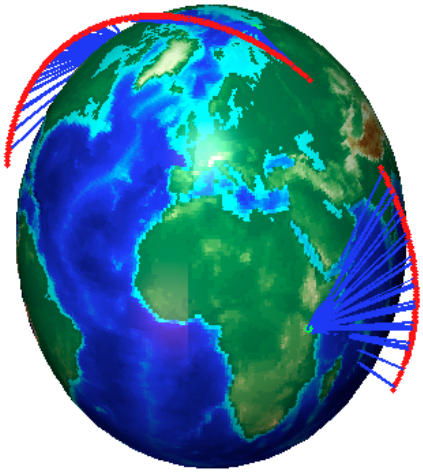
ASEN 5010 Course Project

Zach Dischner
University of Colorado at Boulder
Department of Aerospace Engineering

The essence of attitude determination is the estimation of a body's pointing relative to a known frame, given one or more observations taken in different frames. This paper will discuss the development and theoretical testing of attitude estimation algorithms for a satellite with two such sensors. Numerical simulations will simulate measurements taken in various frames, and from them determine the spacecraft's attitude relative to the inertial frame using the OLAE method. At the same time, spacecraft equations of motion will be developed to find the evolution of the satellite's attitude in time. Various parameters such as measurement noise, and orbit characteristics will be examined in order to quantify their impact on performance on the OLAE algorithm.



Nomenclature

i	Orbit Inclination Angle
Ω	Longitude of the Ascending Node
θ	Orbit Position Angle
r	Orbit Radius
r_E	Mean Radius of the Earth
μ	Gravitational Constant
n	Mean Orbit Rate
$[XY]$	DCM Mapping Y to X Frames
${}^{X}x$	x expressed in X frame components
SNR	Signal-to-Noise Ratio
Left Superscript	
N	Inertial Frame
B	Body Frame
T	Topographical {n,e,d} Frame

I. Introduction

The spacecraft in question for this examination is equipped only with sun and magnetic field direction sensors. Numerical simulations of measurements taken from both of these devices in the B frame, as well as the N frame. Relating measurements between frames, and ultimately providing a measure of the spacecraft's attitude, will be done with the Optimal Linear Attitude Estimation (OLAE) method. Through this numerical simulation, the affect of sensor noise, errors in the orbit parameters, and other sources of error will be examined and quantified.

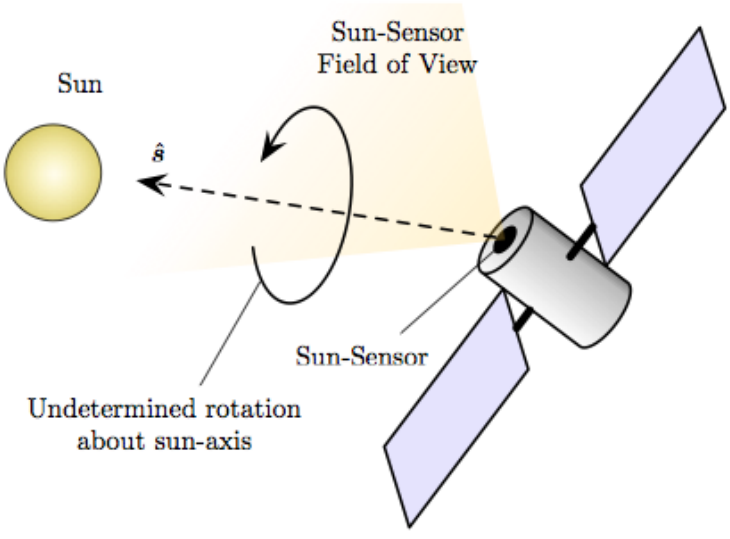


Figure 1. Satellite Sun Sensor Visual

The spacecraft itself is orbiting the Earth in a pure circular orbit, and is tumbling free of any active attitude control. The inertial frame position of the satellite is given by:

$$ {}^{N}r = \begin{pmatrix} \cos\Omega\cos\theta - \sin\Omega\sin\theta\cos i \\ \sin\Omega\cos\theta + \cos\Omega\sin\theta\cos i \\ \sin\theta\sin i \end{pmatrix} \tag{1} $$

where

$$ \Omega = 2^o $$
$$ i = 75^o $$

For this simulation, the longitude of the ascending node is Ω, the orbit inclination is i, and θ is the orbit position angle relative to the equatorial crossing point. As the orbit is circular, its governing equation is:

$$ \theta(t) = \theta_0 + nt \tag{2} $$

where n is the mean orbit rate, r is the constant orbit radius, and mu is the Earth's gravitational constant.

$$ n = \sqrt{\tfrac{\mu}{r^3}} $$
$$ r = 6878\text{km} $$
$$ \mu = 398600\text{km}^3/\text{s}^2 $$

The spacecraft is axially symmetric, and has the inertia tensor given in body frame components:

$$ {}^{B}I = \begin{bmatrix} 25 & 2.5 & 0.5 \\ 2.5 & 20 & 0 \\ 0.5 & 0 & 15 \end{bmatrix} \text{kg m}^2 $$

The spacecraft also is experiencing no external torques, and has initial 3-2-1 Euler angles and initial angular rates defined as:

$$ \begin{bmatrix} \psi_0 \\ \theta_0 \\ \phi_0 \end{bmatrix} = \begin{bmatrix} 5^o \\ 10^o \\ -5^o \end{bmatrix} \quad \Big| \quad {}^{B}\omega_0 = \begin{bmatrix} 0.4 \\ 0.3 \\ 0.2 \end{bmatrix} \text{deg/s} $$

II. Results

In this section, the development, testing, and simulation of various subprocess in the attitude determination process will be discussed. Unless otherwise mentioned, all numerical calculations, simulations, and visualizations were all performed in Matlab, and all code can be found in IV.

II.A. Part A

The first step in setting up a numerical simulation of the satellite was to obtain its position in orbit for an arbitrary time. Using EQ 1, solutions for the satellite positions were found over a ten minute period. Figure 2 shows the result of this simulation.

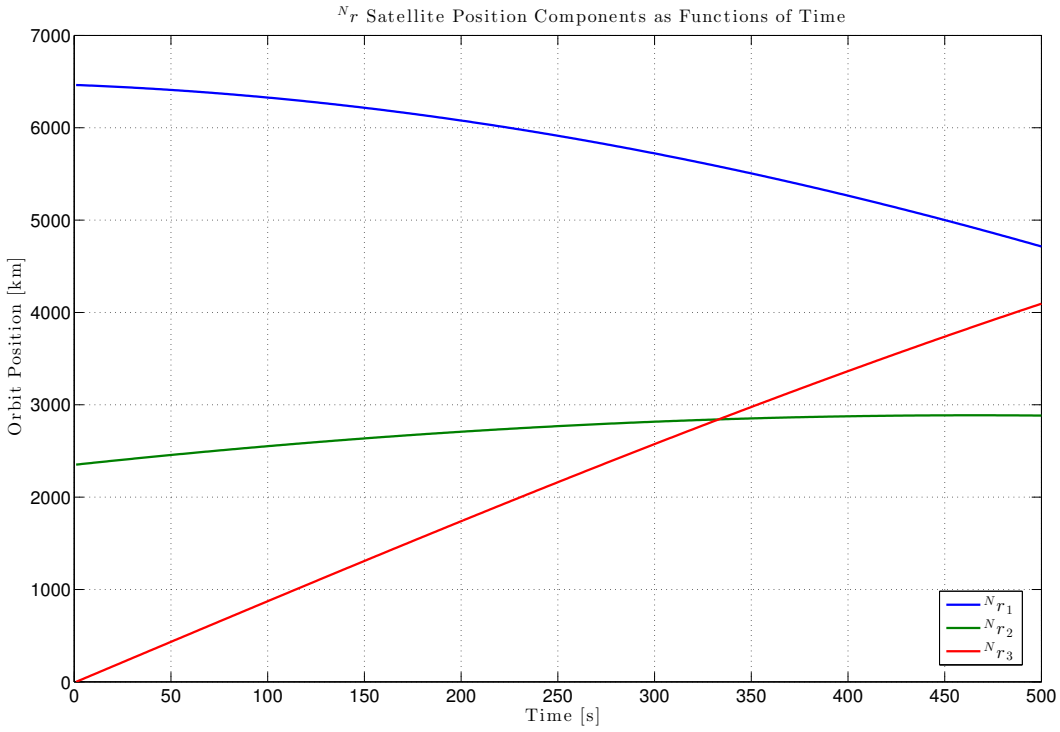


Figure 2. Orbit Position in Time

II.B. Part B

Next, a routine was written to take a body orientation as a set of MRPs $\sigma_{\mathbf{B/N}}$, and with the known inertial sun direction vector ${}^N\hat{s} = (0, -1, 0)^T$, compute the simulated sun direction measurement in body coordinates.

