SSC24-WV-08

Generalized Attitude Estimation for Spacecraft

Patrick McKeen, Kerri Cahoy
Massachusetts Institute of Technology
77 Massachusetts Ave, Room 37-335 Cambridge, MA 02139; (617) 253-7805
pmckeen@mit.edu

ABSTRACT

Attitude state estimation for spacecraft often requires writing an integrator or filter from scratch, with choices depending on the expected sensors, spacecraft and mission properties, and computational resources available. We create a customizable object-oriented satellite dynamics model with a variety of sensors, actuators, and disturbances that can be easily applied to any small spacecraft. Relevant disturbances can be modeled (or ignored), sensor and actuator biases are tracked, and outside effects like eclipse on sun sensors are included. In this work, we demonstrate the use of a new model to create a dynamics-aware unscented Kalman filter (UKF) that, in simulation, outperforms current and previous estimators, even for large satellites and achieves sub-degree accuracy for a small satellite. For example, the mass properties of a satellite, the actuator and sensor properties, and relevant disturbances can be input, and the UKF with relevant dynamics, update, and propagation steps will be created. This allows for rapid testing of various estimation and control paradigms, and the quick development of an attitude determination system (ADS).

Nomenclature

$\bar{q}$	= quaternion representing orientation
β	= bias of sensor or actuator
ω	= angular velocity
τ	= torque
τ_s	= torque on stored momentum
s	= sensor reading
m	= spacecraft mass
$\mathbf{c}$	= location of spacecraft center of mass
α	= body-frame axis of sensor or actuator
J	= spacecraft inertia tensor
$\mathbf{B}_{\mathrm{ECI}}$	= local magnetic field in Earth-Centered Inertial (ECI) frame
$\mathbf{S}_{\mathrm{ECI}}$	= sun position (from satellite) in ECI frame
$\mathbf{R}_{\mathrm{ECI}}$	= position of spacecraft in ECI frame
$\mathbf{V}_{\mathrm{ECI}}$	= velocity of spacecraft in ECI frame
ν	= noise on a sensor or actuator
η	= standard deviation of a stochastic process such as rate of bias drift
$\mathbf{u}$	= control command
$N(\mu, \sigma)$	= Normal distribution with mean μ and variance σ^2
$\mathbf{p}$	= time-varying disturbance parameter
$\hat{\mathbf{r}}$	= unit vector
$I_{3\times3}$	= identity matrix of size 3 by 3
$A(\bar{q})$	= Rotation matrix of $\bar{q}$, from body to ECI
$\lVert\mathbf{x}\rVert$	= 2-norm of $\mathbf{x}$

1 Introduction

Attitude estimation has been a challenge since the early days of space exploration. Hardware sensors capture information used by onboard state estimation and control algorithms to generate commands for hardware actuators in order to achieve the desired spacecraft state. The sensing hardware can include gyroscopes, magnetometers, sun and Earth sensors, star trackers and cameras, and radio beacons.[1] The information from these sensors can be turned into estimated attitude through a number of algorithms, including the TRIAD vector estimation algorithm, the extended Kalman filter (EKF), and the unscented Kalman filter (UKF).[2] Many more complex approaches also exist, including robust filtering for detecting measurement faults[3–5] , lower-computational-cost approaches,[6] active control to improve parameter estimation,[7] full GPS inclusion,[8] and sparse-matrix approaches.[9]

The many approaches to this problem have tradeoffs. Some ADCSs (attitude determination and control systems) do not track angular velocity directly, instead using the readings off the gyroscopes as raw input in the predictive step and tracking their bias.[10] This works fine when noise is small, but can cause problems when higher-noise gyroscopes are used, such as in cubesats. In such applications, tracking angular velocity allows for better estimation. Many ADCS estimators do not include the control commands in their estimation, which means some information is not being used. For example, if a torque is commanded to actuators, the expected change in angular velocity is known from the commands and the mass properties of the satellite. When the expected change in angular velocity is compared to the measured change from the gyroscopes, more information about the gyroscope biases are understood. Similarly, the angular velocity change that results from magnetic torque provides information about the satellite's orientation relative to the magnetic field. Using the noise and bias of the control torques allows for more specific modeling of the covariance and noise in the system, in addition to accounting for it.

Modeling known disturbances, analytically or by estimation, allows for more precise estimation than a general "process error" term. We have more information about the system than simple filters include; for example, the torque from drag and residual dipoles can be modeled near exactly, up to the accuracy of their parameters.[11] When disturbances cannot be fully modeled, or the parameters are uncertain, "generalized" disturbance torque can be estimated. The generalized disturbance torque effectively serves as the residual on state estimation, accounting for uncaptured disturbances/dynamics errors. This can then be used in controllers – for example, as a feedforward adjustment in PID instead of the integral term.

To better estimate system state, and to understand the disturbances so that they can be more directly countered, this work details the creation and testing of an estimator framework for ADCS that fully models system dynamics, control effort and bias, disturbances, and sensor bias. The estimation framework is created in an object-oriented manner to allow for easy modifications and simplify applying tests to different satellites, noise environments, and more. Each estimator, which can include a Extended Kalman Filter or more advanced approaches, processes the sensor and control information, relying on an internal "digital twin" of the satellite to model the dynamics, sensors, actuators, and disturbances.

2 Approach

The attitude determination system described in this work was originally inspired by the ADCS needs for a student CubeSat mission. The mission plan called for only magnetic attitude control and onboard electric propulsion. During development, the concern about torque created by the propulsion (potentially caused by either off-axis thrust, or misalignment between the thrust vector and center of mass) was raised. While the torque would be small, it would be continuous for hours, and if any component of it was aligned with the local **B** field, it would cause rotation that could not be countered. These possibilities led to the creation of an attitude trajectory planner that could include and utilize disturbance torques, using them to counter each other or relying on time-varying properties.[12] This required an estimator that tracked and estimated uncertain or time-varying disturbances, as well as actuator noise and bias. Aside from underactuated CubeSats, this estimator had the potential value of improving ADCS on other satellites, by allowing for direct consideration of disturbances and actuator bias. This allows for feed-forward control where torques are directly countered, as opposed to controlling based on their effect in, for example, a PID control loop. This led to the inclusion of disturbance parameters and actuator biases in an ADCS estimator, as detailed in this work. By fully modeling the dynamics, disturbances, and estimating sensor properties, more information could be factored into the system, allowing for more accurate estimation, even if sensors were of low quality.

To understand the effect of fully including the dynamics, biases, noise, and disturbances in the estimator, a framework is created that allows for satellites to be modeled with specific disturbances, sensor biases and noise, and actuator biases and noise. The disturbances and biases can be turned on or off for various tests, and their relevant parameters can be changed and/or allowed to evolve over time according to specific random-walk models. This object-oriented system allows for an arbitrary set of actuators and sensors for any satellite. This system is used inside the ADS to create an internal "virtual satellite" that matches the system estimates of biases and disturbances. The dynamics of the virtual satellite were modeled to fully include these disturbances and biases (or not, if turned off), and thus the dynamics, random-walk models, and other functions can then be used for attitude determination and control.

To improve estimation accuracy, the estimation system includes control noise separately from integration error in the state estimation system. The separate modeling of control noise, integration noise, and sensor noise is implemented in the estimators in the framework of this paper. With the satellite system parameterized to allow for sensors, actuators, physical properties, and disturbances to be included or not, and with their evolution allowed to vary in different ways, the attitude estimation framework is written to build off these choices in a straightforward manner. Not only does this approach allow for easier testing, but it can be applied to other satellites, with varying properties, plants, and sensors, with minimal effort.

2.1 Estimated State

The core of any estimation method is the state–the parameters or values that the system is trying to estimate. For this framework, the state has several components: the base state (basic rotational dynamics values), actuator biases, sensor biases, and disturbance parameters. The exact parts of each of these vary by application instance, but the form is:

$$\mathbf{x}_{\text{full}} = \begin{bmatrix}\mathbf{x}_{\text{base}}^T & \beta_{\text{actuator}}^T & \beta_{\text{sensor}}^T & \mathbf{p}_{\text{dist}}^T\end{bmatrix}^T \tag{1}$$

where the presence of biases (β) and disturbance parameters ($\mathbf{p}_{\text{dist}}$) depend on how the satellite is set up and whether the components are active (e.g., in the case of disturbances).

The base state is:

$$\mathbf{x}_{\text{base}} = \begin{bmatrix}\bar{q}^T & \omega^T & \mathbf{h}^T\end{bmatrix}^T \tag{2}$$

where $\bar{q}$ is the quaternion defining orientation (converted into a 4-vector), ω is the body-frame angular velocity, and $\mathbf{h}$ is the vector of momenta stored in each actuator (which is zero and thus absent, except for reaction wheels).

With perfect information, this state could be tracked exactly, based on the dynamics that define the relevant physical properties. Without perfect information, the dynamics are used to estimate the system as they are rules governing its evolution. Specifically, the dynamics of these components are governed by the Satellite class (see Section 2.2.1) as follows (allowing quaternions to operate as 4-vectors and ω to operate as a quaternion), where J is the inertia tensor of the satellite about the center of mass, including all reaction wheels, $j_{\text{act},i}$ is the moment of inertia of the ith reaction wheel about its axis and $\mathbf{u}$ represents the control commands,

$$\dot{\bar{q}} = \frac{1}{2}\bar{q}\omega = \frac{1}{2}\begin{bmatrix}0 & -\omega^T \\ \omega & -[\omega]^\times\end{bmatrix}\bar{q} \tag{3}$$

$$\dot{\omega} = J_{\text{noRW}}^{-1}\left(-\omega \times \left(J_{\text{noRW}}\omega + \sum_{i=1}^{N_{\text{act}}}\hat{\alpha}_{\text{act},i}h_i\right) + \tau_{\text{all}}\right) \tag{4}$$

$$\dot{h}_i = -\tau_{s,i} - j_{\text{act,i}}\left(\hat{\alpha}_{\text{act,i}}^T\dot{\omega}\right) \tag{5}$$

where $J_{\text{noRW}} = J - \sum_{i=1}^{N_{\text{act}}} j_{\text{act},i}\hat{\alpha}_{\text{act},i}\hat{\alpha}_{\text{act},i}^T$ represents the spacecraft inertia tensor ignoring all momentum storage and $\tau_{\text{all}} = \sum_{i=1}^{N_{\text{act}}}\tau_{\text{act},i} + \sum_{i=1}^{N_{\text{dist}}}\tau_{\text{dist},i}$ is the sum of all actuator and disturbance torques.

These equations define the system's "real-world" dynamics and can thus be used to complete the filter's predictive step or used in a simulation. The dynamics of other components (biases, noise, and disturbances) are contained in classes relevant to each, and described in the rest of this Section.

