied corresponding to the sensor noise chosen for the same simulation.

## 4.3 Three Nominal Configurations

The equations of motion developed were only for a single point mass and a rigid body. The location and path direction of the mass were varied to simulate the motion of each of the three available linear actuators on the simulator. All simulations included the mass moving 20 cm., which is the full length of the available linear actuators. The system excitation due to the location and path of the masses was investigated.

### 4.3.1 X-Axis Linear Actuator

The first configuration we will consider is the mass moving parallel to the x-axis, towards the y-axis. The nominal position of the mass is at one end of the traverse. The corresponding position vectors are

$$\mathbf{b} = \begin{bmatrix}0.5 & 0.5 & 0\end{bmatrix}^T \qquad \hat{\mathbf{n}} = \begin{bmatrix}1 & 0 & 0\end{bmatrix}^T$$

| | Sim 1 | Sim 2 | Sim 3 | Sim 4 | Sim 5 |
|---|---|---|---|---|---|
| Time Step, (s) | 0.01 | 0.01 | 0.01 | 0.1 | 0.01 |
| Noise, $\sigma$ | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 |
| Attitude, n | 3 | 3 | 1 | 3 | 3 |
| $r_{ocm}$ (m) | [-0.01, 0.01, 0.02] | [-0.01, -0.01, 0.02] | [-0.01, -0.01, 0.02] | [-0.01, -0.01, 0.02] | [-0.01, -0.01, 0.02] |
| $\mathbf{P}_o$ for $\mathbf{r}_{ocm}$ (m^2) | 0.02 | 0.02 | 0.02 | 0.02 | 0.5 |

**Table 1** Inputs used for X-axis simulations, used in eq. 45 and 46

The Euler angles, angular velocity, and angular accelerations all increased dramatically in the first half second of simulation 1. The seemingly skewed model was investigated further. The problem was a discrepancy between the physical simulator and the model. The air bearing setup is by no means a single point of contact for rotation. The forces acting on the simulator are not concentrated directly upward in the center.

As an alternative to the existing model, the location of the center of mass was chosen such that the initial body and point mass torques cancel. The system would become excited as the point mass moved. The resulting angles, angular velocities, and angular accelerations were much more similar to those of the physical simulator (figures 3-4). Both the simulation and the physical simulator approach 10 degrees in roll and yaw after approximately 1 second. All future simulations were conducted with the contrived center of mass. The following, figures 2-4, are the results for simulation 2. Figure 2 shows the convergence of the unknown vector, $\mathbf{r}_{ocm}$, to the correct value to within 2 mm for all three components. The z-component of the center of mass took much longer to converge do to lack of excitation and the length of time the corresponding Euler angle took to converge. The results for simulation 3 are not shown here, the dynamics remained very similar regardless of the increased Euler angle noise. The location of the center of mass converged only slightly faster than the previous simulation.