This problem essentially boils down to rotating a measurement in one frame into another frame. In general, this is done by multiplying one observation by the appropriate rotation matrix.

$$^X a = [XY]^Y x$$

Where in this case, the two frames are the body B and inertial N frames, and the vector quantity being rotated is the measured sun direction vector $\hat{s}$.

$$^B\hat{s} = [BN]^N\hat{s}$$

In the above equation, $[BN]$ is the 3x3 rotation matrix, or DCM, which describes the three dimensional rotation between the B and N frames. In this case, this rotation is given as a set of MRPs. From lecture notes, the 3x3 DCM can be extracted from a set of MRPs through the relationship in EQ 3.

$$[BN] = [I_{3x3}] + \frac{8[\tilde{\sigma}]^2 - 4(1-\sigma^2)[\tilde{\sigma}]}{(1+\sigma^2)^2} \tag{3}$$

This conversion was provided with the course's computational toolbox, and was implemented in Matlab here. The code for this routine is found in Appendix IV.B.

II.C. Part C

The frame for the satellite to be used in this simulation is a North-East-Down (NED) topographic frame τ. Another main frame of interest is the Earth-fixed frame ϵ.

$$\tau : \{\hat{n}, \hat{e}, \hat{d}\}$$

$$\epsilon : \{\hat{e}_1, \hat{e}_2, \hat{e}_3\}$$

This frame is illustrated in

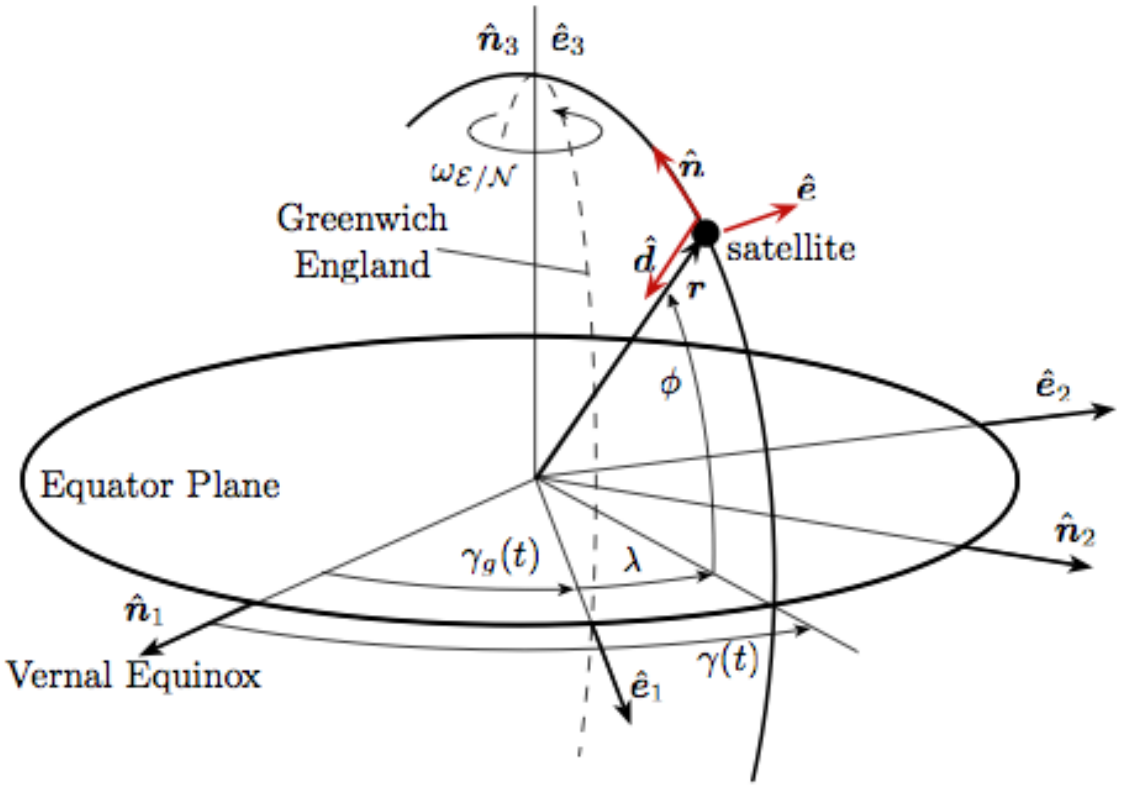


Figure 3. North East Down Frame

In order to describe the arbitrary rotation between the two frames, a DCM was constructed. Starting at the ϵ frame, I first rotated λ about the third axis, $\hat{e}_3 = \hat{n}_3$.

$$[M_3(\lambda)] = \begin{bmatrix} \cos\lambda & \sin\lambda & 0 \\ -\sin\lambda & \cos\lambda & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

Now, $\hat{e}'_2 = \hat{e}$. The next rotation is about $\hat{e}$ needs to be a negative rotation of magnitude θ. However, this rotation produces a frame in which $\hat{e}''_3 = \hat{n}$. By adding another $\pi/2$ rotation to the previous, all three axes of the ϵ'' frame align with the new NED τ frame. Essentially, this step combines two rotations about $\hat{e}$, so that instead of a 3-2-2 rotation, we just have a more intelligent 3-2 rotation set.

$$[M_2(-\phi - \pi/2)] = \begin{bmatrix} \cos(-\phi - \pi/2) & 0 & -\sin(-\phi - \pi/2) \\ 0 & 1 & 0 \\ \sin(-\phi - \pi/2) & 0 & \cos(-\phi - \pi/2) \end{bmatrix} \tag{5}$$

With no orbit inclination, these two orbit inclinations are all that are necessary to rotate the τ frame into the ϵ frame. The resulting DCM is computed by multiplying EQ 5 by EQ 4.

$$[TE] = [M_2][M_3] \tag{6}$$

II.D. Part D

Like in the previous section, another frame of interest is the earth-centered-inertial (ECI) frame. For the purpose of this exercise, the ECI frame differs from the ECEF frame by the angle γ, the Greenwich local sidereal time. It is assumed that this angle is a constant function of time, and only has components in the $\hat{e}_3$ direction.

$$\gamma(t) = \gamma_{t_0} + \omega_{E/N} \tag{7}$$

$$\omega_{E/N} = 361\text{deg/day}$$

Now, to rotate between the ECI and ECEF frames, a rotation matrix DCM can be built in the same manner as before in section II.C. But now, the rotation angle γ is a function of time.

$$[EN] = [M_3(\gamma(t))] = \begin{bmatrix} \cos\gamma(t) & \sin\gamma(t) & 0 \\ -\sin\gamma(t) & \cos\gamma(t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

The Matlab code for this computation is included in Appendix IV.D.

II.E. Part E

Combining many of the above sections, now a routine was written in which the magnetic field unit direction vector in inertial frame components ${}^N M$ was solved for, given inputs ${}^N r$ and t.

The magnetic field vector was given in NED frame components in EQ 9.

$$\begin{bmatrix} M_{north} \\ M_{east} \\ M_{down} \end{bmatrix} = \left(\frac{r_{eq}}{r}\right)^3 \begin{bmatrix} -\cos\phi & \sin\phi\cos\lambda & \sin\phi\sin\lambda \\ 0 & \sin\lambda & -\cos\lambda \\ -2\sin\phi & -2\cos\phi\cos\lambda & -2\cos\phi\sin\lambda \end{bmatrix} \begin{bmatrix} 29900 \\ 1900 \\ -5530 \end{bmatrix} \text{nT} \tag{9}$$

r_{eq} is the earth's mean equatorial radius, r is the mean orbit radius, λ is the satellite longitude relative to ECEF, and ϕ is the satellite latitude.

Since EQ 9 is given, and from the orbit position vector, both λ and ϕ can be calculated, the task now is to rotate the computed magnetic field components from the NED (T) frame to the N frame. In the previous sections, I discussed how the [TE] and [EN] frames were constructed, as functions of λ, ϕ, and t. In this routine, those routines were used to get a final inertial representation of the earth's magnetic field at a point in time along the orbit, as shown in EQ 10.

$$\begin{bmatrix} M_{\hat{n}_1} \\ M_{\hat{n}_2} \\ M_{\hat{n}_3} \end{bmatrix} = [EN]^T[TE]^T \begin{bmatrix} M_{north} \\ M_{east} \\ M_{down} \end{bmatrix} \tag{10}$$

A simulation of this calculation was made for ten minutes of an orbit, and the resulting plot is shown in Figure 4. The code for this computation is included in Appendix IV.E.