2.2 Overall Architecture

To allow for this code to be applied to different satellites and scenarios, and to allow for easy modifications during testing, the estimation framework is created in an object-oriented manner. An Estimator class (with sub-classes for Extended Kalman Filters (EKFs), Unscented Kalman Filters (UKFs), Square Root UKFs, etc.) manages the estimation and updates. To describe the dynamics and what parameters are represented in the state, there is a Satellite class that holds a digital twin (as close as the Estimator can make it) of the satellite. Within the Satellite, there are instances of disturbances, actuators, and sensors, that hold the properties and functions related to those specifically, as shown in Figure 1. This allows easy addition or modification of sensors, actuators, and disturbances. This makes the Satellite class useful for estimation and control, as dynamics, Jacobians, sensor covariance, and other vital functions are simple to call, even if the properties, actuators, and sensors are different between Satellites. The Satellite class itself can also be used outside of the Estimator class. For example, when testing the estimator, a "real" instantiation can be created and modeled–including with properties that differ from the Estimator's understanding to test robustness. The details and structure of these classes are described in this section.

2.2.1 Satellite

The Satellite class holds physical and mass parameters (inertia matrix J, mass m, center of mass position $\mathbf{c}$), as well as lists of sensors, actuators, and disturbances. Combining all of this information, the Satellite class has many methods to represent important aspects and simulate a spacecraft, either as the "real" satellite in a simulation or inside an estimator. These methods include:

1. Turning disturbances on or off.
2. Momentum management: generating a momentum list from actuator properties, or up-

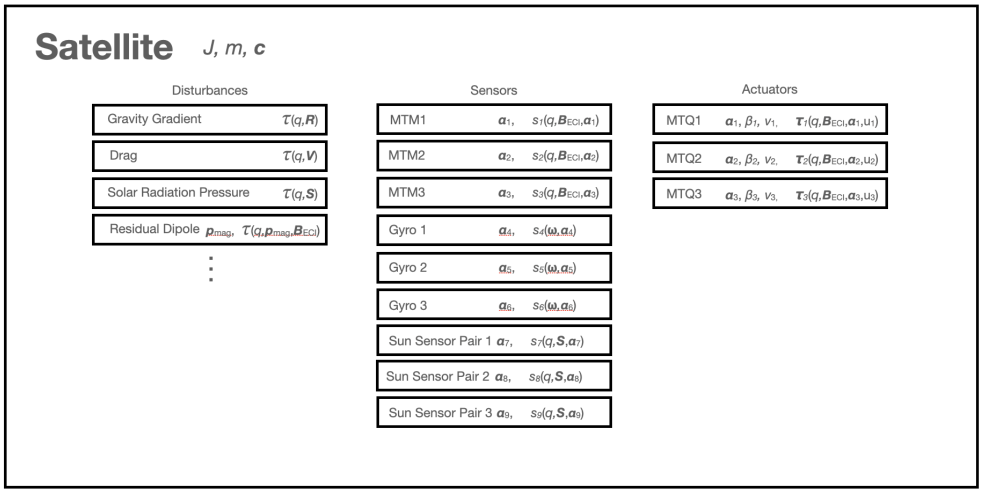
Satellite $J, m, \mathbf{c}$
Disturbances
Gravity Gradient $\tau(q,\mathbf{R})$
Drag $\tau(q,\mathbf{V})$
Solar Radiation Pressure $\tau(q,\mathbf{S})$
Residual Dipole $\mathbf{p}_{mag}$, $\tau(q,\mathbf{p}_{mag},\mathbf{B}_{ECI})$
Sensors
MTM1 $\boldsymbol{\alpha}_1$, $s_1(q,\mathbf{B}_{ECI},\boldsymbol{\alpha}_1)$
MTM2 $\boldsymbol{\alpha}_2$, $s_2(q,\mathbf{B}_{ECI},\boldsymbol{\alpha}_2)$
MTM3 $\boldsymbol{\alpha}_3$, $s_3(q,\mathbf{B}_{ECI},\boldsymbol{\alpha}_3)$
Gyro 1 $\boldsymbol{\alpha}_4$, $s_4(\boldsymbol{\omega},\boldsymbol{\alpha}_4)$
Gyro 2 $\boldsymbol{\alpha}_5$, $s_5(\boldsymbol{\omega},\boldsymbol{\alpha}_5)$
Gyro 3 $\boldsymbol{\alpha}_6$, $s_6(\boldsymbol{\omega},\boldsymbol{\alpha}_6)$
Sun Sensor Pair 1 $\boldsymbol{\alpha}_7$, $s_7(q,\mathbf{S},\boldsymbol{\alpha}_7)$
Sun Sensor Pair 2 $\boldsymbol{\alpha}_8$, $s_8(q,\mathbf{S},\boldsymbol{\alpha}_8)$
Sun Sensor Pair 3 $\boldsymbol{\alpha}_9$, $s_9(q,\mathbf{S},\boldsymbol{\alpha}_9)$
Actuators
MTQ1 $\boldsymbol{\alpha}_1$, β_1, v_1, $\boldsymbol{\tau}_1(q,\mathbf{B}_{ECI},\boldsymbol{\alpha}_1,u_1)$
MTQ2 $\boldsymbol{\alpha}_2$, β_2, v_2, $\boldsymbol{\tau}_2(q,\mathbf{B}_{ECI},\boldsymbol{\alpha}_2,u_2)$
MTQ3 $\boldsymbol{\alpha}_3$, β_3, v_3, $\boldsymbol{\tau}_3(q,\mathbf{B}_{ECI},\boldsymbol{\alpha}_3,u_3)$

Figure 1: Diagram of the Satellite class containing classes for sensors, disturbances, and actuators, with their own listed parameters, as well as satellite parameters: mass m, center of mass c, and inertia matrix J.

dating the actuator properties from given values.

3. Matching itself to a provided state (such as from the previous estimate): setting the momentum to match what is provided, setting sensor and actuator biases and disturbance parameters to match those that have been estimated, copying their expected growth/variances to match those provided. When other commands, such as dynamics, are then run, the unit then reflects the provided values. This is useful for inside estimators, as the internal satellite needs to be made to match the previous estimate. Additionally, this method turns off internal noise so the estimator can proscribe the noise components.

4. Update randomness: there are commands that update all of the random values inside the sensors, actuators, and disturbances, according to their internal update functions. Specifically, actuator noise and bias, sensor bias, and disturbance parameters can be updated. This is useful for simulating a satellite, and can be used in particle filters or Monte Carlo-based approaches.

5. Dynamics: for a given state, orbital information, and control command, find the dynamics of the satellite, including disturbances and actuator biases. The Jacobian of the state change $\dot{\mathbf{x}}$ with respect to state, control command, sensor biases (0), actuator biases, and disturbance main parameters can also be returned, as well as the Hessians of any pair of those. The Jacobians and Hessians are necessary for some estimation and control optimization techniques, such as the Extended Kalman Filter and some methods of Model Predictive Control.

6. RK4: for a given state, time step, orbital information, and control command, perform Runge-Kutta 4 integrationacross the time step using the dynamics function to estimate the next state of the satellite. This includes changing orbital parameters across the time step, as well as disturbances and actuator biases. The values of disturbance parameters and actuator biases are treated as constant across the time step. The Jacobian of the new state with respect to state, control command, sensor biases (though this is 0), actuator biases, and disturbance main parameters can also be returned, as well as the Hessians of any of those with the control command.

   As part of the state is quaternions (see Eq. 2) there is the question of how to integrate them. This approach simply numerically integrates them as vectors and normalizes at each step of RK4 (the Jacobians/Hessians of the normalization are included in the Jacobians and Hessians of this as well). It has been shown that, for well-behaved systems, this performs similarly to "proper" geometric integration.[13]

7. Control bounds: bound a given control command to the limits of the actuators.
8. Control covariance: generate the covariance in the control values, by taking the control noise variance from each actuator and creating a block diagonal matrix. This represents the covariance in executed control.
9. Sensor covariance: Generate the covariance in the sensor readings for a given list of sampled sensors, by taking the noise variance from each sensor and creating a block diagonal matrix. This represents the covariance in sensed reading.
10. Sensor information: for a given list of sampled sensors along with the base state (angular velocity and orientation, Eq. 2) and local vectors (sun position, Earth's magnetic field, etc.), the satellite can return the list of readings from those sensors. This can be used to create simulated sensor readings (including noise) or to find the expected readings in an estimator (without noise). The Jacobians of this list with respect to the base state and the sensor biases can also be found by stacking the Jacobians of each sensor. The Jacobians are vital to an EKF.

2.2.2 Estimator

The Estimator class holds the "virtual satellite" (an object of the Satellite class), which represents the Satellite as the estimator understands it. This includes what disturbances are affecting the satellite, the sensors and actuators it has, and its dynamics, what sensor and actuator biases it has, what the noise is on the sensor readings and actuators, and the evolution underpinning those actuator and sensor biases and disturbance parameters. Biases and disturbances can be fixed, and do not change in the estimated satellite, so they are included in calculations but do not vary. They can also be left out, and it will be assumed they are zero. If biases and disturbances are being estimated, however, this class updates them at each step. This internal satellite can differ from the "real" satellite used in a simulation (though the same class can be reused for that to simplify code structure). This allows for testing in simulation the effect of mis-estimated biases, rates of random change, difference in mass parameters, etc. The Estimator also holds the estimate of the satellite's full state, including the dynamics state and the biases and disturbances which are being estimated, along with the estimate's covariance and expected process noise/covariance.

There is the question of how to manage aspects of the full state that are not being used at a given time step. The momenta, orientation, and angular velocity are always used, as they define the dynamics. The actuator biases are also always used, even if the command is zero, as an actuator with a bias needs to be commanded to remain at zero. Sensor biases are also always kept on, and grow with each time step as described in Equation 14, even if the sensor is not polled in that time step. However, if a sensor is not being polled, then its value and Jacobian will be zero, and it will not be directly used in the update. This is done to keep its covariances relevant with other state elements, and because sensors are likely to be off for a comparatively brief period but will be used again with some relation to its previous value of bias.

Disturbances are the more complicated case. When a disturbance is turned off, it is not used in the estimation until it is turned back on. This includes its portion of the covariance. When turned off and on, there are a number of options for now to reset the disturbance parameter's estimate and covariance. The estimate and covariance can be set to the value they were first initialized to, or they can be saved when turned off and replaced when turned on. Finally, the estimate can be set to zero but the covariance from the saved term kept. In all cases, when turned back on, the cross-terms in the full state covariance are set to 0. What approach should be used depends on what the source of the disturbance is. For propulsion disturbances, which the authors expect to be the most-used case for a disturbance that is turned off and on, the saved value is used, as we expect the largest problems from propulsion to come from alignment between the satellites center of mass and the thrust vector, which should be roughly constant. The estimator class has shortcuts for disabling sun-related sensors and disturbances during eclipse, as well as propulsion, as these are expected to be the most frequently varying cases.

There is also the question in estimation of how to deal with quaternions–when performing the math underlying estimation, should quaternions be treated as a vector and normalized (as they are in the RK4 system in this work) or treated as a three-element parameter and adjusted in rotation space. This class allows for either, and provides a method for adding and subtracting.

