AEM 4305 - SADC Project Part 6
MayFly Mission:
Attitude Control

By: Garrett Ailts

May 8 2020

1 Quaternion-Based PD Attitude Control

(a) The PD control law $\tau_b^{control} = -k_p\epsilon - k_d\omega_b^{ba}$ can not be implmented in practice because it is impossible to know the true values of the vector part of the quaternion ϵ and the angular velocity between frame a and b resolved in the body frame ω_b^{ba} with real sensors.

(b) The revised control law $\tau_b^{control} = -k_p\hat{\epsilon} - k_d\hat{\omega}_b^{ba}$ uses the estimates of the two values mentioned in part (a) and thus can be used on a real spacecraft. The angular velocity measurement $\hat{\omega}_b^{ba}$ can be measured directly with a rate gyroscope. The quaternion and its vector part $\hat{\epsilon}$, however, can not be measured directly with a single sensor. To obtain this value, one can use measurements from multiple sensors in conjunction with an algorithm like QUEST or TRIAD to get a direct estimate of the spacecraft's attitude relative to a frame in which the measurements are known (a model of the value in an inertial frame for example). The types of sensors that can be used with these algorithms measure physical vectors resolved in the body frame. Examples are sun sensors, star trackers, magnetometers, Earth horizon sensors, and a solution from multiple gps satellites.

2 Numerical Simulation of PD Attitude Control

(a) The code for this part can be found in the appendix

(b) First, the control law from problem 1 (a) is applied, assuming access to the true values of the angular velocity and quaternion. A simulation of this control law yielded the following plots