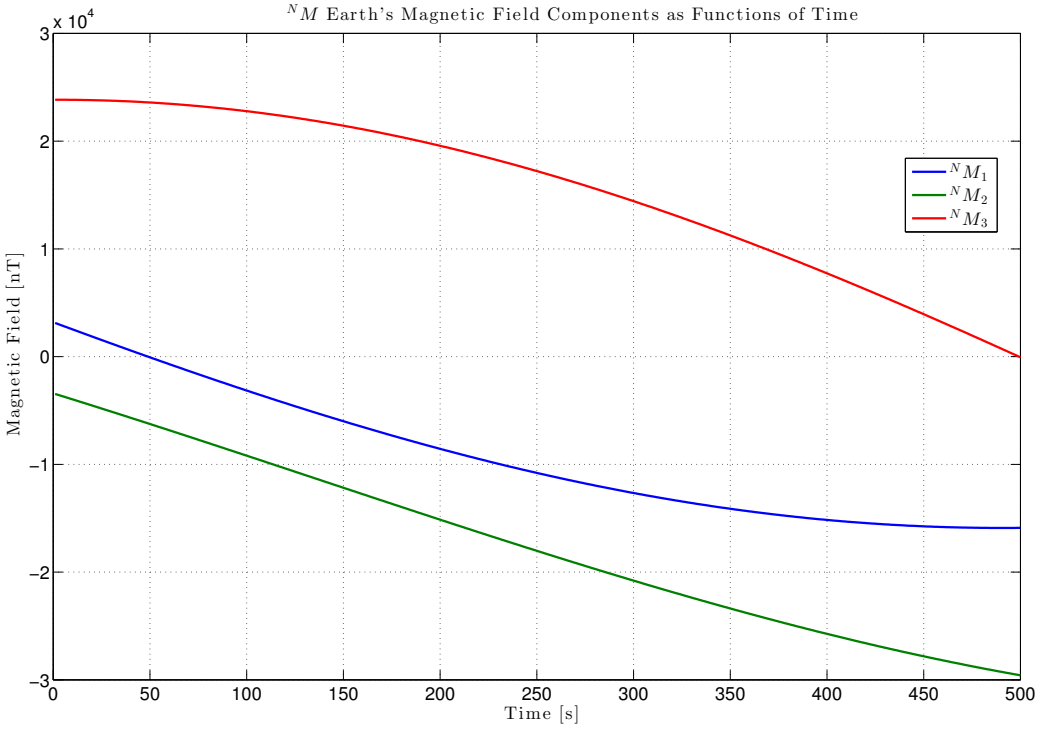


Figure 4. Manetic Field Elements

II.F. Part F

Next, the equations of motion for a rigid body were used to simulate the spacecraft's true attitude by means of numerical integration. To integrate the satellite's attitude in time, both the angular velocity and angualr position quantities need to be described by a set of equations of motion. Those for the angular velocity are the standard equations of motion of a rigidly rotating body, expressed in equation 11. I is the satellite's inertia tensor, and L is the sum of external torques acting on the body. In this case, the external torques are zero.

$$I\dot{\omega} = -[\tilde{w}]I\omega + L \tag{11}$$

To integrate the spacecraft's attitude in time in terms of the initially given 3-2-1 Euler angles, equation 12 was used. $[B(\theta)]$ is the matrix described in the course textbook[4] that is used to relate the angular velocity of a body ω to the rate of change of the body's attitude $\dot{\theta}$.

$$\dot{\theta} = [B(\theta)] * \omega \tag{12}$$

Together, these two equations allow for the integration in time of the spacecraft's attitude (in 3-2-1 Euler angles) and angular velocity vectors. The attitude was converted into MRPs during the integration to check for the need for a shadow set switchover. Integrations were performed in Matlab, using initial conditions and spacecraft properties discussed in section I. A plot for the orbit attitudes and angular velocities over a 10 minute duration can be seen in Figure 5. There is one switchover between MRP shadow sets in this time period. This simulation is not long enough to garner any more insight into the attitude or angular velocity evolution of this system.

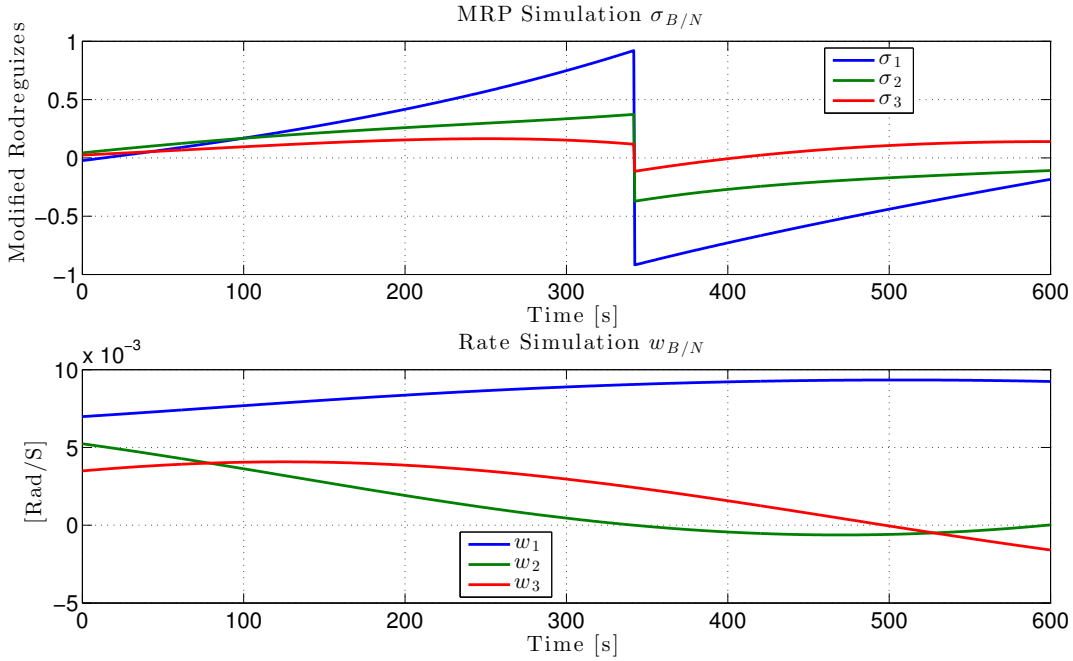


Figure 5. Numerical Integration of Attitude and Angular Velocity

II.G. Part G

Next, the sun sensor and magnetic field sensor measurements outlined in section II.B and section II.E were integrated into the simulation of the equations of motion. These measurements don't arise from system dynamics, but can be calculated as a result of the spacecraft's attitude. Without any sensor noise or modeled sources of error, these are a purely numerical expectation of these two sensors' readings. Figure 6 shows these two expected measurements, as seen by the body frame.

The simulation for all attitude investigations is found in the code appendix, section IV.G. The single script performs all investigations necessary, and will offer different results based on tunable parameters.

II.H. Part H

The actual attitude estimation process is one that compares the estimated attitude, given by numerical simulations and models, with the observed attitude, typically given by in-situ sensor measurements. Section II.F outlines the results of modeling the spacecraft's attitude, while section II.G outlines how we have simulated perfect measurements from two different sensors onboard the spacecraft. The gap needed to fill in comparing modeled and observed attitudes is in converting various body-frame measurements into actual attitude coordinates. This represents one of the fundamental tasks in attitude estimation. Parsing multiple observations into a single attitude estimation has been performed with the Triad method, and later by solving Wahba's problem with the Q-method and QUEST algorithms. More recently, the Optimized Linear Attitude Estimator (OLAE) has garnered some interest, and will be discussed here.