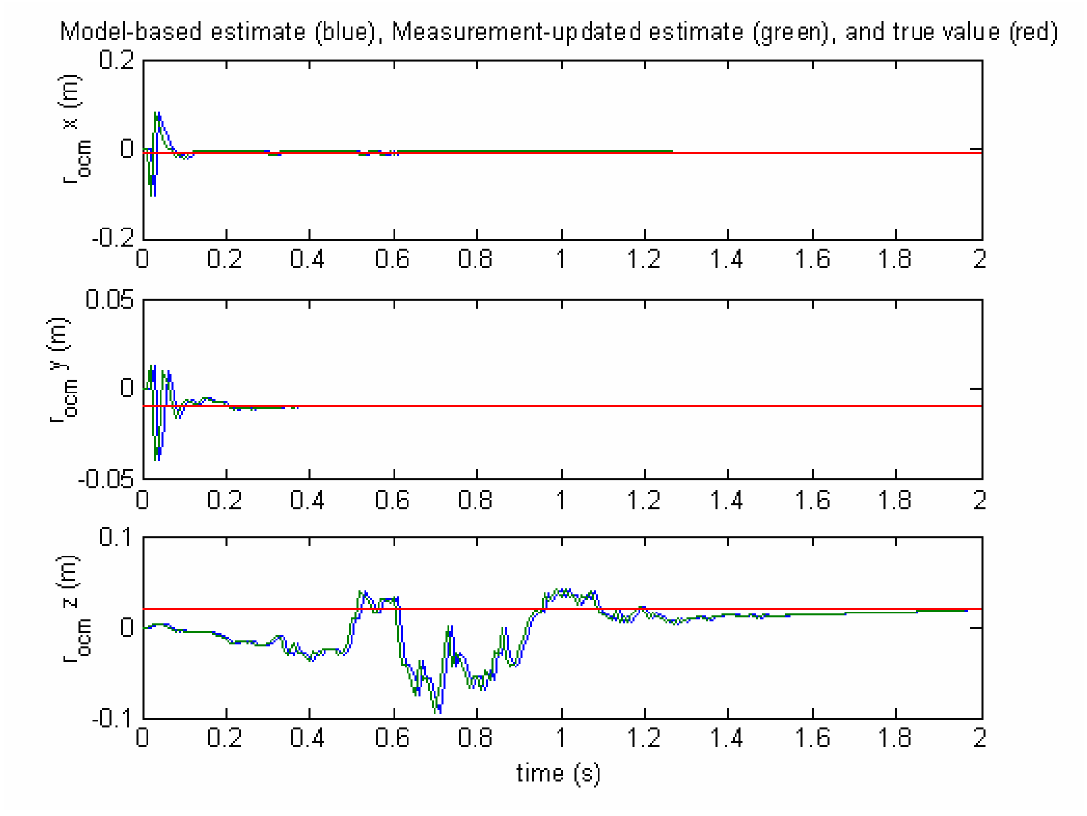

**Figure 2** Simulation 2 results for location of center of mass

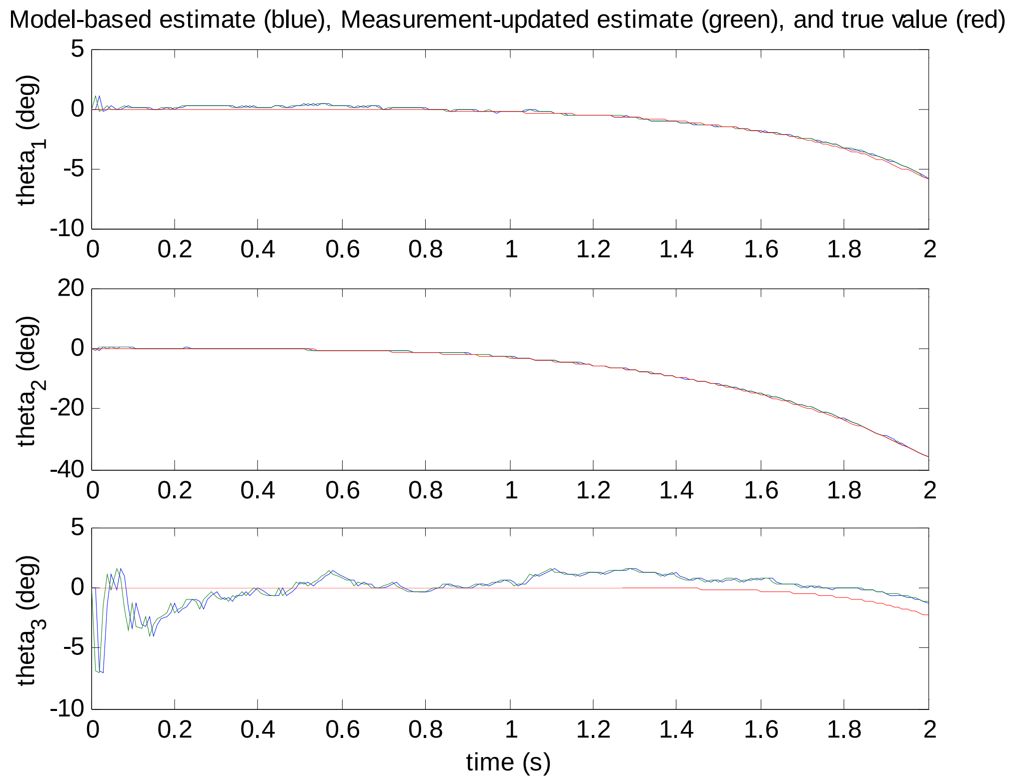

**Figure 3** Simulation 2 results for Euler angles

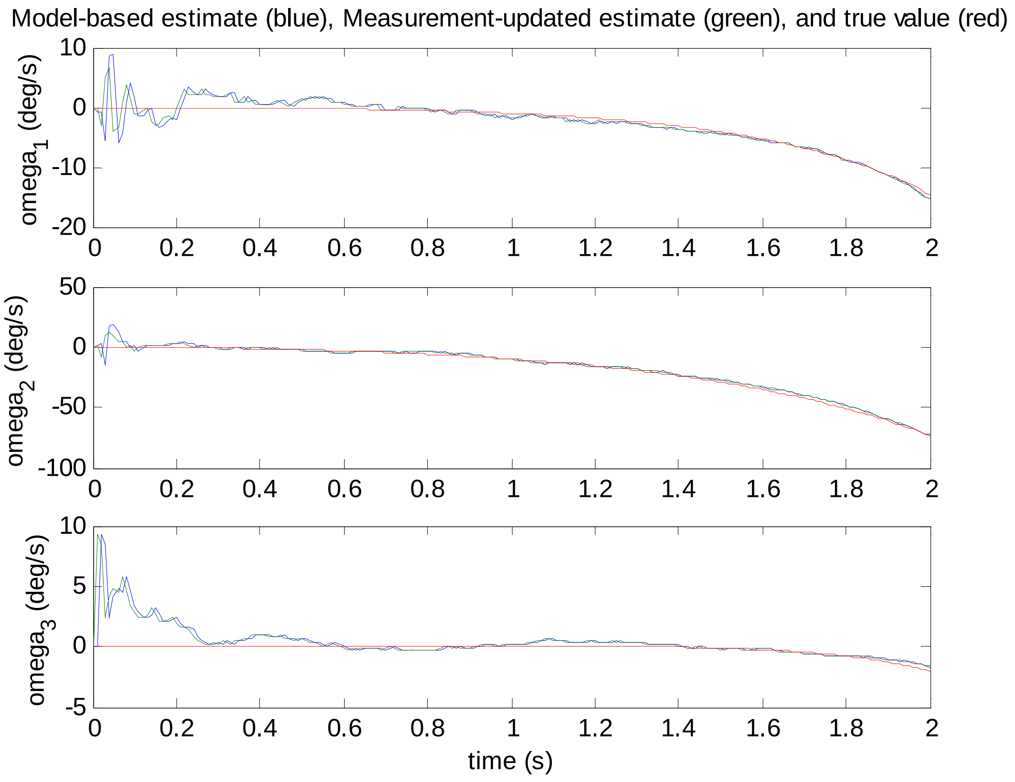

**Figure 4** Simulation 2 results for angular velocity

Simulation 4 investigated the effect of measurement time increments. Having held everything else constant from simulation 2, the sampling rate was changed from 100 Hz. to 10 Hz. The results show that the sampling rate has a large impact on the effectiveness of PAF. Without a large number of measurement updates the simulated model has longer to stray from the true dynamics. Figure 5 shows the estimate of the location of the center of mass.