The Estimator class itself does not include the estimation update method. The specific method used is from classes that inherit from the overall Estimator class. The Estimator class itself manages the input and output. Specific inherited classes that have been implemented include an Extended Kalman Filter and Unscented Kalman Filter. Other sensor fusion algorithms can be used. The estimation class handles saving the new estimation and loading the old estimation, as well as dealing with cross terms. Additionally, it automatically does not use sun sensors (but allows sun sensor pairs) when in eclipse or when their reading is beneath the standard deviation of their noise–because they are likely pointing away from the sun.

2.2.3 Actuators

The actuator class represents all types of actuators, with specific types of actuators being represented as sub-classes. Currently, there are magnetorquers (MTQs), reaction wheels (RWs), and a type of "magic" actuator that simply provides torque (this could be said to represent ideal thruster pairs, with exact torque and limitless fuel).

Each actuator has properties and methods to represent important aspects:

1. Torque generation: each actuator has an axis in the body frame that defines its axis of action, α_i. When given a command, u_i (and provided body frame vectors of external properties where relevant, such as the magnetic field for MTQs), the instance returns the torque, τ_i, created in the body frame by its actions along this axis. The default torque call includes bias and noise, as described in this list, but separate methods can find the value without noise or without either bias or noise. This is valuable for estimation) It can also return the Jacobians of torque generated relative to the "base state" (angular velocity and orientation), any stored momentum in that actuator, the command supplied, and its bias, as well as the Hessian of torque generated to any pair of those variables. There is also a maximum value of actuation that the satellite and algorithms can read to limit their commands. Generated commands above this value produce a warning.

2. Bias: each actuator can be set to have a bias or not. The bias, $\beta_{\text{act},i}$, is added to the commanded value as $u_i+\beta_{\text{act},i}$. The bias can be set directly, or evolve over time. When evolving over time, it is modeled as a discrete random walk approximating a Wiener process. Specifically, a bias rate is used, represented as $\eta_{b,\text{act},i}$. Each time the bias evolves, the time it was updated is saved as t_{prev}. When updated again at time t, the new bias is

   $$\beta'_{\text{act},i} = N(\beta_{\text{act},i}, \eta_{b,\text{act},i}(t - t_{\text{prev}})), \tag{6}$$

   where $N(\mu, \sigma)$ is the normal distribution and $\eta_{b,\text{act},i}$ is a settable parameter governing how quickly the bias changes. The bias is updated on command, though an additional setting in torque calls allows for it to be done then as well.

   Additionally, actuators have a boolean property of "estimate_bias" to describe whether this bias is being estimated, or is simply set and applied to the torque.

3. Noise: while bias changes with a low frequency, high-frequency noise is also represented. Each actuator can be set to have actuation noise or not. The noise, $\nu_{\text{act},i}$, varies over time according to a customizable noise-evolution model with various parameters. The default is simply

   $$\nu'_{\text{act},i} = N(0, \eta_{\text{act},i}), \tag{7}$$

   where $\eta_{\text{act},i}$ is a settable parameter which controls the variance of the noise and $N(\mu, \sigma)$ is again the Normal distribution with mean μ and variance σ^2. Because the functions underlying torque might be called repeatedly for one time step (such as in numerical integration) or might be called twice in a way that needs to be consistent (such as torque on body and stored momentum) noise is only updated on command, not on every call (though it can be done as part of a torque call via optional argument). Noise is added to the command and bias as $u_i + \beta_{\text{act},i} + \nu_{\text{act},i}$.

4. Momentum storage: Each actuator is assigned to have momentum stored or not. It then has a moment of inertia around its axis, $j_{\text{act},i}$ and a maximum momentum that can be stored. There is also a set of functions similar to the torque generation in this list, but corresponding to the torque applied to the momentum storage, $\tau_{s,i}$ with the accompanying Jacobians and Hessians. Finally, it has a noise term that works like the sensor noise terms when it comes to sensing the current momentum stored (note that sensing bias is NOT implemented for this, but torque bias and noise are applied to this

as well as the body torque).

Specifically, body torque and torque on momentum storage are calculated for each actuator, with the ith actuator assumed to be the correct class, as:

- Magnetorquer on axis $\hat{\alpha}_{\text{act},i}$:

$$\tau_i = \left(A^T(\bar{q})\mathbf{B}_{\text{ECI}}\right) \times \hat{\alpha}_{\text{act},i}\left(u_i + \beta_{\text{act},i} + \nu_{\text{act},i}\right) \tag{8}$$

$$\tau_{\text{s,i}} = 0 \tag{9}$$

- Thrusters on axis $\hat{\alpha}_{\text{act},i}$:

$$\tau_i = \hat{\alpha}_{\text{act},i}\left(u_i + \beta_{\text{act},i} + \nu_{\text{act},i}\right) \tag{10}$$

$$\tau_{\text{s,i}} = 0 \tag{11}$$

- RWs on axis $\hat{\alpha}_{\text{act},i}$:

$$\tau_i = \hat{\alpha}_{\text{act},i}\left(u_i + \beta_{\text{act},i} + \nu_{\text{act},i}\right) \tag{12}$$

$$\tau_{\text{s,i}} = -\left(u_i + \beta_{\text{act},i} + \nu_{\text{act},i}\right) \tag{13}$$

This approach can also be extended to other actuators, such as control moment gyroscopes.

2.2.4 Sensors

The sensor class represents all types of sensor, with specific types being represented as sub-classes. Currently, in the tool as developed, there are magnetometers (MTMs), gyroscopes (gyros), GPS (not used in this discussion which focuses on attitude), sun sensors, and sun sensor pairs.

Each sensor has properties and methods to represent important aspects:

1. Reading: each sensor takes a reading of some property in the body frame, s_i. The reading call includes bias and noise, as described in this list, but separate methods can find the value without noise or without either bias or noise. It can also return the Jacobians of the reading generated relative to the "base state" (angular velocity and orientation, plus any stored angular momentum) and the sensor's bias, which are necessary for an EKF.

2. Bias: each sensor can be set to have a bias or not. The bias, $\beta_{\text{sens},i}$, is added to the basic reading value as $s_i = z_i + \beta_{\text{sens},i}$. The bias can be set directly, or evolve over time. When evolving over time, it is modeled as a discrete random walk approximating a Wiener process. Specifically, a bias rate is used, represented as $\eta_{b,\text{sens},i}$. Each time the bias evolves, the time it was updated is saved as t_{prev}. When updated again at time t, the new bias is

$$\beta'_{\text{sens},i} = N\left(\beta_{\text{sens},i}, \eta_{b,\text{sens},i}(t - t_{\text{prev}})\right), \tag{14}$$

where $N(\mu, \sigma)$ is the normal distribution. The bias is updated on reading.

Additionally, sensors have a boolean property of "'estimate_bias" to describe whether this bias is being estimated, or is simply set and applied to the reading.

3. Noise: while bias changes with a low frequency, there is also high-frequency noise represented. Each sensor can be set to have noise or not. The noise, $\nu_{\text{sens},i}$, varies over time according to a customizable noise-evolution model with various parameters. The default is simply

$$\nu'_{\text{sens},i} = N(0, \eta_{\text{sens},i}), \tag{15}$$

where $\eta_{\text{sens},i}$ is a settable parameter. The way that the noise function updates itself is customizable to allow for slowly changing parameters. Noise is generated and added on every call of a noisy reading (not on clean readings, or readings with only bias, and this is important for the UKF). By default, noise is added to the reading and bias as $s_i = z_i + \beta_{\text{sens},i} + \nu_{\text{sens},i}$, but this is customizable.

Readings are calculated for each sensor, designated with index i, as:

- MTMs with axis $\hat{\alpha}_{\text{sens},i}$

$$s_i = \left(\hat{\alpha}^T_{\text{sens},i}A^T(\bar{q})\mathbf{B}_{\text{ECI}}\right) + \beta_{\text{sens},i} + \nu_{\text{sens},i} \tag{16}$$

- Gyros with axis $\hat{\alpha}_{\text{sens},i}$

$$s_i = \left(\hat{\alpha}^T_{\text{sens},i}\omega\right) + \beta_{\text{sens},i} + \nu_{\text{sens},i} \tag{17}$$

- Sun sensor with efficiency ζ_i and axis $\hat{\alpha}_{\text{sens},i}$:

$$s_i = \zeta_i \max\left(0, \left(\hat{\alpha}^T_{\text{sens},i}A^T(\bar{q})\frac{\mathbf{S}_{\text{ECI}}}{\|\mathbf{S}_{\text{ECI}}\|}\right)\right) + \beta_{\text{sens},i} + \nu_{\text{sens},i} \tag{18}$$

- Sun sensor pair with efficiency $\zeta_{i,\pm 1}$ and anti-parallel axes $\pm\hat{\alpha}_{\text{sens},i}$:

$$s_i = \zeta_{i,m}\left(\hat{\alpha}^T_{\text{sens},i}A^T(\bar{q})\frac{\mathbf{S}_{\text{ECI}}}{\|\mathbf{S}_{\text{ECI}}\|}\right) + \beta_{\text{sens},i} + \nu_{\text{sens},i}, \tag{19}$$

where $m = \text{sgn}\left(\hat{\alpha}^T_{\text{sens},i}A^T(\bar{q})\hat{s}\right)$

This approach can also be extended to star trackers or cameras and Earth horizon sensors.

2.2.5 Disturbances

The disturbance class represents all types of disturbances, with specific types being represented as sub-classes. Currently, there is gravity gradient, drag, solar radiation pressure, residual dipole, propulsion torque, and a general disturbance torque. Gravity gradient, drag, and SRP are always modeled analytically. The other three have a specific parameter (moment for dipole, torque for propulsion or general disturbance) that can be set to a constant or estimated (it can also be allowed to vary over time in simulation). Each disturbance has properties and methods to represent import aspects:

1. Torque generation: each disturbance has a formula, based on those in Fitzgerald,[14] that return the torque in the body frame. It can also return the Jacobians of torque generated relative to the orientation. Torques that have a varying parameter, $p_{\text{dist},i}$, also have a way to calculate the Jacobian relative to that parameter, which is necessary for the EKF. It can also find the Hessian with respect to orientation and the parameter.

2. Time evolution: For the disturbances that vary over time (propulsion, generalized, and inherent dipole), if they are set to be time-varying, they can evolve when an update command is called. When evolving over time, they are modeled as a discrete random walk approximating a Wiener process. Specifically, a rate is used, represented as $\eta_{\text{dist},i}$. Each time the parameter evolves, the time it was updated is saved as t_{prev}. When updated again at time t, the new parameter value is

$$\mathbf{p}'_{\text{dist},i} = N_3\left(\mathbf{p}_{\text{dist},i}, I_{3\times3}\left(\eta_{\text{dist},i}(t-t_{\text{prev}})\right)^2\right), \tag{20}$$

   where $N_x(\mu, \sigma^2)$ is the x-variate normal distribution. Some disturbances also have the magnitude of their main parameter capped at a maximum value, as a random walk can veer off towards infinity but there is a limit to the torque a propulsion system can create or the strength of a satellite's inherent dipole. If so, the new value is simply divided by the norm, then times this maximum value.