Recall that each observation must satisfy

$$^{B}\hat{v}_i = [BN]^{N}\hat{v}_i \tag{13}$$

$[BN]$ is the desired attitude estimate. Decomposing $[BN]$ with the Cayley transformation,

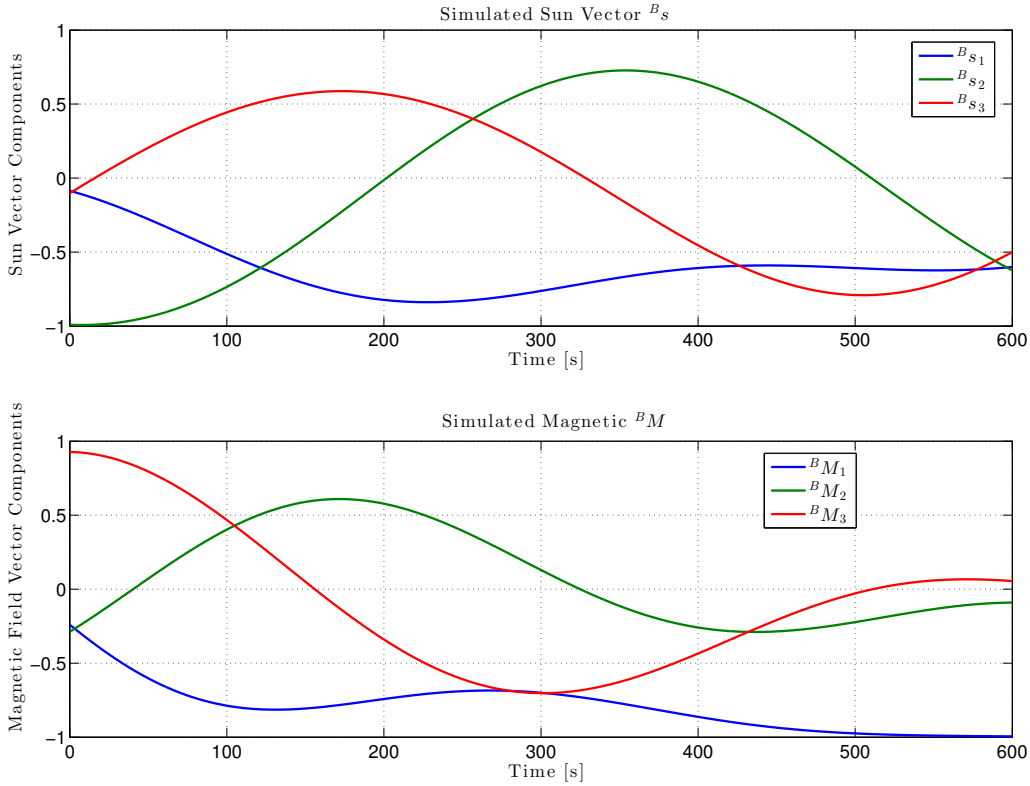


Figure 6. Expected Magnetic Field and Sun Sensor Measurements

equation 14 shows the elegantly simple result.

$$[BN] = (I_{3\times3} + [\tilde{\bar{q}}])^{-1}(I_{3\times3} - [\tilde{\bar{q}}]) \tag{14}$$

Applying equation 14 to equation 13 and multiplying by $(I_{3\times3} + [\tilde{\bar{q}}])$ yields

$${}^{B}\hat{v}_i - {}^{N}\hat{v}_i = -[\tilde{\bar{q}}]({}^{B}\hat{v}_i - {}^{N}\hat{v}_i) \tag{15}$$

Now, the desired attitude estimate is described by $\bar{q}$. By defining the summation and difference matrices as

$$\mathbf{d}_i = ({}^{B}\hat{v}_i - {}^{N}\hat{v}_i) \tag{16}$$

$$\mathbf{s}_i = ({}^{B}\hat{v}_i + {}^{N}\hat{v}_i) \tag{17}$$

Now, rewriting equation 15 with equations 16 and 17, the surprisingly simple linear result is found to be

$$\mathbf{d}_i = [\tilde{\mathbf{s}}_i]\bar{q} \tag{18}$$

As is to be expected with solving for attitudes, a single observation fed into equation 18 will not yield a full 3d attitude solution. In this convenient linear form, it is simple enough to augment equation 18 to accommodate N number of observations, which allows for $\bar{q}$ to be solved for analytically using least squares. This formulation also allows for weightings to be applied to each

observation set, which is crucial in trying to determine attitude from various sensors with different accuracies.

For N measurements, each with its own weighting value w_k, the matrix components of equation 18 are formed in the following manner. The $\mathbf{d}$ matrix is stacked to become a $3N \times 1$ matrix of differences. [$\mathbf{S}$] becomes a $3N \times 3$ summation tilde matrix, and the weighting values are arranged into a $3N \times 3N$ matrix.

$$\mathbf{d} = \begin{bmatrix} \mathbf{d}_1 \\ \mathbf{d}_2 \\ \vdots \\ \mathbf{d}_N \end{bmatrix} \quad [\mathbf{S}] = \begin{bmatrix} \tilde{\mathbf{s}}_1 \\ \tilde{\mathbf{s}}2 \\ \vdots \\ \tilde{\mathbf{s}}_N \end{bmatrix} \quad W = \begin{bmatrix} w_1 I_{3\times3} & 0I_{3\times3} & \cdots \\ 0I_{3\times3} & \cdots & 0I_{3\times3} \\ \cdots & 0I_{3\times3} & w_N I_{3\times3} \end{bmatrix} \tag{19}$$

Now, rearranging and solving for $\bar{q}$, the optimal CRP attitude can be solved using a least squares solution.

$$\bar{q} = ([S]^T[W][S])^{-1}[S]^T[W]\mathbf{d} \tag{20}$$

This solution is very powerful in that it is a purely linear solution, and includes variable weights through which a fine solution can be found. However, as it does use CRPs, the singularities at 180 degrees must be addressed in the same manner as in the QUEST algorithm. More details into the mathematics of this and other algorithms are discussed further in the project description document.[3]

The OLAE method was integrated into my simulation discussed in previous sections. I wrote a function that takes arbitrary N number of body frame and inertial frame observations, as well as optional weighting. With them, I dynamically construct the matrices in equation 19. The code for this operation is surprisingly simple and short, and can be found in the code appendix, section IV.F.

At this point in the simulation there are N frame measurements for the sun and magnetic field sensors, calculated theoretically. B frame measurements are also available. Both are calculated purely based off of the simulation data, so the 'true' simulated attitude (β) should perfectly match the estimated body attitude ($\bar{\beta}$) found from the OLAE algorithm, since all measurements are perfectly computed (to machine precision).

In Figure 7, the true and estimated attitudes are presented overlaid, in terms of their MRP components. It looks like there is just a single plotted value, but that is because the estimated and true values are nearly identical.

Perhaps more informative is a plot of the residuals of the true and OLAE attitude computations. Figure 8 shows just that. For almost the entire simulation, the residual is nonexistent. On the order of 10^{-15}. There is a blip visible right at the MRP shadow set switchover, but it is still such a small magnitude that the residual is assumed to still be zero.

Thus, with perfect observations from both frames, the OLAE method proves to be a perfect attitude estimator (to numerical precision). But real systems will have biases, timing errors, and sensor noise. So OLAE's performance must be examined with these things in mind.

II.I. Part I

Now, sensor corruption must be included in my simulation, to see its effect on the estimated attitude found via OLAE. Sensor corruption can come from a variety of sources. There can be miss-calibration biases, modeling errors, optical and electrical noise, resolution-induced variability, and many other places. Simply adding random numbers to the simulation is not adequate.

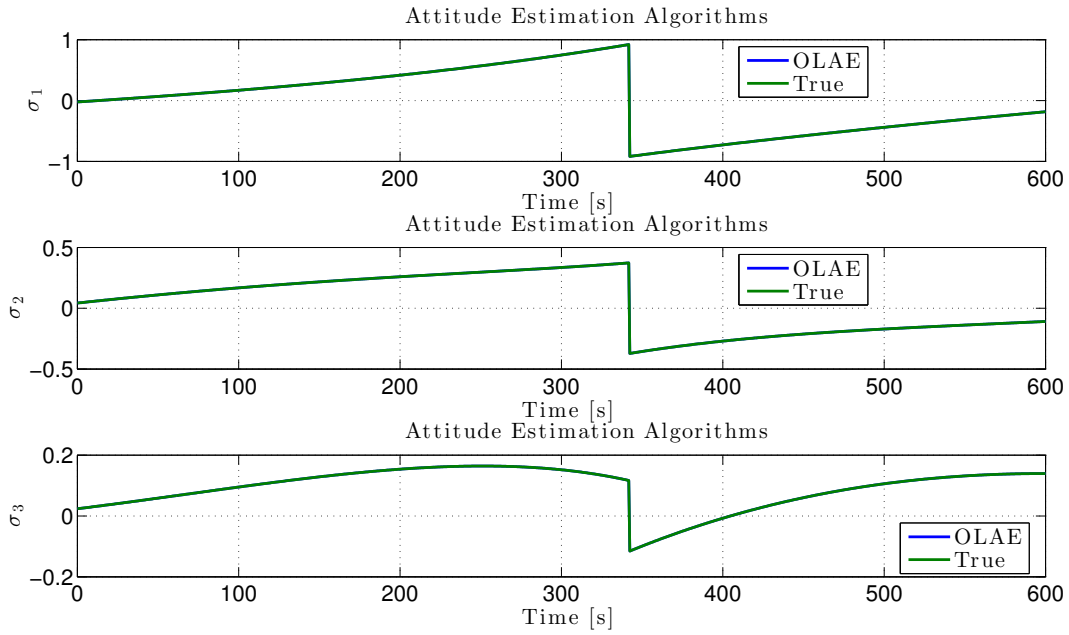


Figure 7. MRP Comparision Without Observation Noise

The method I chose to induce sensor corruption was based around the Signal-to-Noise-Ratio (SNR). No matter what is being measured, whether microvolt EM field vectors, kilometer-ranged vectors from GPS, or essentially infinite star-tracker attitude vectors, every signal has some ratio of its signal over the cumulative background noise from the sensor the signal originates from. So with a given signal, I chose to use Matlab's *awgn* function to augment a given signal with random white noise, constrained by a given signal-to-noise requirement for the signal. This is very flexible in that it can be used to augment any signal, no matter the magnitude, in an intelligent way. Unit vectors and nano-unit vectors get the same relative treatment.