Simulation 5 investigated the effect of the *a priori* estimate of the center of mass. This was done by changing the initial covariance, $\mathbf{P}_o$. Again, simulation 2 was used to compare against. The covariance was increased from 0.02 m^2 until the system diverged. This occurred at 0.3 m^2, figure 6 shows the results for the first 0.7 sec of the simulation, at which time the estimate begins to diverge.

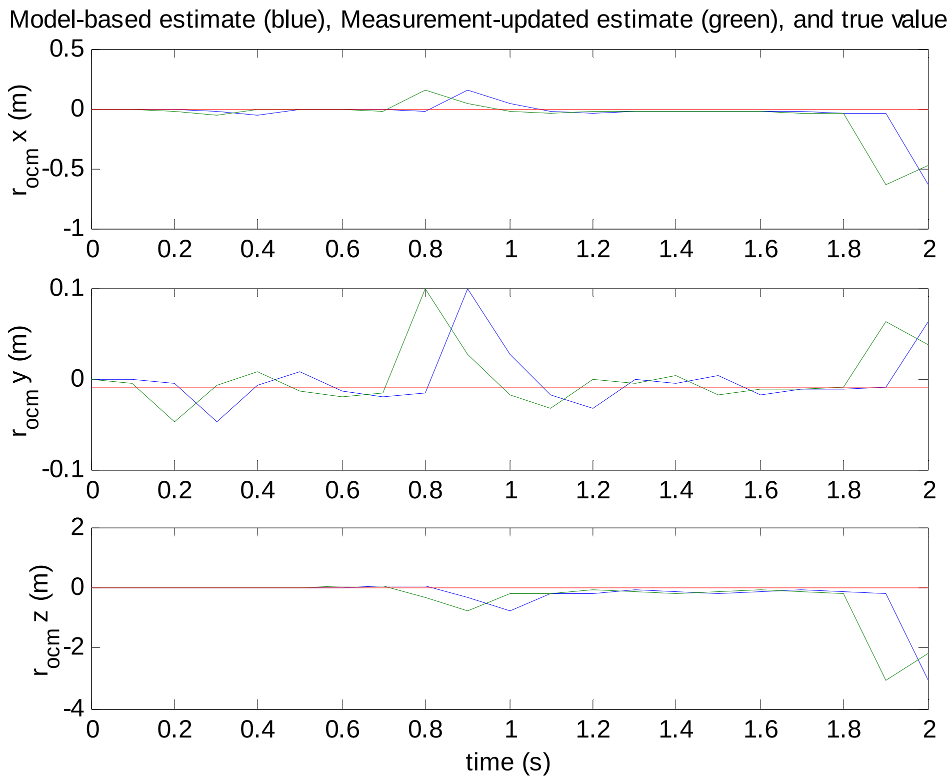

**Figure 5** Simulation 4 results for location of center of mass

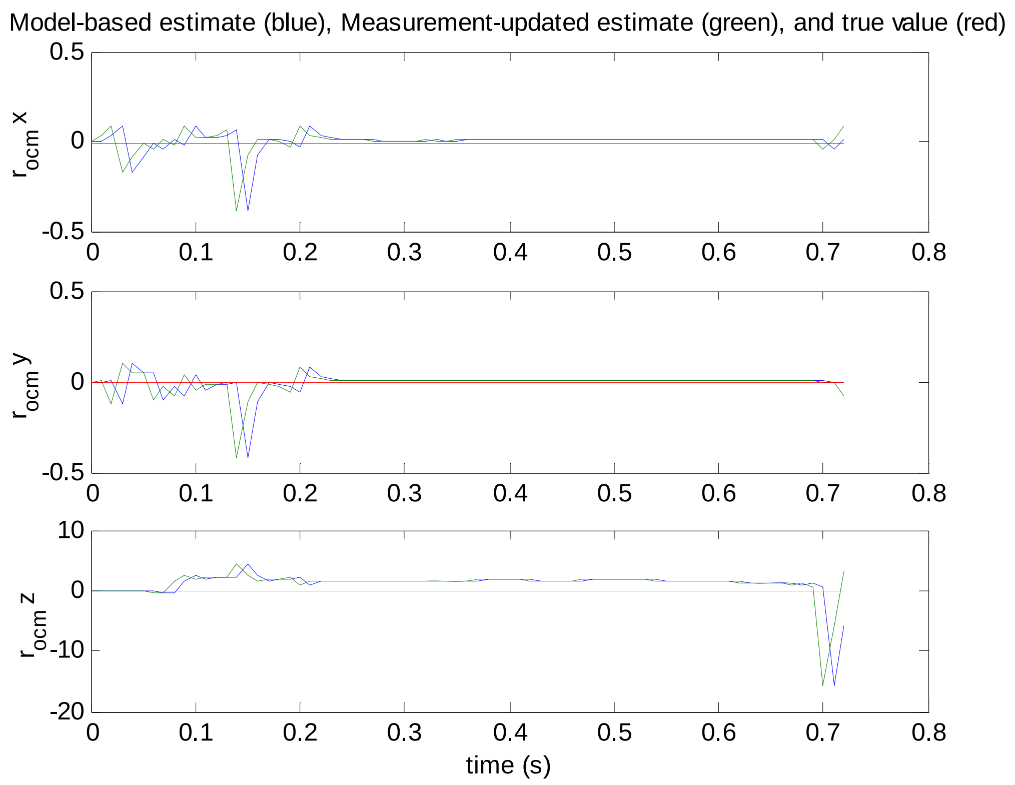

**Figure 6** Simulation 5 results for location of center of mass, through divergence

### 4.3.2 Y-Axis Linear Actuator

Simulations were run with the mass moving parallel to the y-axis, moving toward the x-axis. The nominal position of the mass is at one end of the traverse. The corresponding position vectors are

$$\mathbf{b} = \begin{bmatrix} 0.5 & 0.5 & 0 \end{bmatrix}^T \qquad \hat{\mathbf{n}} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T$$

There were no significant differences from the x-axis linear actuator simulations. This is due to the symmetry of the location of the point mass and the 'close to symmetric' properties of the body inertia matrix.