3. Active/not: disturbances have an "active" parameter, which allows them to be turned on and off throughout a run or mission. When off (active=False), the torque and relevant Jacobians/Hessians return 0.

The disturbances can also be used in control algorithms, by calculating the disturbance torque and accounting for it in the control law. This is especially helpful for something like the general disturbance torque, which can eliminate the need for an integral term in some control algorithms.

2.3 Kalman Control Noise

In this work, control noise is separated from process noise in the Extended and Unscented Kalman Filters. The implementation of control noise in the UKF simply adds sigma points corresponding to the variance in control. The EKF, however, requires the control noise to be added with the state dynamics Jacobians. Consider:

$$\dot{\mathbf{x}} = \mathbf{f}\left(\mathbf{x}, \mathbf{u} + \nu_{\text{control}}, t\right). \tag{21}$$

where $\mathbf{x}$ is the state, $\mathbf{u}$ is the control, ν_{control} is the control noise, and $\mathbf{f}$ represents the time evolution of the state. The t component captures things that vary with orbit, like atmospheric density and the magnetic field.

Discretizing Equation 21 between time steps:

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{x}_k + \int_{t_k}^{t_{k+1}} \mathbf{f}\left(\mathbf{x}(t), \mathbf{u}(t) + \nu_{\text{control}}(t), t\right) dt \\ &= g\left(\mathbf{x}_k;\ t_k, t_{k+1};\ \mathbf{u}_k + \tilde{\nu}_{\text{k}}\right) + \nu_{\text{int,k}}\end{aligned} \tag{22}$$

where $\mathbf{u}$ is treated as constant throughout the integration step, $\tilde{\nu}$ is a noise value that corresponds to the time-integration of the random process ν_{control} over the given time interval, g is the selected method of numerical integration, and $\nu_{\text{int,k}}$ is the error from numerical integration. For clarity, we will write $g\left(\mathbf{x}_k;\ t_k, t_{k+1};\ \mathbf{u}_k + \tilde{\nu}_{\text{k}},\right) = g_k\left(\mathbf{x}_k;\ \mathbf{u}_k + \tilde{\nu}_{\text{k}},\right)$.

We thus model the predictive step in an EKF as:

$$\hat{\mathbf{x}}_{k+1|k} = E\left[g_k\left(\mathbf{x}_k;\ \mathbf{u}_k + \tilde{\nu}_{\text{k}}\right) + \nu_{\text{int,k}}\right] = g_k\left(\hat{\mathbf{x}}_{k|k};\ \mathbf{u}_k\right), \tag{23}$$

assuming the control noise and integration noise are zero-mean. The covariance of this is

$$\begin{aligned}P_{k+1|k} &= E\left[\mathbf{x}_{k+1}\mathbf{x}_{k+1}^T\right] - E\left[\mathbf{x}_{k+1}\right]E\left[\mathbf{x}_{k+1}\right]^T \\ &= E\left[\mathbf{x}_{k+1}\mathbf{x}_{k+1}^T\right] - \hat{\mathbf{x}}_{k+1|k}\hat{\mathbf{x}}_{k+1|k}^T\end{aligned} \tag{24}$$

Applying Equation 22 gives

$$P_{k+1|k} = E\left[(g_k(\mathbf{x}_k;\ \mathbf{u}_k + \tilde{\nu}_\mathrm{k}))(g_k(\mathbf{x}_k;\ \mathbf{u}_k + \tilde{\nu}_\mathrm{k}))^T\right] + \mathrm{cov}(\nu_\mathrm{int,k}) - \hat{\mathbf{x}}_{k+1|k}\hat{\mathbf{x}}_{k+1|k}^T, \tag{25}$$

assuming the integration error is independent of the state and control error. We approximate the system as

$$P_{k+1|k} = E\left[(g_k(\mathbf{x}_k;\ \mathbf{u}_k) + B_k\tilde{\nu}_\mathrm{k})(g_k(\mathbf{x}_k;\ \mathbf{u}_k) + B_k\tilde{\nu}_\mathrm{k})^T\right] + \mathrm{cov}(\nu_\mathrm{int,k}) - \hat{\mathbf{x}}_{k+1|k}\hat{\mathbf{x}}_{k+1|k}^T, \tag{26}$$

where B_k is the Jacobian of $g_k(\mathbf{x}_k;\ \mathbf{u}_k)$ with respect to $\mathbf{u}_k$. Our system can then be rewritten, treating the control error as zero-mean:

$$P_{k+1|k} = E\left[g_k(\mathbf{x}_k;\ \mathbf{u}_k)\, g_k(\mathbf{x}_k;\ \mathbf{u}_k)^T\right] + B_k\mathrm{cov}(\tilde{\nu}_k)\, B_k^T + \mathrm{cov}(\nu_\mathrm{int,k}) - \hat{\mathbf{x}}_{k+1|k}\hat{\mathbf{x}}_{k+1|k}^T, \tag{27}$$

We use $U_k = \mathrm{cov}(\tilde{\nu}_k)$ and $Q_k = \mathrm{cov}(\nu_\mathrm{int,k})$. Linearizing around $\hat{\mathbf{x}}_{k|k}$, we have

$$P_{k+1|k} = E\left[g_k(\hat{\mathbf{x}}_{k|k} + \delta\mathbf{x}_k;\ \mathbf{u}_k)\, g_k(\hat{\mathbf{x}}_{k|k} + \delta\mathbf{x}_k;\ \mathbf{u}_k)^T\right] + B_kU_kB_k^T + Q_k - \hat{\mathbf{x}}_{k+1|k}\hat{\mathbf{x}}_{k+1|k}^T, \tag{28}$$

Approximating with G_k as the derivative of $g_k(\mathbf{x}_k;\ \mathbf{u}_k)$ with respect to $\mathbf{x}_k$:

$$P_{k+1|k} = E\left[g_k(\hat{\mathbf{x}}_{k|k};\ \mathbf{u}_k)\, g_k(\hat{\mathbf{x}}_{k|k};\ \mathbf{u}_k)^T\right] + G_k\mathrm{cov}(\delta\mathbf{x}_k)\, G_k^T + B_kU_kB_k^T + Q_k - \hat{\mathbf{x}}_{k+1|k}\hat{\mathbf{x}}_{k+1|k}^T, \tag{29}$$

Because $\delta\mathbf{x}_k = \mathbf{x}_k - \hat{\mathbf{x}}_{k|k}$, we have $\mathrm{cov}(\delta\mathbf{x}_k) = P_{k|k}$. As we have accounted (with varying assumptions) for all sources of error, we have $E\left[g_k\left(\hat{\mathbf{x}}_{k|k};\ \mathbf{u}_k\right) g_k\left(\hat{\mathbf{x}}_{k|k};\ \mathbf{u}_k\right)^T\right] = g_k\left(\hat{\mathbf{x}}_{k|k};\ \mathbf{u}_k\right) g_k\left(\hat{\mathbf{x}}_{k|k};\ \mathbf{u}_k\right)^T = \hat{\mathbf{x}}_{k+1|k}\hat{\mathbf{x}}_{k+1|k}^T$ exactly, so we find:

$$P_{k+1|k} = G_kP_{k|k}G_k^T + B_kU_kB_k^T + Q_k, \tag{30}$$

which is similar to the conventional EKF expression, with only an added control noise term.

2.3.1 Overall equation components from states

It is worth noting that Equations 23-30 include control error, error in state estimation, integration error, and disturbing torques. If the model is used and the disturbing torques are erroneous (or a general disturbance is not included in the state but is present in reality), then Equation 22 would need to include a noise torque term. Assuming this is randomly distributed between times, independent of other errors, and zero-mean, that torque noise can be treated similarly to the $\nu_{int,k}$ term. (If these assumptions are incorrect, then it should be approximated or represented as a generalized torque, as described in Section 2.2.5.) The equations are the same, except with $Q_k = \mathrm{cov}(\nu_\mathrm{int,k}) + HJ_{x,\omega}\mathrm{cov}\left(\int_{t_k}^{t_{k+1}} \tau_\mathrm{dist}(t)dt\right) J_{x,\omega}^T H^T$ or $Q_k = \mathrm{cov}(\nu_\mathrm{int,k}) + C_kT_kC_k^T$, where T_k is the covariance integral of the unestimated torque and C_k is derivative of g_k with respect to a torque term; it is equal to the columns of G_k corresponding to a generalized disturbance torque.

3 Results

This framework and estimator are tested in several ways. First, in Cases A and B (Sections 3.1 and 3.2), the approach from this work is compared against a well-regarded traditional estimator. It is tested for a (simulated) large satellite on which the traditional estimator is demonstrated. The difference between Cases A and B is that Case A starts with larger errors and thus shows convergence, while Case B has smaller initial errors and is more representative of behavior once the estimators have converged. Cases C and D (Sections 3.3 and 3.4) then show the behavior of the dynamics-aware filter on a nanosatellite, with worse sensors and less inertia. Case C represents the system behavior when starting with large error and converging. Case D shows the behavior when starting with small error and continuing. Cases E and F are used to explore the importance of certain aspects of the dynamics-aware filter. Specifically, Case E shows the effects of ignoring the actuator bias or the disturbances, while Case F demonstrates the importance of considering a propulsion torque, when present, and shows how well the estimator can converge on the torque value.

3.1 Case A: TRMM Satellite with Initial Attitude and Bias Errors

This system is first tested against a simple baseline attitude filter under a variety of conditions. The baseline is the simple Unscented QUaternion Estimator (USQUE) by Crassidis and Markley.[10] There are more involved and mission-specific filters, including versions of the unscented attitude filter that account for sensor bias and other factors, but this is a well-behaved and generalizable filter that could easily be the basis of an approach for a nanosatellite project. The test cases in the USQUE paper[10] are recreated, with the addition of all disturbances (except propulsion) described in,[14] as well as active magnetic control with MTQs, and biases on the sensors and actuators.