With this as my sensor corruption approach, I made a few other assumptions. I assumed that the inertial measurements were perfect, and that all noise is applied to the body-frame measurements. I did this assuming more thorough modeling and more sensitive instruments are providing inertial measurements, but also chose to do this to better isolate variables in order to see a more direct cause-effect relationship in the noise and OLAE results. I also assumed that noise wasn't time or position varying, and stayed at the same order of magnitude throughout the entire orbit.

With these assumptions in mind, I modified the simulation to add noise as constrained by a given SNR to each body-frame B observation computed. The desired SNR for each instrument was chosen due to legacy stats on some sun sensors and magnetic field sensors. I found an acceptable SNR for sun sensors to be around 60dB.[2] For magnetic field sensors, I found a newer high-sensitivity magnetic field sensor to have a SNR of 200dB.[1] These values are reasonable for modern sensors, but are still on the cutting edge compared to many older legacy instruments currently in operation.

Now, I re-performed the simulation with white noise being generated for each body and observation of the sun and magnetic field vectors. With these high SNR values, the OLAE estimated attitude is still very close to the truth value. Figure 9 shows the new residual differences between the true and estimated attitudes. Compared with Figure 8, it is clear that adding noise has made the residuals worse by many orders of magnitudes. The differences are still on the order of mili-radians, which may be fine for a given mission, or it may be out of spec. Again, there is a spike

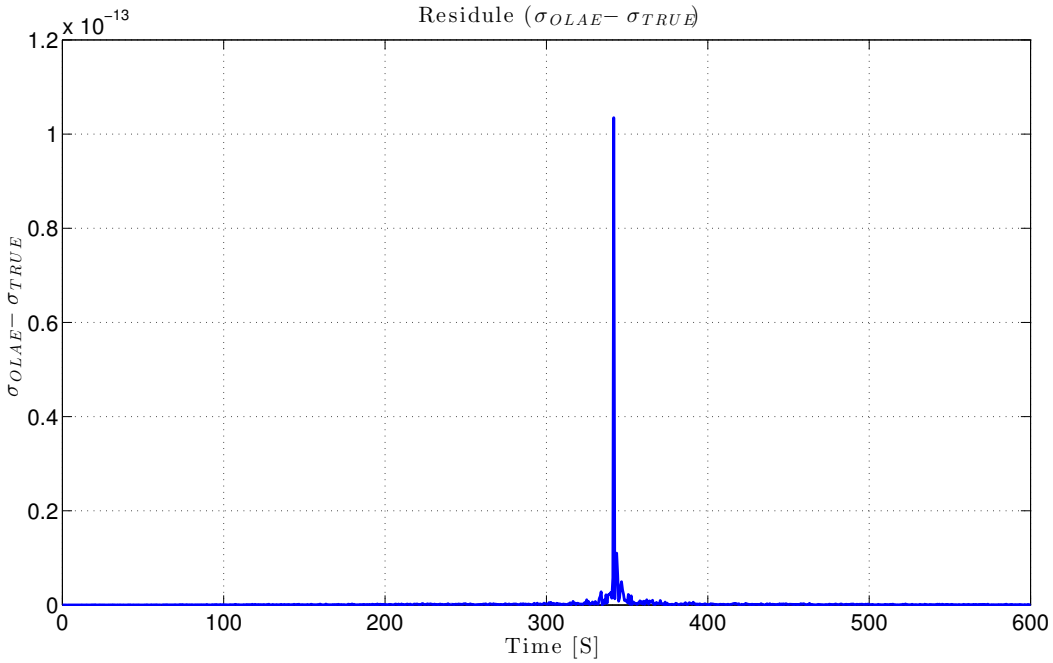


Figure 8. MRP Residules Without Observation Noise

right at the MRP switchover. This simulation also ignores weighting of the different measurements. Weighting's effect will be discussed shortly.

Simulation Summary Without Noise or Weights	
Magnetic Field Sensor SNR	200dB
Sun Sensor SNR	60dB
Magnetic Field Sensor Weight	1
Sun Sensor Weight	1
Mean of Residuals	0.000384 rad

The takeaway here is that for newer, clean sensors, the OLAE method still provides a good solution for the spacecraft's attitude, accurate to the milli-radian level.

From personal experience, I know that similar sensors often operate at much magnitude lower SNRs. Some of the lower cost, smaller instruments on CubeSats can sometimes operate at orders of magnitude lower SNR. To try and get a better idea of how the OLAE algorithm performs on a more realistic system, I decided simulate a sun sensor with a SNR of 30dB and a magnetic field sensor with a SNR of 10dB.

At this point, when dealing with imperfect measurements, forgoing the use of weighting in the OLAE algorithm is unacceptable. It is formulated as a weighted least squares, and will perform best when the weighting is applied. I chose to dynamically create the weighting values for each sensor measurement based on the relative SNR of each. This way, the measurement with a higher SNR will be weighted more heavily, since it implies that that sensor is the more accurate one.

Figure 10 shows the overlaid truth and OLAE computed MRP components throughout the simulation. Clearly, in contrast with Figure 7, the OLAE attitude is now variable about the truth solution. In addition, at the time of MRP switchover, the OLAE estimated attitude is very unsteady. At the point of switchover, the computed attitude with noise can cross back and forth across the switchover bound.

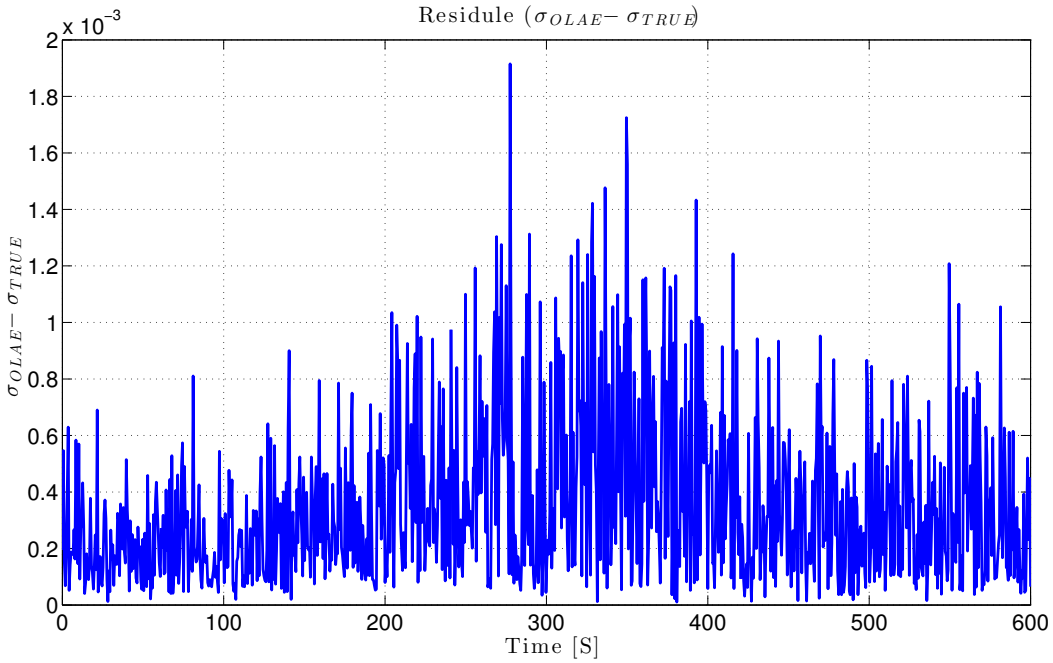


Figure 9. MRP Residules With Observation Noise

Figure 11 shows the average MRP residuals for this case. Now, the residuals are on the order of hundreds of mili-radians. At the switchover, this becomes a residual of nearly 2 radians. Clearly, for a noisy system, further measures need to be taken to get rid of these anomalies around the shadow set switch.