### 4.3.3 Z-Axis Linear Actuator

The final configuration considered the mass moving parallel to the z-axis, away from the X-Y plane. The nominal position is once again at one end of the traverse. The position vectors defining this configuration are

$$\mathbf{b} = \begin{bmatrix}0.5 & 0 & 0\end{bmatrix}^T \qquad \hat{\mathbf{n}} = \begin{bmatrix}0 & 0 & 1\end{bmatrix}^T$$

The mass moving along the z-axis provides an inherently different system because the gravity torque produced by the point mass remains relatively constant. Again, the contrived center of mass was used, although different than the previous center of mass due to geometry. Simulations were once again run to compare the variables previously discussed, with surprisingly similar results. The effect of output noise was then investigated using this configuration. Table 2 shows the input variables.

| | Sim 6 | Sim 7 |
|---|---|---|
| Time Step, (s) | 0.01 | 0.01 |
| Noise, $\sigma$ | 0.1 | 0.2 |
| Attitude, n | 1 | 1 |
| $r_{ocm}$ (m) | [-0.01, 0.0, 0.02] | [-0.01, 0.0, 0.02] |
| $\mathbf{P}_o$ for $\mathbf{r}_{ocm}$ (m^2) | 0.02 | 0.02 |

**Table 2** Inputs used for Z-axis simulations

Simulation 6 had the standard noise applied to the output (figure 7). As in previous cases the x and y components of $\mathbf{r}_{ocm}$ converged to the correct values. However, the z component never converged for this configuration. Also, the simulation diverged approximately 2 out of 3 times it was run. The fact that it diverged shows a weakness of PAF. The technique has no guarantee of convergence, nonetheless it can be run a number of times until it does converge, still providing useful information. A larger motion about the Z-axis would provide a better rate of convergence.

When the noise was increased in simulation 7, results similar to simulation 6 were achieved. However, the initial error was larger and the settling time was longer (figure

8). If noise was increased further the system did not converge for any of the components of $\mathbf{r}_{ocm}$.