Specifically, a Satellite object is created to mimic the Tropical Rainfall Measuring Mission (TRMM) used in.[10] The satellite has the properties:

- a mass of $m = 3000$ kg;
- an inertia matrix of $J = 500 * \mathrm{diag}\left(\begin{bmatrix}1 & 3 & 3\end{bmatrix}\right)$kg m^2;
- three-axis MTQs with an arbitrarily-chosen initial bias of $\beta_{\mathrm{mtq}} = 5 * \begin{bmatrix}\frac{1}{3\sqrt{2}} & \frac{1}{3\sqrt{2}} & \frac{4}{3\sqrt{2}}\end{bmatrix}$, a maximum magnetic moment of 100 Am^2 for each MTQ, a noise standard deviation of $\eta_{\mathrm{mtq}} = 0.0001$ Am^2 $\cdot$s$^{0.5}$, and a bias drift rate of $\eta_{\mathrm{b,mtq}} = 0.000001$ Am^2/s$^{0.5}$;
- three-axis gyroscope with an initial bias of $\beta_{\mathrm{gyro}} = \begin{bmatrix}0.1 & 0.1 & 0.1\end{bmatrix}$ deg/hr (mimicking the second test case in[10]), a noise standard deviation of $\eta_{\mathrm{gyro}} = 0.31623$ μrad/s$^{0.5}$, and a bias drift rate of $\eta_{\mathrm{b,gyro}} = 3.1623 \times 10^{-4}$ μrad/s$^{1.5}$;
- three-axis magnetometer with an initial bias of $\beta_{\mathrm{mtm}} = \begin{bmatrix}-0.9948 & -0.0199 & -0.0995\end{bmatrix}$ nT, a noise standard deviation of $\eta_{\mathrm{mtm}} = 50$ nT $\cdot$ s$^{0.5}$, and a bias drift rate of $\eta_{\mathrm{b,mtm}} = 1$ nT/s$^{0.5}$;
- three pairs of coarse sun sensors each on axis, with efficiency of 0.3 and assuming no degradation, an initial bias of $\beta_{\mathrm{sun}} = \begin{bmatrix}0.015 & 0.027 & -0.009\end{bmatrix}$ (there are no units on this), a noise standard deviation of $\eta_{\mathrm{sun}} = 0.0003$ s$^{0.5}$, a bias drift rate of $\eta_{\mathrm{b,sun}} = 0.000003$ s$^{-0.5}$; and
- a residual dipole with initial value $\begin{bmatrix}0.5 & 0.001 & 2\end{bmatrix}$ Am^2, capped maximum value of 2 Am^2, and a drift rate of $\eta_{\mathrm{dipole}} = 0.0001$ Am^2/s$^{0.5}$.

The satellite is simulated in an approximately circular orbit (initial position of 7200 km along the ECI x-axis and an initial velocity of 7.4 km/s perpendicular to this), inclined at 35°. The effects of J2 on the orbit are considered. The simulation is run with a time step based on the update period for the control, sensing, and estimation (either 1 or 10 seconds). The "real" dynamics are integrated using an RK45 variable time-step solver between time steps. Sensor and actuator biases, as well as disturbances, are updated as described in, respectively, Sections 2.2.4, 2.2.3, and 2.2.5, with random distributions. The simulation runs for either 12 ($\Delta t = 1$ s) or 24 hours ($\Delta t = 10$ s). To simulate conditions with active control, a simple magnetic PD algorithm based on Lovera[15] is applied (with a proportional gain of 10 and a velocity gain of 1000). Specifically, for the first 30 minutes, no control is applied. From then until the 2-hour mark, the basic b-dot algorithm[16] is applied using numeric differentiation between sensor readings and a gain of 10^{10}. For the remainder of the time, the Lovera PD algorithm is applied, with a quaternion goal of $\begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}$ until the 6-hour mark (3 hours in case of 12-hour run), then $\begin{bmatrix}0 & 0 & 1 & 0\end{bmatrix}$ until 12 hours (6 hours in case of 12-hour run), $\begin{bmatrix}0 & 1 & 0 & 0\end{bmatrix}$ until 18 (9 hours in case of 12-hour run), and $\begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}$ again until the end at 24 hours or 12 hours, creating a rather demanding control schedule.

In addition to the already-given initial biases, the relevant real state components are an initial angular velocity of $\omega(0) = \left[0, \frac{1}{15}, 0\right]$ deg/s (this is roughly 1 y-axis rotation per orbit), and an initial quaternion of $\begin{bmatrix}0.63 & -0.33 & -0.63 & 0.33\end{bmatrix}$ (which is the orientation such that the body z-axis is pointing nadir and body x-axis is pointing ram).

An estimated Satellite object is also created. It has identical mass properties to the "real" satellite, but does not have modeled actuator bias or biases on magnetometers and sun sensors. The standard deviation of the noise on the sensors is identical to the "real" satellite. Initial estimated gyro bias is set as $\begin{bmatrix}0 & 20.1 & 0\end{bmatrix}$ deg/hr (mimicking the second test case in[10]), and the correct gyro bias drift rate is used. The other components of the estimated state are an initial angular velocity (though this is not used, due to the nature of USQUE filter) of $\hat{\omega}(0) = \begin{bmatrix}0 & 0 & 0\end{bmatrix}$ deg/s, and an initial quaternion of $\begin{bmatrix}-0.16 & -0.85 & 0.31 & 0.39\end{bmatrix}$ (this is similar to the second case in,[10] as it is rotations of -50, 50, and 160 degrees around the x, y, and z-axes (calculated as a Modified Rodrigues Parameter) from the correct orientation.

The covariance on the initial estimate has no cross-terms. As in,[10] $P_{\text{att}} = (50^\circ)^2$ and $P_{\text{gyro bias}} = (20^\circ/\text{hr})^2$. The integration covariance is calculated as in.[10] For comparison with expected values on the magnetometer and sun sensor, the position of the satellite is assumed to be known exactly. The magnetic field is calculated with the 13th generation IGRF (up to degree $N = 13$) and the sun position information with the python package skyfield. Disturbances are modeled with exact information. However, there may be discrepancies between the disturbance experienced by the "real" satellite and the virtual satellite based on error in the orientation estimation. For example, if the estimated orientation is such that major principal axis of the satellite is pointing nadir, it will estimate zero gravity gradient torque, while the "real" satellite may experience a torque corresponding to its actual orientation.

The USQUE was compared against an instantiation of the filter described throughout this work, in an identical simulation. The estimated Satellite object for this filter has identical mass properties to the real satellite, and models actuator and sensor biases (and a dipole, when relevant). The standard deviation of the noise on sensors is identical to the real satellite. Initial state estimates are the same as in the USQUE case, with assumed zeros for the biases. The correct bias (and dipole) drift rates are used.

The covariance on the initial estimate has no cross-terms. The initial covariance estimates are $P_\omega = P_{\text{gyro bias}} = (20^\circ/\text{hr})^2$, $P_{\text{att}} = (50^\circ)^2$, $P_{\text{mtq bias}} = (5\text{Am}^2)^2$, $P_{\text{mtm bias}} = (1\text{nT})^2$, $P_{\text{sun bias}} = (0.03)^2$, and $P_{\text{dipole}} = (1\text{Am}^2)^2$. The integration covariance for actuators, sensors, and biases with randomness are identical to drift rates used in the random calculation times the time step Δt. The integration covariance on attitude is $10^{-12}(\Delta t)^2$ rad^2 and on the angular velocity it is $10^{-17}(\Delta t)$ rad^2/s^2. These values were selected based on experimentation. For the unscented filter parameters, a value of $\alpha = 1$, $\beta = 2$, and $\kappa = 3 - (L + 3)$ (with the additional 3 to account for control noise and L is the state length).

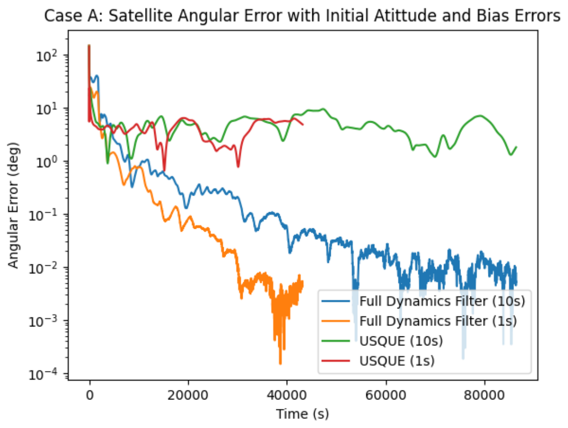

Figure 2: Angular estimation error in Case A (Section 3.1), showing quicker convergence and higher accuracy with the Dynamics-Aware Filter.

Figure 2 shows the overall angular error of the different methods (the dynamics-aware filter in this work vs. USQUE, and Δt of 10 s and 1 s) on this test. The USQUE approach stays between 1 and 10 degrees of error. The dynamics-aware filter, on the other hand, continues to improve, until it eventually reaches approximately 0.01 degrees of error. This pattern is true in both the 1 s and 10 s cases, though the 1 s case converges faster. Figure 3 shows breakdown of this angular error by axis into a Modified Rodrigues Parameter (MRP). The plot is bounded at $\pm 1^\circ$ of error to show the behavior of the dynamics-ware filter. The dynamics-aware filter converges quickly and evens out to near-zero error, while the USQUE oscillates dramatically, including well past 1 degree and thus past the bounds of the plot. As can be seen in Figure 4, the dynamics-aware filter is also more accurate on angular velocity. Figure 4 shows the magnitude of angular velocity error. The USQUE has around 0.001 $^\circ$/s of error, while the dynamics-aware filter continues to converge further to around 0.00001 $^\circ$/s of error. This is true in both the 1 s and 10 s timestep cases, though again the 1 s case converges faster. Overall, the dynamics-aware filter outperforms the standard USQUE filter, as it accounts for the control and disturbances governing the system's movement.

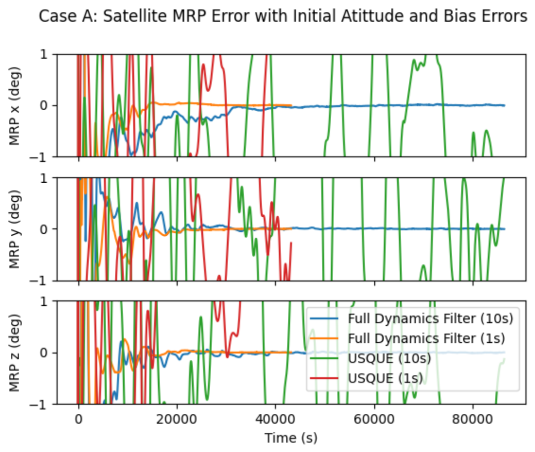

Figure 3: Case A (Section 3.1) orientation error, expressed as Modified Rodrigues Parameter (MRP), showing faster convergence and higher accuracy for the Dynamics-Aware Filter. Notably, the error in the USQUE cases exceeds 1 degree and thus goes past the edge of the plot.

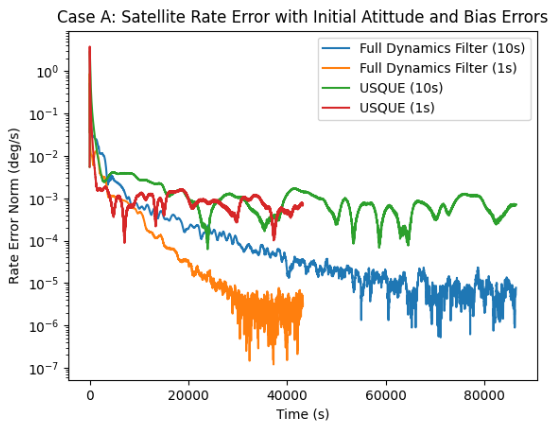

Figure 4: Case A (Section 3.1) angular velocity error, showing faster convergence and higher accuracy for the Dynamics-Aware Filter.