Table II.I summarizes this simulation's findings. With sensor noise and weightings, this setup yields residuals that are almost 2 orders of magnitude larger than with the cutting edge instruments. Again, this is for just two sensors operating on a satellite with no more advanced estimation algorithms. That being said, the results are still on track with the truth values.

Simulation Summary With Noise and Weights	
Magnetic Field Sensor SNR	10dB
Sun Sensor SNR	30dB
Magnetic Field Sensor Weight	0.25
Sun Sensor Weight	0.75
Mean of Residuals	0.166410 rad

Next, I wanted to investigate what happens with inappropriate weightings. To do this, I used the same lopsided SNR ratios as before, but switched their weighting values. This way, the magnetic field sensor (with worse SNR and worse performance) was weighted heavier in the OLAE's weighted least squares estimation of the spacecraft's attitude $\bar{\beta}$.

Figure 12 Shows these results. Compared to Figure 11, the residuals are different, but not notably worse. Looking at the mean of residuals is where the difference becomes visible.

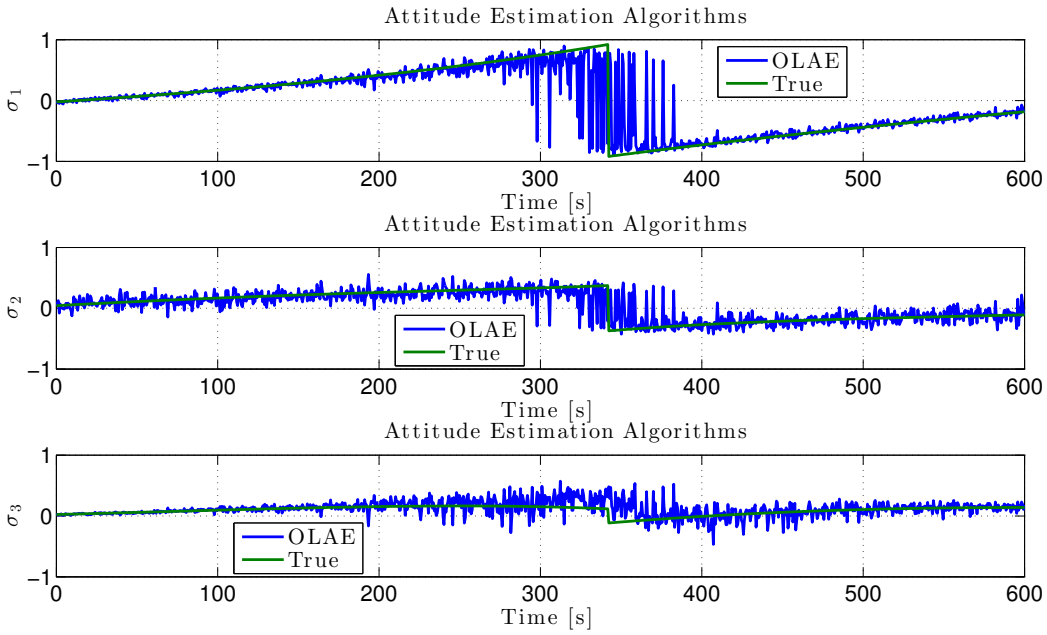


Figure 10. MRP Comparision With Observation Noise

Simulation Summary With Noise and False Weights	
Magnetic Field Sensor SNR	10dB
Sun Sensor SNR	30dB
Magnetic Field Sensor Weight	0.75
Sun Sensor Weight	0.25
Mean of Residuals	0.184266 rad

The takeaway here is that, at least in this simulation, false weights do have an effect. The mean of the residual error increased by about 10% by applying higher weighting to the worse sensor. This is interesting to see, in that the exact same data sets can yield different results depending on your interpretation and setup. So if an instrument the satellite depends on for an attitude observation has been miss-calibrated, or suffers from some degradation in flight, the computed attitude can become skewed.

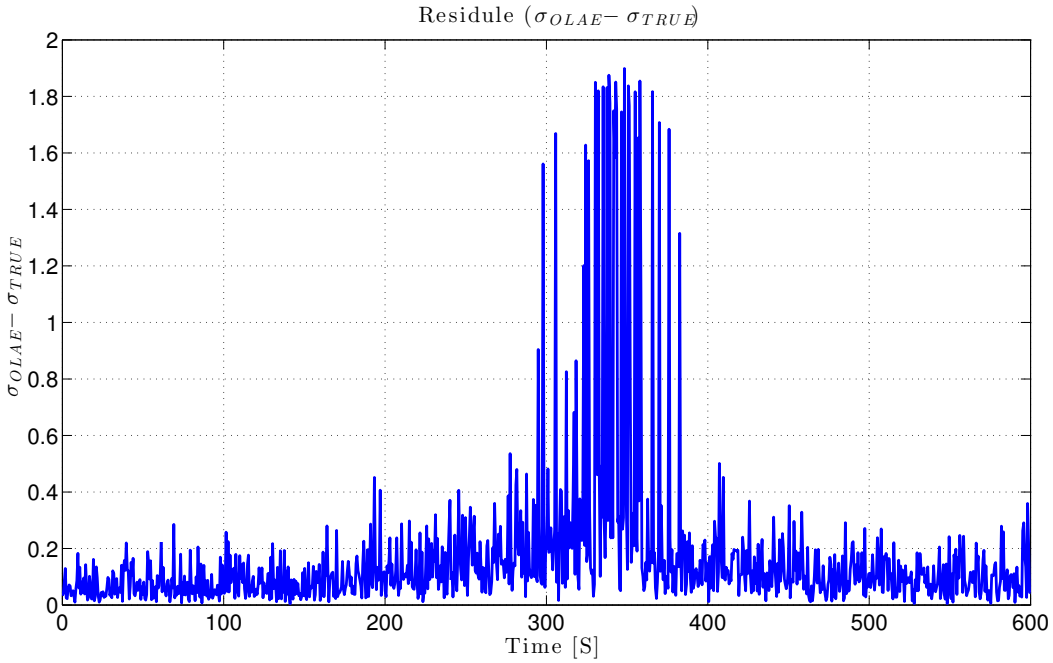


Figure 11. MRP Residules With Observation Noise and Weighting

III. Conclusion

This investigation was very insightful. Matlab was used to create a simulation of a tumbling satellite given some initial angular position and rate conditions. Equations of motion were used to then propagate its attitude in time. At the same time, methods were developed to determine inertial and body observations of both a sun sensor and a magnetic field sensor based on the time/position in orbit of the satellite. These were all used to test the behavior of a newer attitude determination method, OLAE. With this method, I was able to determine that with perfect observations in both the body and inertial frames, the $[BN]$ rotation set between the two frames could be perfectly determined.

However, when noise is added to the sensor measurements, the accuracy of the OLAE method starts to degrade. More so is the degradation when inaccurate weighting schemes are used to add calculation bias to one sensor or another.

The investigation into OLAE's performance was a limited one, however. It only investigated how consistently random noise applied to each sensor's measurement (while remaining in a given SRN) skews the estimated attitude. As such, this is a perfect place to implement an intelligent estimator, or Kalman Filtering algorithm to help and tone down the residual errors. In addition, no sensor corruptions beyond a white noise addition was examined. A more thorough work would look into sensor biases, pattern noise, and time varying noise measurements. However, the simple case looked at did provide good insight into the performance of the OLAE algorithm, and the project as a whole gave a good starting point as far as a numerical simulation of a satellite's attitude determination performance in orbit.