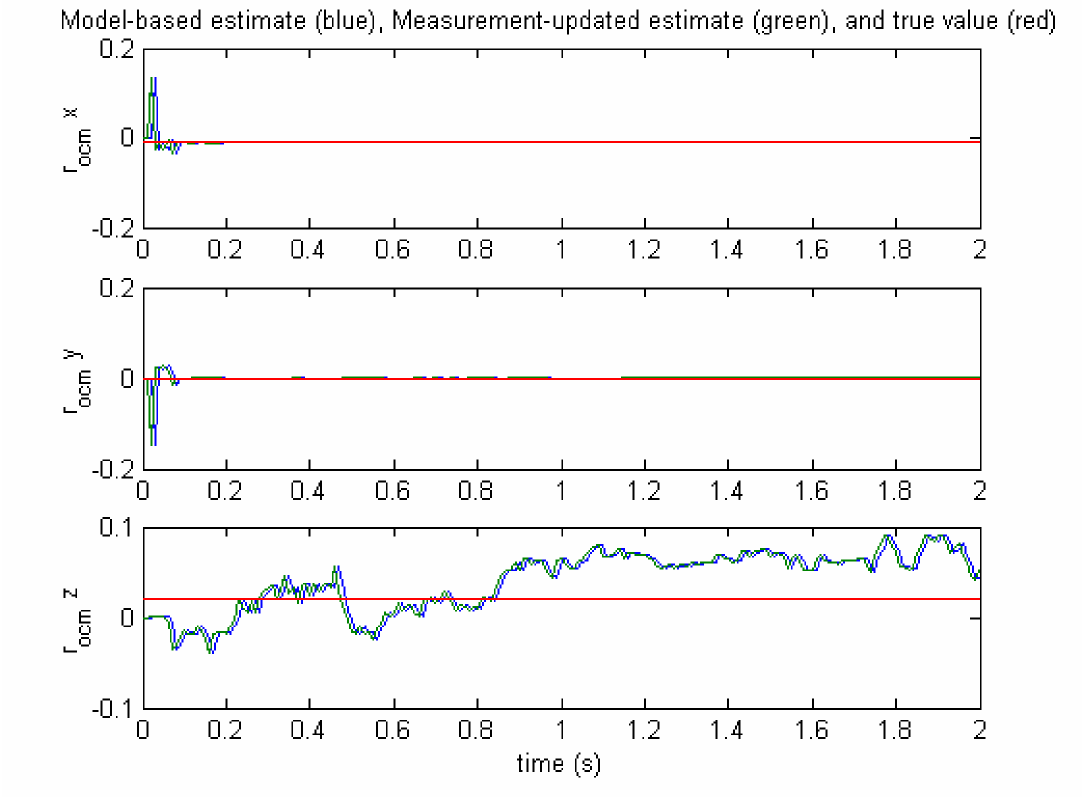

**Figure 7** Simulation 6 results for location of center of mass

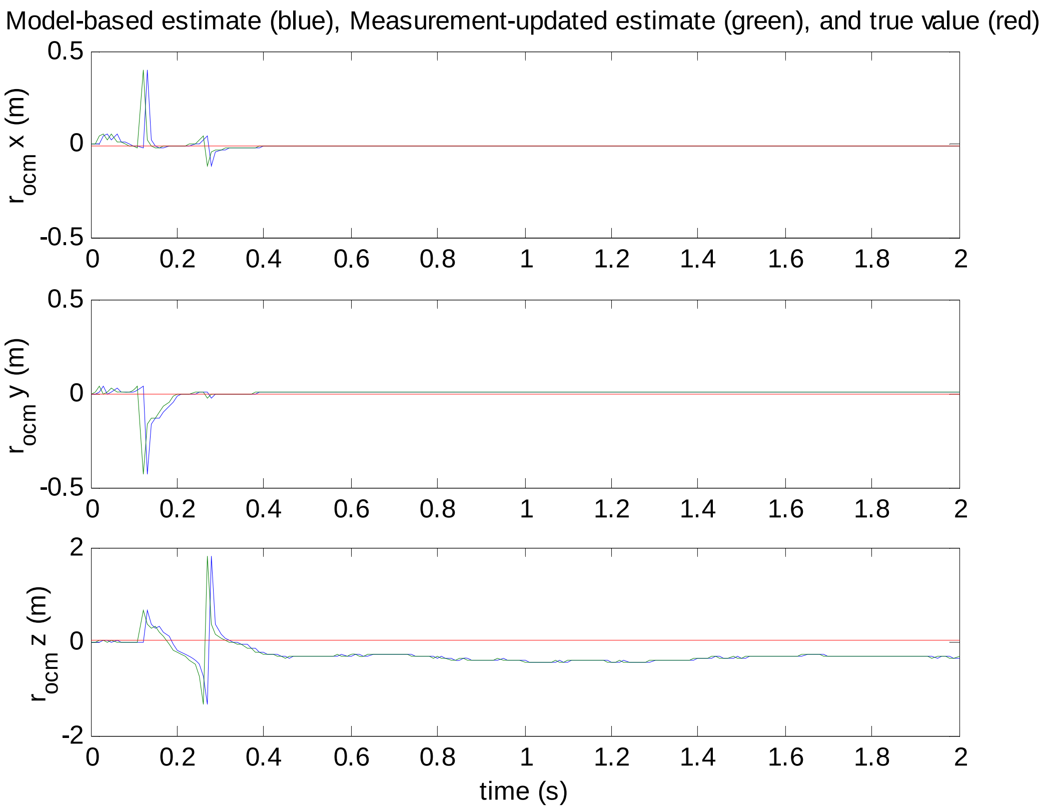

**Figure 8** Simulation 7 results for location of center of mass

### 4.3.4 Results Summary

PAF is a legitimate method to compute the center of mass of a spacecraft simulator based upon the given results. Once the location has been estimated, the masses can be actuated to adjust the center of mass to coincide with the center of rotation. Future work could expand the given algorithm to include other unknowns such as the mass of the rigid body, and/or the moments of inertia of the body. The effectiveness of PAF will decrease as the number of unknowns increase. However, if more sensor measurements become available then the effectiveness will increase.

# 5. Conclusion

The problem of estimating the center of mass of a spacecraft simulator was investigated. The equations of motion were developed for a rigid body with an attached point mass. The Parameter-Adaptive Filtering technique was outlined for a nonlinear system of equations using a hybrid extended Kalman filter. Three configurations corresponding to the three linear actuators on Whorl-I, a spacecraft simulator, were explored. PAF was successful if the output noise was sufficiently low, *a priori* knowledge of the true location was known to within 5 cm., and output measurements were taken often enough.

## References

1. Clemen, C., "New Method for On-Orbit-Determination of Parameters for Guidance, Navigation, and Control", *Acta Astronautica*, Vol. 51, No. 1-9, pp. 457-465, 2002.

2. Psiaki, Mark, "Estimation of the Parameters of a Spacecraft's Attitude Dynamics Model using Flight Data", NASA special publication, grant number NAG5-11919, 2003.

3. Tanygin and Williams, "Mass Property Estimation using Coasting Maneuvers", *Journal of Guidance, Control, and Dynamics*, Vol. 20, No. 4, July-August 1997.

4. Lee and Wertz, "In-Flight Estimation of the Cassini Spacecraft's Inertia Tensor", *Journal of Spacecraft and Rockets*, Vol. 39, No. 1, Jan. 2001, p. 153-155.

5. Paynter and Bishop, "Adaptive Nonlinear Attitude Control and Momentum Management of Spacecraft", *Journal of Guidance, Control, and Dynamics*, Vol. 20, No. 5, September-October 1997.

6. Kim and Lee, "Spacecraft Attitude Dynamics Analysis using Three-Axis Air Bearing"

7. Wright, S., "Mass Property Estimation of a Spacecraft Simulator" Project and Report. Virginia Polytechnic and State University, 2004

8. Woolsey, C.A. "Lecture 22: The Continuous and Hybrid Extended Kalman Filter," Linear Optimal Control Systems, AOE 5224, Virginia Polytechnic and State University, 2005

9. R. F. Stengel, *Optimal Control and Estimation.* Dover, 1986.

10. Schaub, H. and Junkins, J.L. *Analytical Mechanics of Space Systems.* Reston, VA. American Institute of Aeronautics and Astronautics, Inc. 2003.