3.2 Case B: TRMM Satellite with Only Small Initial Bias Errors

The tests from Case A, in Section 3.1, were also repeated with a "close" start. This is to match the third example in[10] and to show what happens over time, after the filters have converged. The satellite, orbit, and most parameters are the same. The difference is how accurate the estimator's initial estimates are. In this case, the initial estimated angular velocity and attitude are exactly correct, the initial estimate of gyro biases is all zeros, and $P_{\text{att}} = (0.5°)$. For the dynamics-aware filter from this work, $P_\omega = (0.5°/\text{min})^2$.

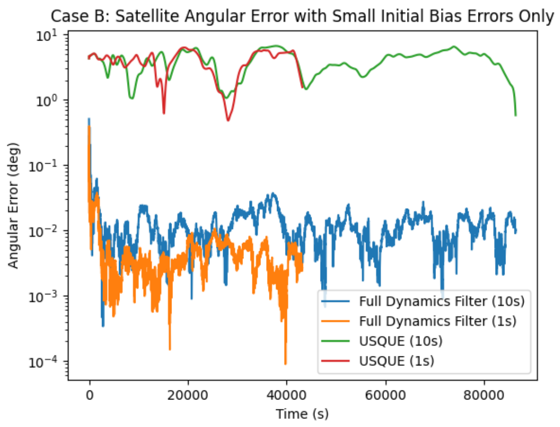

Figure 5: Angular estimation error in Case B (Section 3.2), showing higher accuracy with the Dynamics-Aware Filter. With the "close" initial estimate, both filters converge very quickly.

The overall angular error of this "close" case is shown in Figure 5. The USQUE approach has single-digit degrees of error, while the dynamics filter again reaches 0.01 degrees of error, supporting the results from Case A in Figure 2, and showing that both approaches are stable once converged. With the close initial estimate, the filter converges much more quickly. Figure 6 shows the per-axis breakdown of this angular error, represented as a Modified Rodrigues Parameter (MRP). The axes are each capped at $\pm 0.1°$ to show the beahvior of the dynamics-aware filter, which remains close to zero, with very small error. The USQUE results exhibit a much greater error, passing the bounds of the plot. Figure 7 shows the magnitude of the angular velocity error. Again, both the 10 s and 1 s results are similar for both USQUE and the dynamics-aware filter. The USQUE has around 0.001 °/s of error while the dynamics-aware filter has about 0.00001 °/s of error. Overall, the dynamics-aware filter outperforms the standard USQUE filter, as it accounts for the control and disturbances governing the system's movement.

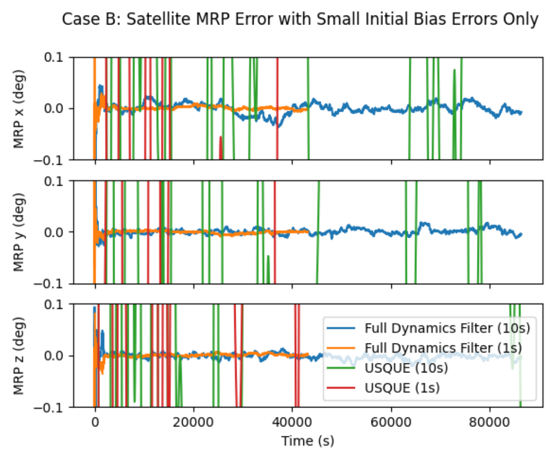

Figure 6: Case B (Section 3.2) orientation error, expressed as Modified Rodrigues Parameter (MRP), showing higher accuracy for the Dynamics-Aware Filter. Notably, the error in the USQUE cases exceeds 0.1 degree and thus goes past the edge of the plot.

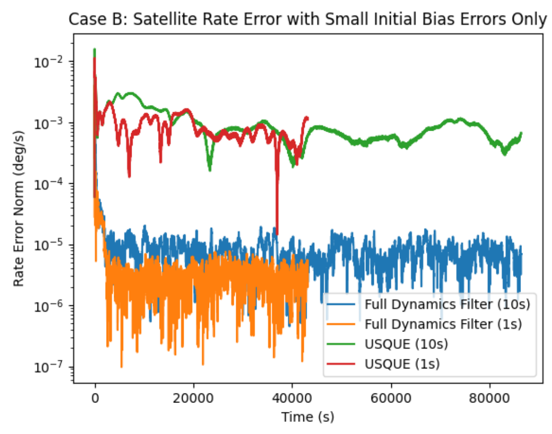

Figure 7: Case B (Section 3.2) angular velocity error, showing higher accuracy for the Dynamics-Aware Filter.

3.3 Case C: CubeSat with Initial Bias and Attitude Errors

The dynamics-aware filter is perhaps most useful for nanosatellites where the sensors may be significantly worse than those in Sections 3.1 and 3.2. With such sensors, tracking the angular velocity and disturbances alongside the gyroscope bias increases the information available to the filter by incorporating knowledge of the dynamics and disturbances. Thus, other tests were run benchmarking this filter in a cubesat test case.

Specifically, a Satellite object was created with:

- a mass of $m = 4$ kg;
- an inertia matrix of

$$J = \begin{bmatrix} 0.0314 & 5.9\times10^{-5} & -0.0067 \\ 5.9\times10^{-5} & 0.0341 & -0.0001 \\ -0.0067 & -0.0001 & 0.01005 \end{bmatrix}$$

  kgm^2;
- three-axis MTQs with arbitrarily-chosen error parameters: an initial bias of $\beta_{\text{mtq}} = 0.05 * \begin{bmatrix} \frac{1}{3\sqrt{2}} & \frac{1}{3\sqrt{2}} & \frac{4}{3\sqrt{2}} \end{bmatrix}$, a maximum magnetic moment of 1 Am^2 for each MTQ, a noise standard deviation of $\eta_{\text{mtq}} = 0.0001$ Am^2 $\cdot$s$^{0.5}$, and a bias drift rate of $\eta_{\text{b,mtq}} = 0.000001$ Am^2/s$^{0.5}$;
- three-axis gyroscope with an initial bias of $\beta_{\text{gyro}} = 0.1 * \begin{bmatrix} \frac{1}{\sqrt{11}} & \frac{-1}{\sqrt{11}} & \frac{3}{\sqrt{11}} \end{bmatrix}$ deg/s, a noise standard deviation of $\eta_{\text{gyro}} = 0.0004$ deg/s$^{0.5}$, and a bias drift rate of $\eta_{\text{b,gyro}} = 0.03$ deg/s$^{1.5}$, similar to MEMS gyros[17];[18]
- three-axis magnetometer with an initial bias of $\beta_{\text{mtm}} = \begin{bmatrix} -9.948 & -0.199 & -0.995 \end{bmatrix}$ nT, a noise standard deviation of $\eta_{\text{mtm}} = 300$ nT $\cdot$ s$^{0.5}$,[18] and a bias drift rate of $\eta_{\text{b,mtm}} = 1$ nT/s$^{0.5}$;
- three pairs of coarse solar sensors each axis, with efficiency of 0.3 and assumed no degradation, an initial bias of $\beta_{\text{sun}} = \begin{bmatrix} 0.015 & 0.027 & -0.009 \end{bmatrix}$ (there are no units on this), a noise standard deviation of $\eta_{\text{sun}} = 0.0003$ s$^{0.5}$, and a bias drift rate of $\eta_{\text{b,sun}} = 0.000003$ s$^{-0.5}$.

The satellite is simulated in an approximately circular orbit (initial position of 7000 km along the ECI x-axis and an initial velocity of 8 km/s perpendicular to this), inclined at 45°. The effects of J2 on the orbit are considered. The simulation is run with a time step based on the update period for the control, sensing, and filter of 1 second. The "real" dynamics are integrated using an RK45 variable timestep solver between time steps. Sensor and actuator biases, as well as disturbances, are updated as described in Sections 2.2.4, 2.2.3, and 2.2.4, respectively, with random distributions. The simulation runs for 3 hours. To simulate real-world conditions with control, a simple magnetic PD algorithm based on Lovera[15] is applied (with a proportional gain of 10 and a velocity gain of 1000). Specifically, for the first 5 minutes, no control is applied. From then until the 15-minute mark, the basic b-dot algorithm is applied using numeric differentiation between sensor readings and a gain of 10^6. For the remainder of the time, the Lovera PD algorithm is applied, with a quaternion goal of $\begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}$ until 25% of the way through, then $\begin{bmatrix}0 & 0 & 1 & 0\end{bmatrix}$ until halfway, $\begin{bmatrix}0 & 1 & 0 & 0\end{bmatrix}$ until 75% , and $\begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}$ again until the end.

In addition to the already-given initial biases, the relevant real state components are an initial angular velocity of $\omega(0) = \begin{bmatrix}-0.00698 & 0.00563 & -0.01497\end{bmatrix}$ deg/s, and an initial quaternion of $\begin{bmatrix}0.897 & -0.391 & 0.164 & -0.128\end{bmatrix}$.

This satellite is simulated and its orientation estimated by the dynamics-aware filter. The estimated Satellite object has identical mass properties to the "real" satellite, and models actuator and sensor biases (and propulsion, when relevant). The standard deviation of the noise on sensors is identical to the "real" satellite. The initial estimate for all biases and propulsion is 0. The correct bias (and propulsion) drift rates are used. The initial angular velocity is estimated to be 0 and the initial quaternion is estimated to be $\begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}$.

The covariance on the initial estimate has no cross-terms. The initial covariance estimates are $P_\omega = P_{\text{gyro bias}} = (1°/\text{s})^2$, $P_{\text{att}} = 10$ rad^2, $P_{\text{gyro bias}} = (0.2°/\text{s})^2$, $P_{\text{mtq bias}} = (0.1\text{Am}^2)^2$, $P_{\text{mtm bias}} = (100\text{nT})^2$, and $P_{\text{sun bias}} = (0.03)^2$.The integration covariance for actuators, sensors, and biases with randomness are identical to drift rates used in the random calculation times the time step Δt (which is 1 second). The integration covariance on attitude is $10^{-12}(\Delta t)^2$ rad^2 and on the angular velocity it is $10^{-17}(\Delta t)$ rad^2/s^2. These values are selected based on experimentation. For the unscented filter parameters, a value of $\alpha = 1$, $\beta = 2$, and $\kappa = 3 - (L + 3)$ are used, with the additional 3 to account for control noise added to the state length L.

For comparison with expected values on the magnetometer and sun sensor, the position of the satellite is assumed to be known exactly. The magnetic field is calculated using the 13th generation IGRF model (up to order $N = 13$) and the sun position information with the python module skyfield. Disturbances are modeled with exact information, but their value may vary based on discrepancies between the "real" and estimated orientations.