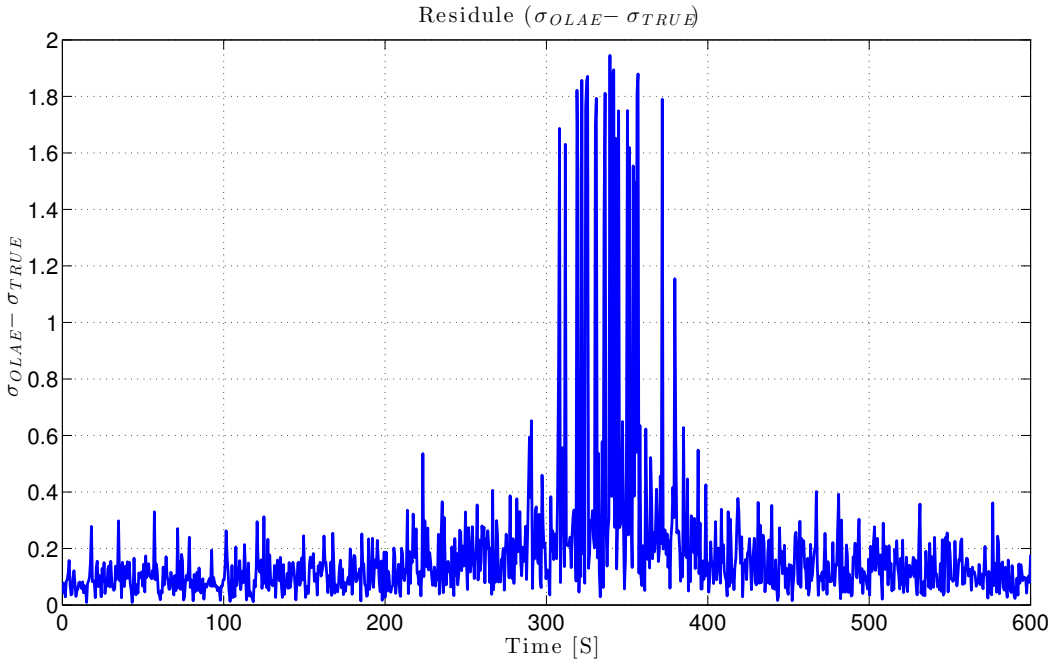


Figure 12. MRP Residules With Observation Noise and False Weighting

References

[1]Ntn tr75 en p036.pdf, 2013.

[2]Sun tracker.pdf, 2013.

[3]H. Schaub. Asen 5010 term project description, 2013.

[4]H. Schaub and J. Junkins. *Analytical mechanics of space systems*. AIAA Education Series. American Institute of Aeronautics and Astronautics, 2003.

IV. Appendix

IV.A. Appendix A: Code for part A

```
function r_N = computeR_N(t)
%>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
%                              computeR_N.m
% Author:   Zach Dischner
% Date:     April 1, 2013
%
% Usage:
%   r_N = computeR_N(t)
%
% Description:  Computes orbit position vector in inertial frame components
%               for ASEN 5010 project circular orbit
%
% Inputs:  t    ==> Time value array [s]. Either row or column vector
%
% Outputs: r_N  ==> Orbit position in N frame components
%                   | r1(t1)    r2(t1)  r3(t1) |
%                   | r1(t2)    r2(t2)  r3(t2) |
%                   | r1(t3)    r2(t3)  r3(t3) |
%                                ...
%<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

%% Orbit Parameters
Omega   = deg2rad(20);      % RAAN?                      [rad]
i       = deg2rad(75);      % Orbit Inclination Angle    [rad]
mu      = 398600;           % Gravatational Parameter    [km^3/s^2]
r       = 6878;             % Circular Orbit Radius      [km]
n       = sqrt(mu/r^3);     % Mean Orbit Rate            [s]
theta_0 = 0;                % Initial Angle              [rad]

%% Compute r_N(t)
if ~iscolumn(t)
    t=t';
end

theta   = theta_0 + n*t;

r_N = r*[ cos(Omega).*cos(theta)-sin(Omega).*sin(theta).*cos(i) , ...
        sin(Omega).*cos(theta)+cos(Omega).*sin(theta).*cos(i) , ...
                              sin(theta).*sin(i)                 ];
```

IV.B. Appendix B: Code for part B

```
    function s_B = computeSunVec_B(sigma_BN)
%>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
%                              computeSunVec_B.m
% Author:   Zach Dischner
% Date:     April 4, 2013
%
% Usage:
%   s_B = computeSunVec_B(sigma_BN)
%
% Description:  Computes the sun attitude attitude as seen by the satellite
%               body. It does so with a constant inertial sun attitude, and
```

```
%               the MRP set sigma_B/N.
%
%               Given s_N, and BN, get s_B
%
% Inputs:  sigma_BN    ==> MRP set describing rotation between B and N
%                          frames
%
% Outputs: s_B  ==> Sun direction vector in B frame components
%
%<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

%% Data Masseuse
if ~iscolumn(sigma_BN)
    sigma_BN = sigma_BN';
end

%% Define Sun Position Vector
s_N = [ 0 ; -1 ; 0 ];

%% Convert MRPs into [BN] DCM
BN  = MRP2C(sigma_BN);

%% Get the Rotation
s_B = BN*s_N;
```

IV.C. Appendix C: Code for part C

```
function TE = Earth2TopoDCM( lambda, phi)
%>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
%                           Earth2TopoDCM.m
% Author:   Zach Dischner
% Date:     April 4, 2013
%
% Usage:
%   TE = Earth2TopoDCM( lambda, phi, inclination )
%
% Description:  Maps the earth-fixed frame [EN] to a topographic frame
%               [TE] ==> {n e d}
%
%               Lamens ==> Rotation matrix between earth-fixed and
%                          topographic frame
%
% Inputs:  lambda ==> Angle between grenwich and equator orbit crossing
%          phi    ==> Along-track angle between satellite and equator plane
%                     (satellite latitude)
% ???      inclination ==> Orbit inclination?
%
% Outputs: TE     ==> DCM from earth fixed frame to the topographic frame
%
%<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<


%% Define Individual Rotations
% First: Rotate LAMBDA about e^3
rot1    = Euler3(lambda);

% Second: Rotate (-PHI  - pi/2)about e^2' (Can be done in just one
% rotation.
rot2    = Euler2(-phi - pi/2);
```

```
% Third: Inclination angle?
inclination = 0;
rot3    = Euler1( inclination );

%% Assemble the Rotation Matrix
TE = rot3*rot2*rot1;
```

IV.D. Appendix D: Code for part D

```
function EN = ECI2ECF( t )
%>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
%                           ECI2ECF.m
% Author:   Zach Dischner
% Date:     April 6, 2013
%
% Usage:
%   EN = ECI2ECF( t )
%
% Description:  Maps the earth-centered inerital frame [N] to an
%               earth-centered earth-fixed frame [E]. Assumes constant
%               rotational rate
%
%               Lamens ==> Rotates ECI to ECF by [EN]
%
% Inputs:  t  ==> Time that has passed since grenwich 0:00:00 ???
%
% Outputs: EN ==> DCM from ECI to ECEF.
%
%<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

%% Obtain Rotation Rate
w_EN = 361 *(pi/180)*(1/24)*(1/3600); %rotation [deg/day][rad/deg][day/hour][hour/sec]

%% Obtain Rotation Angle
gamma_t0 = deg2rad(20);

gamma    = gamma_t0 + w_EN*t;

%% Assemble DCM
EN       = Euler3(gamma);
```

IV.E. Appendix E: Code for part E

```
function M_N = r2MagVec( r_N, t )
%>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
%                           r2MagVec.m
% Author:   Zach Dischner
% Date:     April 6, 2013
%
% Usage:
%   M_N = r2MagVec( r_N, t )
%
% Description:  Compute the Magnetic field vector for a given satellite
%               position and time
%
%               Lamens ==> Find the magnetic field vector at a point in
%                          space and time
```

```
%
% Inputs:    r_N  ==> Inertial satellite position vector [r1 r2 r3]
%            t    ==> Time that has passed since grenwich 0:00:00 ???
%
% Outputs:   M_N  ==> Magnetic field vector in N frame components
%
%<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

%% Defines and Constants
r_E     = 6378; % Earth's Equatorial Radius [km]
r_orbit = norm(r_N);
% lam     = 0.001*sin(t);
% phi     = 0.1*cos(t);
lam     = atan2(r_N(2), r_N(1)); % *look at it again
phi     = asin(r_N(3)/r_orbit);

%% Calculate Magnetic Field Vector
M_T     = -(r_E/r_orbit)^3.*...
                        [ -cos(phi)     sin(phi)*cos(lam)     sin(phi)*sin(lam);...
                            0             sin(lam)              -cos(lam)   ;...
                        -2*sin(phi)  -2*cos(phi)*cos(lam)   -2*cos(phi)*sin(lam)]...
                        *[29900; 1900; -5530];       % nT


EN      = ECI2ECF(t);
TE      = Earth2TopoDCM(lam,phi);

M_N     = EN'*TE'*M_T;
```