11. http://www.sssl.aoe.vt.edu/documentation/sensors/MotionPakII.pdf

12. Hughes, P.C. *Spacecraft Attitude Dynamics.* Wiley, New York, 1986, pp.61-65.

13. Hauschild, A. "Development of a Damping Mechanism for a Spacecraft Simulator" Project and Report. Virginia Polytechnic and State University, 2005.

14. Schwartz, J. L. "The Distributed Spacecraft Attitude Control System Simulator: From Design Concept to Decentralized Control," PhD thesis, Virginia Polytechnic and State University. Blacksburg, VA. 2004.

# Appendix

## Wscript.m

```
% This script implements a hybrid extended Kalman filter for a spacecraft simulator

%define global variables for the simulation
global deltadot Jb mb mp b nhat g x_cs A

syms w_s1 w_s2 w_s3 th_s1 th_s2 th_s3 r_ocm_s1 r_ocm_s2 r_ocm_s3 delta_s real

omega_s = [w_s1;w_s2;w_s3];
theta_s = [th_s1;th_s2;th_s3];
r_ocm_s = [r_ocm_s1;r_ocm_s2;r_ocm_s3];

%define system constants
deltadot = -0.1 ;                    %?1x1?
Jb = [6.2, -0.9, -.2; -.9, 7.5, 0.1; -.2, 0.1, 12.1];   %?3x3?
mb = 100;                            %?1x1?
mp = 2;                              %?1x1?
b = [.5;.5;0];                       %?3x1?
nhat = [1;0;0];                      %?3x1?
r_y = [-.15;-.15;.05];                   %?3x1?
g = 9.81;                            %m/s^2

%calculate symbolic variables for use in A calculation
r_op = b+delta_s*nhat;
Jp = mp*skew(r_op)*skew(r_op);

r_opdot = deltadot*nhat;
Jpdot = mp*skew(r_op)*skew(r_opdot);

fb = mb*g*[th_s2;-th_s1;-1];
fp = mp*g*[th_s2;-th_s1;-1];

omegadot_s = inv(Jb+Jp)*(cross(r_ocm_s,fb)+cross(r_op,fp)-Jpdot*omega_s-
skew(omega_s)*(Jb+Jp)*omega_s-mp*deltadot*skew(omega_s)*(cross(r_op,nhat)));

%calculate the sub-matrix which is the omega dot jacobian
A_sub = jacobian(omegadot_s,[theta_s;omega_s;delta_s;r_ocm_s]);


%Determine the symbolic A matrix to be evaluated at each time step later

A = [  0       0   -omega_s(2)     1   -theta_s(3) 0   0   0   0   0;
       0       0    omega_s(1) theta_s(3)   1      0   0   0   0   0;
       0  -omega_s(1)   0      -theta_s(2)  0      1   0   0   0   0;
       A_sub;
       0       0       0          0       0      0   0   0   0   0;
       0       0       0          0       0      0   0   0   0   0;
       0       0       0          0       0      0   0   0   0   0;
       0       0       0          0       0      0   0   0   0   0];


%Determine the symbolic C matrix to be evaluated at each time step later

yacc_s = skew(omega_s)*skew(omega_s)*r_y - g*[-theta_s(2);theta_s(1);1];
```

```
C_sub = jacobian(yacc_s,[theta_s;omega_s;delta_s;r_ocm_s]);

C = [  1      0      0          0       0      0  0  0  0   0;
       0      1      0          0       0      0  0  0  0   0;
       0      0      1          0       0      0  0  0  0   0;
       0      0      0          1       0      0  0  0  0   0;
       0      0      0          0       1      0  0  0  0   0;
       0      0      0          0       0      1  0  0  0   0;
       C_sub;
       0      0      0          0       0      0  1  0  0   0];



% Initial states where theta is the attitude, omega is the angular rate,
% delta is the displacement of the mass from nominal position, r_ocm is
% vector from point o to the mass center

theta1_0=0;
theta2_0=0;
theta3_0=0;
omega1_0=0;
omega2_0=0;
omega3_0=0;
delta_0=0;
r_ocm1_0=-0.01;
r_ocm2_0=-0.01;
r_ocm3_0=.02;

x_0=[theta1_0;
   theta2_0;
   theta3_0;
   omega1_0;
   omega2_0;
   omega3_0;
   delta_0;
   r_ocm1_0;
   r_ocm2_0;
   r_ocm3_0];


% Sample time interval
dT=0.01;
T=[0:dT:2.0];
N=length(T);

%Actual values of the state (full nonlinear dynamics)

[T,TrueState] = ode45('True_RHS',T,x_0);


%Initial state estimate
% xhat_0  =  [theta1_0;
%            theta2_0;
%            theta3_0;
%            omega1_0;
%          omega2_0;
%          omega3_0;
%          delta_0;
%             0;
%             0;
%             0];


%Initial state covariance
P_0(1:3,1:3)=eye(3);
P_0(4:6,4:6)=eye(3);
P_0(8:10,8:10)=0.4*eye(3);

%Noise covariance

sigma=0.1;
V= (sigma^2)*eye(10);

%Initialize arrays

xhat_minus = zeros(10,N);
P_minus = zeros(10,10,N);
xhat_plus = zeros(10,N);
P_plus = zeros(10,10,N);

%Initialize extended kalman filter

xhat_plus(:,1) = xhat_0;
P_plus(:,:,1) = P_0;

%Start kalman filter

for k = 1:N-1

   %Update the current state for each interval
   x_cs = xhat_plus(:,k)

   %Update state estimate by integrating the nonlinear equations
   x_minus_0 = xhat_plus(:,k);
   t0 = T(k);
   tfinal = T(k+1);
   [t, x_minus_output] = ode45('True_RHS',[t0 tfinal], x_minus_0);
   xhat_minus(:,k+1) = x_minus_output(length(t),:)';
   clear t x_minus_0 x_minus_output

   %Update the covariance by locally linearizing the nonlinear model
   P_minus_0 = [P_plus(1,1:10,k), P_plus(2,2:10,k), P_plus(3,3:10,k) P_plus(4,4:10,k), P_plus(5,5:10,k),
P_plus(6,6:10,k) P_plus(7,7:10,k), P_plus(8,8:10,k), P_plus(9,9:10,k) P_plus(10,10,k)]';
   [t,P_minus_output] = ode45('Covariance_RHS',[t0 tfinal],P_minus_0   );