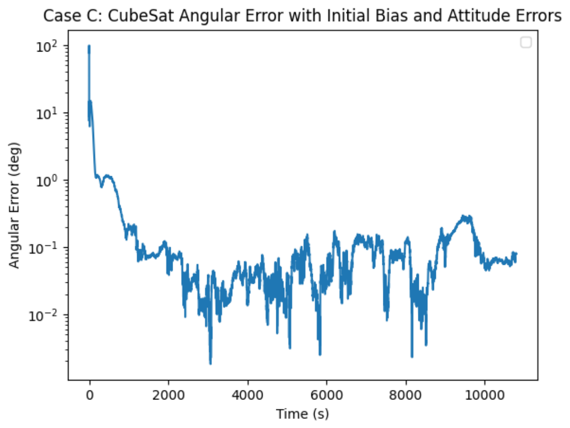

Figure 8: Case C (Section 3.3) showing the estimator orientation estimate converging to within 1 degree (usually close to 0.1 degree) on a CubeSat with simple sensors.

Under these conditions, the dynamics-aware filter described in this work performs admirably, despite the poor information from sensors. As shown in Figure 8, it achieves roughly 0.1 degree estimation accuracy (and always within 1 degree), despite the disturbances, constant motion, and poor sensor quality. It also shows that the filter converges in less than an hour, roughly 3000 seconds. The breakdown of this angular error per-axis (using Modified Rodrigues parameters) can be seen in Figure 9. This more clearly shows that the angular estimation error around each axis is close to 0.1 degrees, and always beneath 0.5 degrees. Finally, the norm of the error in estimated angular velocity is shown in Figure 10. The estimate of angular velocity is off by roughly 0.001 degrees/second, usually less. It again shows a convergence time of less than an hour. This would be a promising performance for a CubeSat without a star tracker and using simple sensors, with an estimation accuracy of a fraction of a degree.

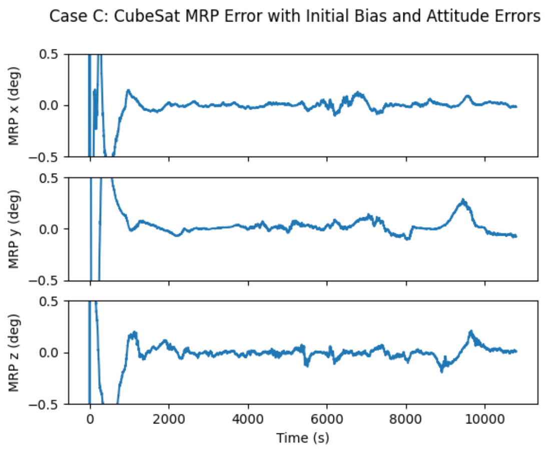

Figure 9: Case C (Section 3.3) showing the estimator's orientation error by-axis as a Modified Rodrigues parameter, running on a CubeSat with simple sensors. On each axis, it converges to within 0.5 degrees.

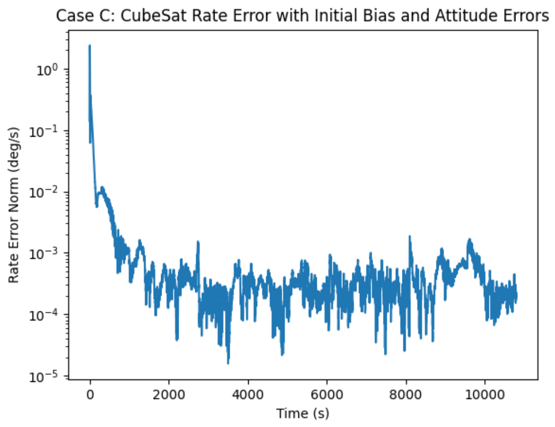

Figure 10: Case C (Section 3.3) showing the norm of the estimator's angular velocity error, running on a CubeSat with simple sensors. It remains within 0.001 degrees/sec.

3.4 Case D: CubeSat with Only Small Initial Bias Errors

The test from Case C, Section 3.3, was repeated with a "close" start, to demonstrate performance over time, after the filter has already converged. In this simulation, the initial estimated angular velocity and attitude are exactly correct, $P_{\text{att}} = (0.5°)$, and $P_\omega = (0.5°/\text{min})^2$.

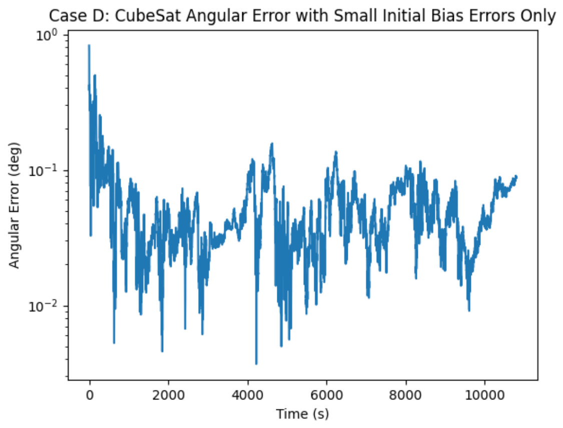

Figure 11: Case D (Section 3.4) shows the estimator orientation estimate staying to roughly 0.1 degrees on a CubeSat with simple sensors.

This "close" case mirrors the original performance from Section 3.3, indicating that such performance would continue over time, after the system has already converged. Figure 11 shows an angular estimation accuracy of approximately 0.1 degrees, similar to Figure 8. The breakdown of this angular error per axis (as Modified Rodrigues parameters) is provided in Figure 12. It can be seen that the angular error on each axis remains under 0.1 degrees. The norm of the angular velocity error, shown in Figure 13, remains under 0.001 degrees/second. These results support those in Section 3.3 as a promising estimation accuracy for a CubeSat without a star tracker and using simple sensors.

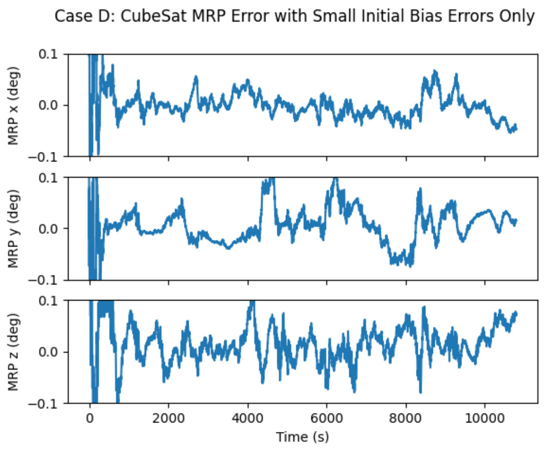

Figure 12: Case D (Section 3.4) showing the estimator's orientation error by axis as a Modified Rodrigues parameter, running on a CubeSat with simple sensors. On each axis, it stays mostly within 0.1 degrees.

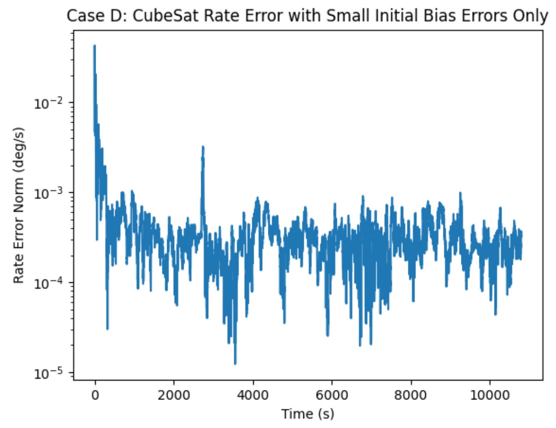

Figure 13: Case D (Section 3.4) showing the norm of the estimator's angular velocity error, running on a CubeSat with simple sensors. It remains within 0.001 degrees/sec.

3.5 Case E: Effect of Ignoring Actuator Bias or Disturbances

The dynamics-aware estimator in Case C is tracking quaternion orientation and angular velocity, and utilizing the dynamics, but it is also modeling actuator bias and disturbances. To test whether these are important, the same tests as in Case C are run (with the same setup), but excluding actuator bias or disturbances from the estimator (but not the "real" satellite).

Figure 14 compares the angular estimation between an exact replica of Case C against an identical satellite but an estimator that does not consider actuator bias. When the actuator bias is considered, the estimations stay at approximately 0.1 degrees of accuracy. However, when ignored, the system diverges wildly and remains approximately 180 degrees off from the correct orientation.

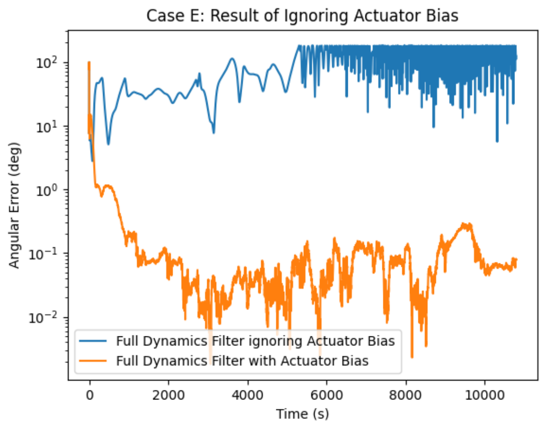

Figure 14: Case E (Section 3.6) comparing the full dynamics filter's performance when actuator bias (starting value of $\beta_{\mathbf{MTQ}} = 0.05 * \begin{bmatrix} \frac{1}{3\sqrt{2}} & \frac{1}{3\sqrt{2}} & \frac{4}{3\sqrt{2}} \end{bmatrix}$, drift rate of $\eta_{\mathbf{b,mtq}} = 0.000001$ $\mathbf{Am^2/s^{0.5}}$) is ignored by the estimator or considered. Including the actuator bias has an accuracy of around 0.1 degrees, while excluding it results in being 180 degrees off.

Figure 15 compares the angular estimation between an exact replica of Case C against an identical satellite but an estimator that does not consider the system disturbances. When the disturbances are considered, the estimations stay at approximately 0.1 degrees of accuracy. However, when ignored, the system only has an accuracy of around 1 degree.

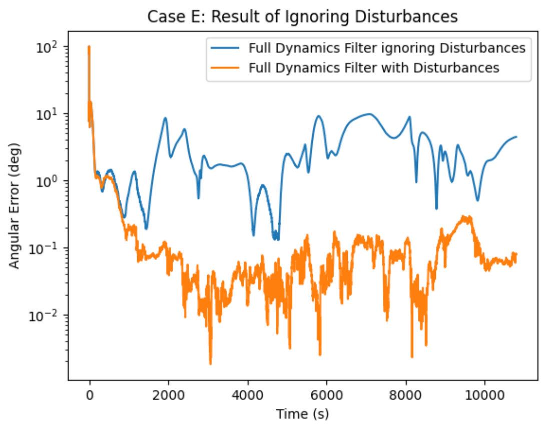

Figure 15: Case E (Section 3.6) comparing the full dynamics filter's performance when disturbances are ignored by the estimator or considered. Including the disturbances generates an accuracy of around 0.1 degrees, while excluding them results in an estimation error of around 1 degree.

3.6 Case F: CubeSat with Propulsion

To understand how the system deals with disturbances and to test the initial use case, a repeat of Case C (Section 3.3) was run, with the addition of a disturbing propulsion torque with initial value $\begin{bmatrix} \frac{-3}{\sqrt{74}} & \frac{-8}{\sqrt{74}} & \frac{1}{\sqrt{74}} \end{bmatrix}$ μNm, capped maximum value of 10 μNm, and a drift rate of $\eta_{\text{dipole}} = 0.01\ \mu\text{Nm/s}^{0.5}$.