IV.F. Appendix F: Code for part H-OLAE Algorithm

```
function q = OLAE(Vb, Vi, w)

%>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
%                           OLAE.m
% Author:   Zach Dischner
% Date:     April 20, 2013
%
% Usage:
%   q = OLAE( Vb, Vi ) (W is optional)
%
% Description:  Uses the Optimal Linear Attitude Estimation (OLAE)
%               algorithm to find a least squares solution for the DCM
%               between the B and N frames.
%
%               Lamens ==> Solves Wabba's problem in a super cool linear
%                          way. Finds [BN]
%
% Inputs:  Vb  ==> Body frame vectors for different measurements
%                  Looks like [ Vb1 Vb2 ...Vbn ]
%                  where Vbn is a column vector
%          Vi  ==> Corresponding inertial frame vectors for different measurements
%                  Looks like [ Vi1 Vi2 ...Vin ]
%                  where Vin is a column vector
% %        W   ==> Optional weights to apply to each measurement
%                   [W1 W2...] becomes | W1 0 ... |
%                                      |  0 W2 ...|
% Outputs: q   ==> Classical Rodriguez param representing rotation between
%                  B and N frames
```

```
%
%<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

%% Assemble Weighting Matrix
numvecs = size(Vb,2);
if nargin < 3
    w = ones(numvecs,1);
end

W = zeros(3*length(w), 3*length(w/norm(w)));
for ii = 1:length(w)
    W(1+3*(ii-1):3+3*(ii-1),1+3*(ii-1):3+3*(ii-1)) = w(ii)*eye(3,3);
end


%% Assemble 'd' and 'S' matrices
d = Vb - Vi;
d = reshape(d,numel(d),1);

s = Vb + Vi;
S = zeros(3*numvecs,3);

for ii=1:numvecs
    S(1+3*(ii-1):3+3*(ii-1),1:3) = tilde(s(:,ii));
end

%% Solve it brah!
q = (S'*W*S)\(S'*W*d);
```

IV.G. Appendix G: Code for part F,G,H and I-Numerical Simulations

```
%>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
%                              intMRPW.m
% Author:   Zach Dischner
% Date:     April 20, 2013
%
% Usage:
%   intMRPW;
%
% Description:  Numerically integrate attitude and omega. Project part f
%
% Inputs:
%
% Outputs:
%
%<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
clc;clear all; close all;

SNR_M = mag2db(1e20);     % SNR of magnetic field vector
SNR_S = mag2db(1e20);     % SNR of sun field vector
weights = [SNR_M SNR_S];  % Weighting matrix is defined relative to the SNR
                          % So a high SNR on the magnetic field vector makes
                          % the relative weighting of that observation go up
weights = [1 1];

% Variables
%-------------------------------------------------
% Initial Attitude
```

```
euler321_init    = deg2rad([5 10 -5])';     % Rad    % Assumes relative to inertial!
% Initial Rate
w_init      = deg2rad([0.4 0.3 0.2])';      % Rad/s
I           = [ 25     2.5       0.5;...
                2.5     20        0;...
                0.5     0         15];
%-------------------------------------------------

% Integration
%-------------------------------------------------
t       = linspace(0,60*10,1000);
X_init  = [w_init ; euler321_init];
X       = zeros(length(t),length(X_init));
X(1,:)  = X_init;
sigma   = zeros(length(t),length(euler321_init));
sigma(1,:) = MRPswitch( Euler3212MRP(euler321_init),1);
w       = zeros(length(t), length(w_init));
w(1,:)  = w_init;

% Initial Values at t=0
s_B     = zeros(length(t),3);
s_B(1,:)= awgn( computeSunVec_B( sigma(1,:) ), SNR_S);

M_B     = zeros(length(t),3);

r_N     = computeR_N(t(1));
EN      = ECI2ECF( t(1) );

r_ECEF  = EN*r_N';
M_N     = r2MagVec( r_ECEF, t(1) );


r_orbit = norm(r_N);
lam     = atan2(r_ECEF(2), r_ECEF(1));
phi     = asin(r_ECEF(3)/r_orbit);


TE      = Earth2TopoDCM(lam,phi);

BN = MRP2C(sigma(1,:)');

M_B(1,:)= awgn(BN*M_N,SNR_M);
s_N     = [0 -1 0]'; % EN'*TE'*s_B(1,:)';
VB = [M_B(1,:)',s_B(1,:)'];
VN = [M_N,s_N];
sigma_OLAE(1,:)  = MRPswitch( Gibbs2MRP( OLAE(VB,VN,weights)),1)';



for ii=2:length(t)

    %% Integrate
   % Find the Derivative
   Xprime = diff321w(t(ii-1),X(ii-1,:),I);

   % General linear integration:
   %    x_(n+1) = x_(n) + x'*delta_t
   X(ii,:) = X(ii-1,:) + Xprime'*(t(ii)-t(ii-1));
   sigma(ii,:) = MRPswitch( Euler3212MRP(X(ii,4:6)),1);
   w(ii,:) = X(ii,1:3);
```

```
    %% Get Sun and Magnetic Field Vectors
    s_B(ii,:)= awgn(computeSunVec_B( sigma(ii,:) ),SNR_S);

%     s_B(ii,:)= computeSunVec_B( sigma(ii,:) ),SNR_S);

    % Variable to hold Magnetic Field vector computations
    r_N      = computeR_N( t(ii-1) );
    r_orbit  = norm( r_N );
    lam      = atan2(r_ECEF(2), r_ECEF(1)); % *look at it again
    phi      = asin(r_ECEF(3)/r_orbit);
    EN       = ECI2ECF( t(ii) );
    TE       = Earth2TopoDCM(lam,phi);

    r_ECEF   = EN*r_N';
    M_N      = r2MagVec( r_ECEF, t(ii) );

    BN = MRP2C(sigma(ii,:)');

    M_B(ii,:)= awgn(BN*M_N,SNR_M);

%    M_B(ii,:)=BN*M_N;
    s_N      = [0 -1 0]'; %EN'*TE'*s_B(ii,:)';   % N frame, s is always [0 -1 0]


%    sigmaTrue(ii,:)   = MRPswitch( C2MRP(TE*EN)' ,1 );
    VB = [M_B(ii,:)',s_B(ii,:)'];
    VN = [M_N,s_N];
    sigma_OLAE(ii,:)  = MRPswitch( Gibbs2MRP( OLAE(VB,VN,weights)),1)';
%    sigmaDiff(:,ii) = sigmaTrue-sigma_OLAE;

%    sigma(ii,:) = MRPswitch(sigma(ii-1,:) + sigmaprime'*(t(ii)-t(ii-1)),1);

     %% Compare OLAE
end

%% MRP and Rate Simulation
figure;set(gcf,'Color',[1 1 1], 'Position',[10 (900)  900 500])
subplot(2,1,1)
plot(t,sigma)
title('MRP Simulation $\sigma_{B/N}$')
xlabel('Time [s]'); ylabel('Modified Rodreguizes')
legend('$\sigma_1$','$\sigma_2$','$\sigma_3$','location','best')

subplot(2,1,2)
plot(t,w);title('Rate Simulation $w_{B/N}$')
xlabel('Time [s]'); ylabel('[Rad/S]')
legend('$w_1$','$w_2$','$w_3$','location','best')

%% Field Vector Sets
figure;set(gcf,'Color',[1 1 1], 'Position',[10 (900)  900 500])
subplot(2,1,1)
plot(t,s_B)
title('Simulated Sun Vector $^Bs$')
xlabel('Time [s]'); ylabel('Sun Vector Components')
legend('$^Bs_1$','$^Bs_2$','$^Bs_3$','location','best')

subplot(2,1,2)
plot(t,M_B)
title('Simulated Magnetic $^BM$')
```

```
xlabel('Time [s]'); ylabel('Magnetic Field Vector Components')
legend('$^BM_1$','$^BM_2$','$^BM_3$','location','best')


%% Difference in Attitudes
figure;set(gcf,'Color',[1 1 1], 'Position',[10 (900)  900 500])
subplot(3,1,1)
plot(t,sigma_OLAE(:,1),t,sigma(:,1))
title('Attitude Estimation Algorithms')
xlabel('Time [s]'); ylabel('$\sigma_1$')
legend('OLAE','True','location','best')

subplot(3,1,2)
plot(t,sigma_OLAE(:,2),t,sigma(:,2))
title('Attitude Estimation Algorithms')
xlabel('Time [s]'); ylabel('$\sigma_2$')
legend('OLAE','True','location','best')

subplot(3,1,3)
plot(t,sigma_OLAE(:,3),t,sigma(:,3))
title('Attitude Estimation Algorithms')
xlabel('Time [s]'); ylabel('$\sigma_3$')
legend('OLAE','True','location','best')

tmp = sigma_OLAE-sigma;
for ii = 1:length(tmp)
   res(ii) = norm(tmp(ii,:));
end

figure;set(gcf,'Color',[1 1 1], 'Position',[10 (900)  900 500])
plot(t,res);
title('Residule ($\sigma_{OLAE} - \sigma_{TRUE}$)')
xlabel('Time [S]'); ylabel('$\sigma_{OLAE} - \sigma_{TRUE}$')
```