   P_minus(:,:,k+1) = [P_minus_output(length(t),1:10);
                  P_minus_output(length(t),2), P_minus_output(length(t),11:19);
                  P_minus_output(length(t),3), P_minus_output(length(t),12),
P_minus_output(length(t),20:27);
                  P_minus_output(length(t),4), P_minus_output(length(t),13), P_minus_output(length(t),21),
P_minus_output(length(t),28:34);
```

```
                P_minus_output(length(t),5), P_minus_output(length(t),14), P_minus_output(length(t),22),
P_minus_output(length(t),29), P_minus_output(length(t),35:40);
                P_minus_output(length(t),6), P_minus_output(length(t),15), P_minus_output(length(t),23),
P_minus_output(length(t),30), P_minus_output(length(t),36), P_minus_output(length(t),41:45);
                P_minus_output(length(t),7), P_minus_output(length(t),16), P_minus_output(length(t),24),
P_minus_output(length(t),31), P_minus_output(length(t),37), P_minus_output(length(t),42),
P_minus_output(length(t),46:49);
                P_minus_output(length(t),8), P_minus_output(length(t),17), P_minus_output(length(t),25),
P_minus_output(length(t),32), P_minus_output(length(t),38), P_minus_output(length(t),43),
P_minus_output(length(t),47), P_minus_output(length(t),50:52);
                P_minus_output(length(t),9), P_minus_output(length(t),18), P_minus_output(length(t),26),
P_minus_output(length(t),33), P_minus_output(length(t),39), P_minus_output(length(t),44),
P_minus_output(length(t),48), P_minus_output(length(t),51), P_minus_output(length(t),53:54);
                P_minus_output(length(t),10),P_minus_output(length(t),19), P_minus_output(length(t),27),
P_minus_output(length(t),34), P_minus_output(length(t),40), P_minus_output(length(t),45),
P_minus_output(length(t),49), P_minus_output(length(t),52), P_minus_output(length(t),54),
P_minus_output(length(t),55)];
    clear t P_minus_0 P_minus_output t0 tfinal

    %Compute observer gain
    thetahat = xhat_minus(1:3,k+1);
    omegahat = xhat_minus(4:6,k+1);
    deltahat = xhat_minus(7,k+1);
    r_ocmhat = xhat_minus(8:10,k+1);
    w_s1 = omegahat(1);w_s2 = omegahat(2);w_s3 = omegahat(3);
    C_num = eval(C);
    P = P_minus(:,:,k+1);
    G = P*C_num'*inv(C_num*P*C_num'+V);

    %Revise state estimate
    theta = TrueState(k+1,1:3)';
    omega = TrueState(k+1,4:6)';
    delta = TrueState(k+1,7);
    r_ocm = TrueState(k+1,8:10)';
    if k>30
    omegadot_appr = ((omega-TrueState(k,4:6)')./dT);
    omegadot_apprhat = ((omegahat-xhat_minus(4:6,k))./dT);
    else
        omegadot_appr = [0;0;0];
        omegadot_apprhat = [0;0;0];
    end
    yacc = skew(omegadot_appr)*r_y+skew(omega)*skew(omega)*r_y - g*[-theta(2);theta(1);1];
    yacchat = skew(omegadot_apprhat)*r_y+skew(omegahat)*skew(omegahat)*r_y - g*[-
thetahat(2);thetahat(1);1];
    y = [theta;omega;yacc;delta] + [3*sigma*randn(3,1); sigma*randn(3,1); sigma*randn(3,1);0];
    yhat = [thetahat;omegahat;yacchat;deltahat];
    xhat_plus(:,k+1) = xhat_minus(:,k+1)+G*(y-yhat);
    clear theta omega delta r_ocm thetahat omegahat deltahat r_ocmhat w_s1 w_s2 w_s3

    %Revise covariance estimate
    thetahat = xhat_plus(1:3,k+1);
    omegahat = xhat_plus(4:6,k+1);
    deltahat = xhat_plus(7,k+1);
    r_ocmhat = xhat_plus(8:10,k+1);
    w_s1 = omegahat(1);w_s2 = omegahat(2);w_s3 = omegahat(3);
    C_num = eval(C);
    P_plus(:,:,k+1) = (eye(10)-G*C_num)*P_minus(:,:,k+1);
    clear thetahat omegahat deltahat r_ocmhat w_s1 w_s2 w_s3
    k
end

figure(1)
subplot(3,1,1);
plot(T,xhat_minus(8,:),T,xhat_plus(8,:),T,TrueState(:,8));
ylabel('r_o_c_m x (m)');
title('Model-based estimate (blue), Measurement-updated estimate (green), and true value (red)');
subplot(3,1,2);
plot(T,xhat_minus(9,:),T,xhat_plus(9,:),T,TrueState(:,9));
ylabel('r_o_c_m y (m)');
subplot(3,1,3);
plot(T,xhat_minus(10,:),T,xhat_plus(10,:),T,TrueState(:,10));
ylabel('r_o_c_m z (m)');
xlabel('time (s)');


figure(2)
subplot(3,1,1);
plot(T,xhat_minus(1,:)*180/pi,T,xhat_plus(1,:)*180/pi,T,TrueState(:,1)*180/pi);
ylabel('theta_1 (deg)');
title('Model-based estimate (blue), Measurement-updated estimate (green), and true value (red)');
subplot(3,1,2);
plot(T,xhat_minus(2,:)*180/pi,T,xhat_plus(2,:)*180/pi,T,TrueState(:,2)*180/pi);
ylabel('theta_2 (deg)');
subplot(3,1,3);
plot(T,xhat_minus(3,:)*180/pi,T,xhat_plus(3,:)*180/pi,T,TrueState(:,3)*180/pi);
ylabel('theta_3 (deg)');
xlabel('time (s)');

figure(3)
subplot(3,1,1);
plot(T,xhat_minus(4,:)*180/pi,T,xhat_plus(4,:)*180/pi,T,TrueState(:,4)*180/pi);
ylabel('omega_1 (deg/s)');
title('Model-based estimate (blue), Measurement-updated estimate (green), and true value (red)');
subplot(3,1,2);
plot(T,xhat_minus(5,:)*180/pi,T,xhat_plus(5,:)*180/pi,T,TrueState(:,5)*180/pi);
ylabel('omega_2 (deg/s)');
subplot(3,1,3);
plot(T,xhat_minus(6,:)*180/pi,T,xhat_plus(6,:)*180/pi,T,TrueState(:,6)*180/pi);
ylabel('omega_3 (deg/s)');
xlabel('time (s)');

figure(4)
plot(T,xhat_minus(7,:),T,xhat_plus(7,:),T,TrueState(:,7));
ylabel('delta (m)');
xlabel('time (s)');
```