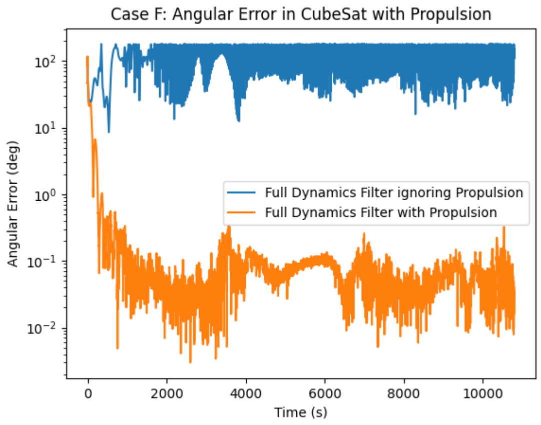

Figure 16: Case F (Section 3.6) comparing the full dynamics filter's performance when a propulsion disturbance is ignored by the estimator or considered. Including the propulsion generates an accuracy of around 0.1 degrees, while excluding them results in an estimation error of around 180 degrees.

The importance of including this torque in the estimator can be seen in Figure 16. Similar to the results in Figure 14, estimating while ignoring the disturbance torque results in a failure to converge and an angular error of around 180 degrees. However, when it is accounted for and being estimated by the system, the estimate converges to about 0.1 degree. It does this by estimating the value of the propulsion torque and tracking it over time, including it in the dynamics that it uses to estimate the evolution of the system state. The system's estimate of the propulsion torque compared to its "real" value can be seen in Figure 17. The estimate of the disturbing torque (dotted lines) converges quickly to the actual value (solid lines) and tracks it over time. It is not exact, but rather close, allowing for the effect of this torque to be included in the estimator's understanding of system dynamics.

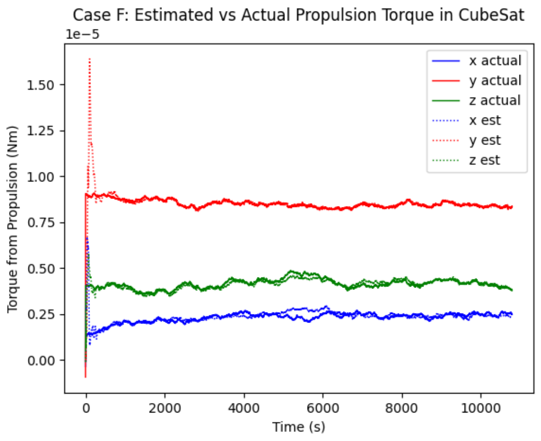

Figure 17: Case F (Section 3.6) comparing the full dynamics filter's estimated of a propulsion disturbance against the "real" value. The system estimate quickly converges and generally tracks the torque to roughly $5{\times}10^{-7}$ Nm, an order of magnitude less than the disturbance magnitude.

4 Summary and Future Work

These results show the potential value of fully including system dynamics in state estimation and, through this framework, allowing for complicated and detailed dynamics-aware estimation for a wide range of satellites with many different properties and disturbances. Specifically, dynamics-aware filtering outperforms USQUE by up to orders of magnitude in terms of angular and angular velocity errors when applied to a system with high-quality sensors (e.g., TRMM in Cases A and B, Sections 3.1 and 3.2). It also performs rather well, with less than a degree of angular error, when applied to a simulated cubesat with poor sensor quality, and this is true when there is bias error or bias and attitude error. The most significant result, however, may be that both of these outcomes were achieved by simply providing basic information about the sensors, actuators, and satellite mass properties (along with the time step), to the filter. Despite the differences in sensor accuracy, satellite properties, disturbance strength, and orbit, good performance was achieved in both cases with identical code. The ability to use this filter by simply describing the satellite's properties has great potential among small satellites. These promising results indicate the potential use of dynamics-based filters, especially for smallsats, where mass properties can be more accurately determined. However, there is much work remaining on this problem.

There are many sensors and a few actuators that are not yet implemented. This includes star cameras and trackers, control moment gyroscopes, Earth horizon sensors, and more. Extending to cover these would increase the number of use cases of this system, allowing it to model more varieties of spacecraft and be applicable to more missions. Given the class-based structure of this framework, such modifications should feasible.

The use of dynamics-aware modeling and larger states makes this approach more computationally complex than other approaches. The trade-off between accuracy and calculation resources should be further investigated. There are many sub-topics in the computational resources space, including questions of time step, and how to make this estimation approach (and the specific framework in this work) more efficient or precise for a specific case. As the dynamics-aware filter framework used in this work was implemented in Python, it can likely be made much faster using C++ or other compiled languages.

The extended Kalman Filter can have difficulties when there are "corners" in the system relating to the state. One example would be when the faces affected by drag and SRP change through rotation. One approach to deal with these sharp changes would be to approximate the torques using spherical harmonics, based on the orientation of the ram (for drag) and sunward (for SRP) vectors in the body frame.

There is the question of what happens if the disturbance models (or the inertia matrix and other properties) are incorrect. If the estimator has enough "slack," small modeling errors or inaccuracies should be handled well. However, wild mis-estimates, or those that interact with other aspects of the dynamics, may have a greater effect. Further research to understand when these cases occur and what their impact is would be well-warranted. Similar questions exist for mis-estimating variances and noise on sensors, actuators, disturbances, and other parameters. This includes more complex or specific questions such as how solar panel degradation can be modeled as bias and the effect of a spacecraft's own dipole (inherent, or from magnetorquers, and potentially electric activity) on magnetometer bias. All of this also ties into classic estimation questions regarding choosing parameters, covariances, and noise, and how to tune these values for a given application, as well as more modern questions regarding choosing correct error and evolution models, and cases of dependency.

There are many questions relating to varying parameters in this approach, as well. For example, how well can it track multiple time-varying disturbance torques? There is the possibility of including a generalized disturbance torque (estimated and time-varying, as described in Section 2.2.5) instead of each torque individually; this could reduce computational complexity and rely less on specific knowledge of the system.

There is room to extend this work in new directions, as well. An expectation-maximization approach is worth considering, switching estimating the state based on dynamics, and estimating system properties–like mass, inertia matrix properties, and off-axis actuator components, as well as properties currently covered in this system, like residual dipole or biases. A particle filter-based approach would also be interesting to see, as it may better capture the nonlinearities in this system.

Lastly, this has only been tested in one simulation software. Further tests in other simulations (potentially including hardware-in-loop) could show this system's benefits and weaknesses. Testing it on an actual satellite, on-orbit, would be the most compelling validation of its performance.

Funding Sources

This work was completed while supported by Northrop Grumman for work on the project "MIT BeaverCube II" and the US Air Force Artificial Intelligence Accelerator for "Automation in Space Domain Awareness."

Acknowledgments

Alex Meredith provided invaluable assistance in building the simulation and models in this project.

References

[1] Peter W Fortescue, Graham Swinerd, and John P. W Stark. *Spacecraft systems engineering.* Wiley, Chichester, 2011. OCLC: 1166768290.

[2] Chantal Cappelletti, Simone Battistini, and Benjamin Malphrus. *CubeSat Handbook: From Mission Design to Operations.* September 2020.

[3] Halil Ersin Soken, Cengiz Hacizade, and Shinichiro Sakai. Simultaneous Adaptation of the Process and Measurement Noise Covariances for the UKF Applied to Nanosatellite Attitude Estimation. *IFAC Proceedings Volumes*, 47(3):5921–5926, January 2014.

[4] Halil Ersin Soken and Chingiz Hajiyev. Pico satellite attitude estimation via Robust Unscented Kalman Filter in the presence of measurement faults. *ISA Transactions*, 49(3):249–256, July 2010.

[5] Halil E. Soken and Chingiz Hajiyev. Fault Tolerant Attitude Estimation for Pico Satellites Using Robust Adaptive UKF. *IFAC Proceedings Volumes*, 45(20):726–731, January 2012.

[6] John Crassidis, Stephen Andrews, Landis Markley, and Kong Ha. Contingency designs for attitude determination of TRMM. January 1995.

[7] Avishai Weiss, Frederick Leve, Ilya V. Kolmanovsky, and Moriba Jah. Reaction Wheel Parameter Identification and Control through Receding Horizon-Based Null Motion Excitation. In Daniel Choukroun, Yaakov Oshman, Julie Thienel, and Moshe Idan, editors, *Advances in Estimation, Navigation, and Spacecraft Control*, pages 477–494, Berlin, Heidelberg, 2015. Springer.

[8] John L Crassidis. Sigma-Point Kalman Filtering for Integrated GPS and Inertial Navigation.

[9] J. W. Chia, M. S. C. Tissera, K. S. Low, S. T. Goh, and Y. T. Xing. A low complexity Kalman filter for improving MEMS based gyroscope performance. In *2016 IEEE Aerospace Conference*, pages 1–7, March 2016.

[10] John L. Crassidis and F. Landis Markley. Unscented Filtering for Spacecraft Attitude Estimation. *Journal of Guidance, Control, and Dynamics*, 26(4):536–542, July 2003.

[11] Mark Wood and Wen-Hua Chen. Regulation of Magnetically Actuated Satellites using Model Predictive Control with Disturbance Modelling. In *2008 IEEE International Conference on Networking, Sensing and Control*, pages 692–697, April 2008.

[12] Patrick McKeen. *Improved Attitude Control of Small Satellites via Planning and Tailored Control.* PhD thesis, Massachusetts Institute of Technology, 2024. Unpublished PhD thesis.

[13] Michael Andrle and John Crassidis. Geometric Integration of Quaternions. In *AIAA/AAS Astrodynamics Specialist Conference*, Minneapolis, Minnesota, August 2012. American Institute of Aeronautics and Astronautics.

[14] Riley M. Fitzgerald and Kerri L. Cahoy. Localized In-Situ Density Measurement in Low Earth Orbit via Drag Torque Estimation. *Journal of Spacecraft and Rockets*, 56(5):1564–1579, September 2019. Publisher: American Institute of Aeronautics and Astronautics.

[15] M. Lovera and A. Astolfi. Global magnetic attitude control of spacecraft. In *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*, volume 1, pages 267–272 Vol.1, December 2004. ISSN: 0191-2216.

[16] Marco Lovera. Magnetic satellite detumbling: The b-dot algorithm revisited. In *2015 American Control Conference (ACC)*, pages 1867–1872, July 2015. ISSN: 2378-5861.

[17] Nikolas Trawny and Stergios I Roumeliotis. *Indirect Kalman Filter for 3D Attitude Estimation.* PhD thesis, University of Minnesota, March 2005.

[18] Jason Tuthill. *DESIGN AND SIMULATION OF A NANO-SATELLITE ATTITUDE DETERMINATION SYSTEM.* PhD thesis, Naval Postgraduate School, December 2009.