## True_RHS.m

```
%True Dynamics for the simulation of a rigid body and point mass

function x1dot = True_RHS(t,x1)

global deltadot Jb mb mp b nhat g


%break state into individual vectors
theta = x1(1:3);
omega = x1(4:6);
delta = x1(7);
r_ocm = x1(8:10);

%calculate current location of point mass from 'o'
r_op = b+delta*nhat;

%calculate current contribution of point mass to inertia matrix
Jp = mp*skew(r_op)*skew(r_op);

%calculate rate of change of Jp
r_opdot = deltadot*nhat;
Jpdot = mp*skew(r_op)*skew(r_opdot);

%calculate the forces due to gravity

fb = mb*g*[theta(2);-theta(1);-1];
fp = mp*g*[theta(2);-theta(1);-1];

%propagate the states
thetadot = [ omega(1)-theta(3)*omega(2);
             theta(3)*omega(1)+omega(2);
            -theta(2)*omega(1)+omega(3)];

%        cross(r_ocm,fb)
%        cross(r_op,fp)
%        Jpdot*omega
%        skew(omega)*(Jb+Jp)*omega
%        mp*deltadot*skew(omega)*(cross(r_op,nhat))


omegadot = inv(Jb+Jp)*(cross(r_ocm,fb)+cross(r_op,fp)-Jpdot*omega-skew(omega)*(Jb+Jp)*omega-
mp*deltadot*skew(omega)*(cross(r_op,nhat)));

r_ocmdot = [0;0;0];

%assemble statedot column vector
x1dot = [thetadot;omegadot;deltadot;r_ocmdot];
```

## Covariance_RHS.m

```
%RHS file for the covariance propagation

function xdot = Covariance_RHS(t,x)

global deltadot Jb mb mp b nhat g x_cs A


%break state into individual values, these are the current state values...constant during the integration
period
th_s1 = x_cs(1);
th_s2 = x_cs(2);
th_s3 = x_cs(3);
w_s1 = x_cs(4);
w_s2 = x_cs(5);
w_s3 = x_cs(6);
delta_s = x_cs(7);
r_ocm_s1 = x_cs(8);
r_ocm_s2 = x_cs(9);
r_ocm_s3 = x_cs(10);

P = [x(1:10)';
    x(2), x(11:19)';
    x(3), x(12), x(20:27)';
    x(4), x(13), x(21), x(28:34)';
    x(5), x(14), x(22), x(29), x(35:40)';
    x(6), x(15), x(23), x(30), x(36), x(41:45)';
    x(7), x(16), x(24), x(31), x(37), x(42), x(46:49)';
    x(8), x(17), x(25), x(32), x(38), x(43), x(47), x(50:52)';
    x(9), x(18), x(26), x(33), x(39), x(44), x(48), x(51), x(53:54)',
    x(10),x(19), x(27), x(34), x(40), x(45), x(49), x(52), x(54), x(55)];

A_num = eval(A);

Pdot = A_num*P + P*A_num';

xdot =
[Pdot(1,1:10),Pdot(2,2:10),Pdot(3,3:10),Pdot(4,4:10),Pdot(5,5:10),Pdot(6,6:10),Pdot(7,7:10),Pdot(8,8:10),P
dot(9,9:10),Pdot(10,10)]';
```

## skew.m

```
% xc = cr(x)
%     x = 3x1 matrix
%     xc = skew symmetric 3x3 matrix
%         [ 0  -x(3),  x(2)
%           x(3)  0,   -x(1)
%          -x(2)  x(1)   0  ]
function xc = cr(x)
xc = [ 0,  -x(3),  x(2);
      x(3),  0,   -x(1);
     -x(2), x(1),   0 ];
```