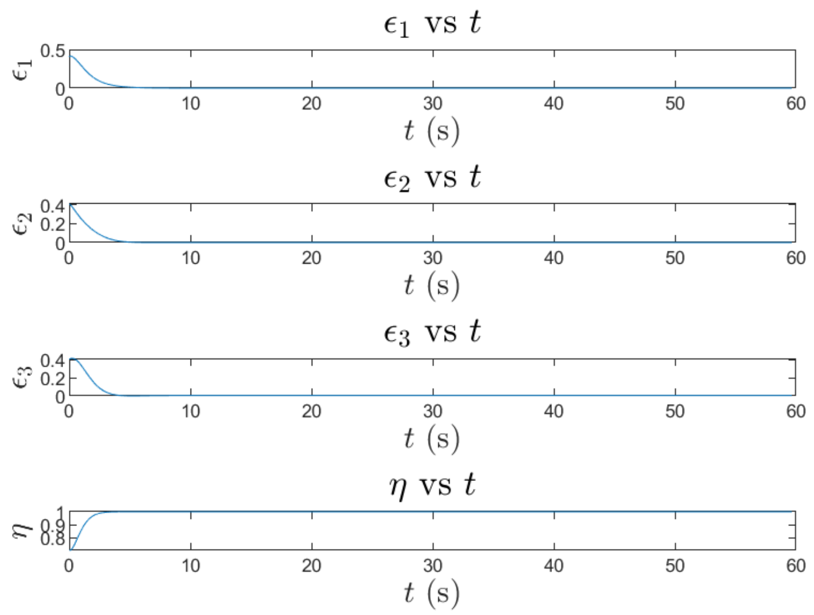

Figure 1: Spacecraft attitude quaternion parameters wrt time

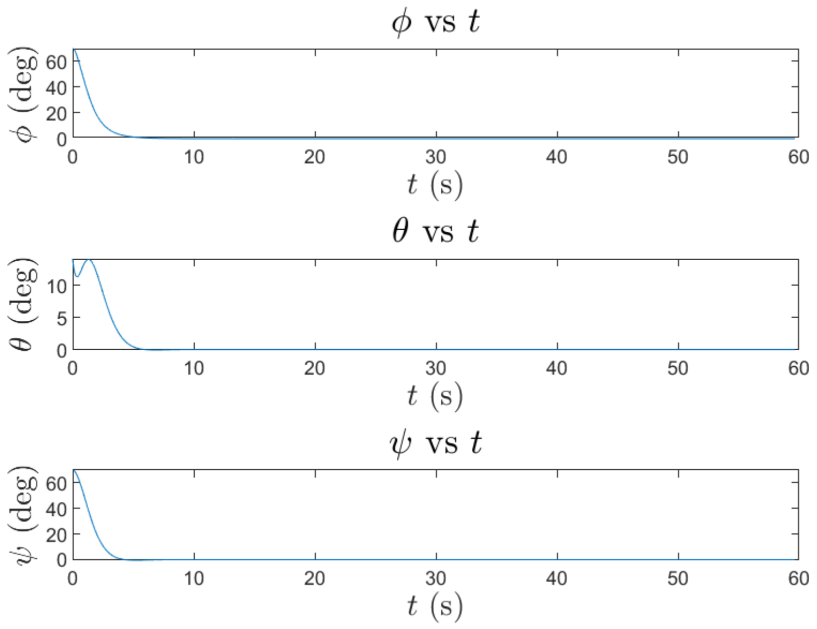

Figure 2: Spacecraft euler angles for a 3-2-1 euler sequence wrt time

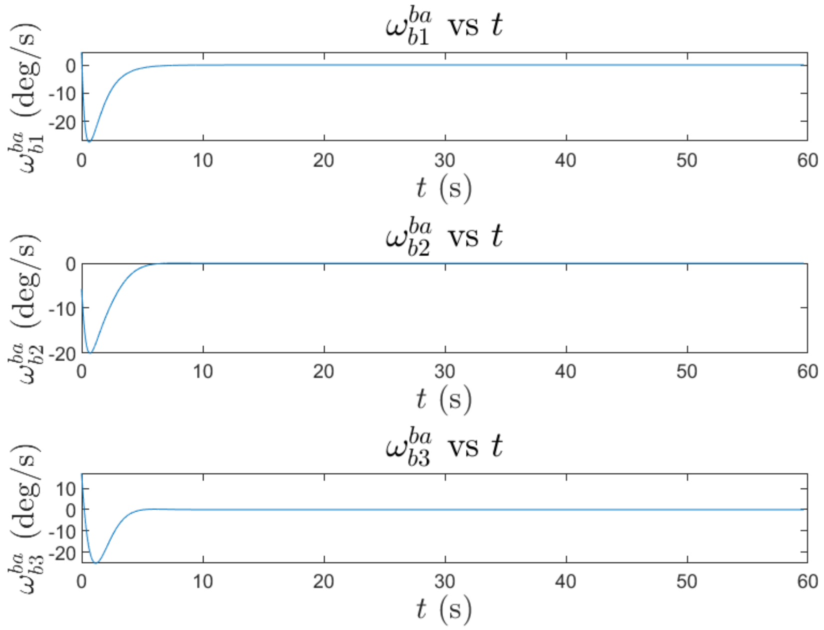

Figure 3: Spacecraft angular rates wrt time

Next, the control law from problem 1 (b) was implemented using the estimates of the attitude and angular velocity by simulating noisy sensors and estimating the attitude using the TRIAD algorithm. The simulation yielded the following results.

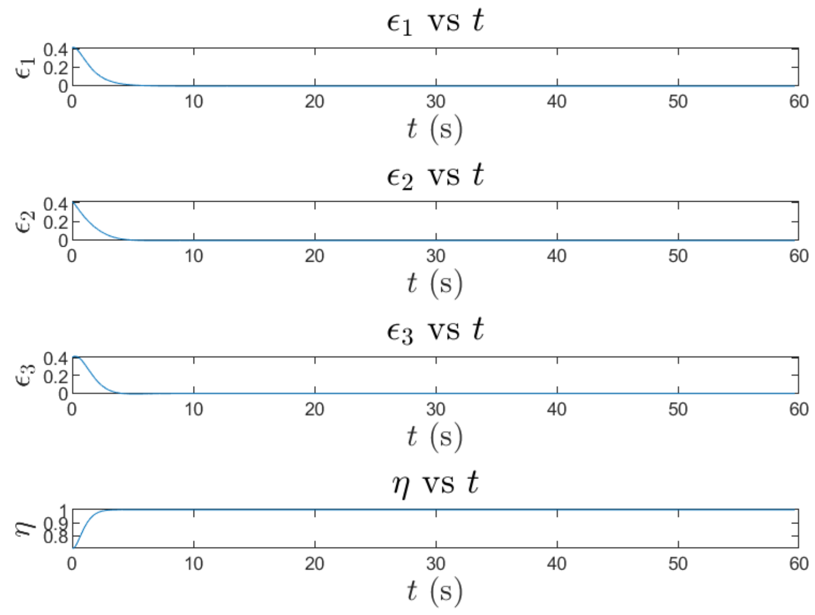

Figure 4: Spacecraft attitude quaternion parameters wrt time

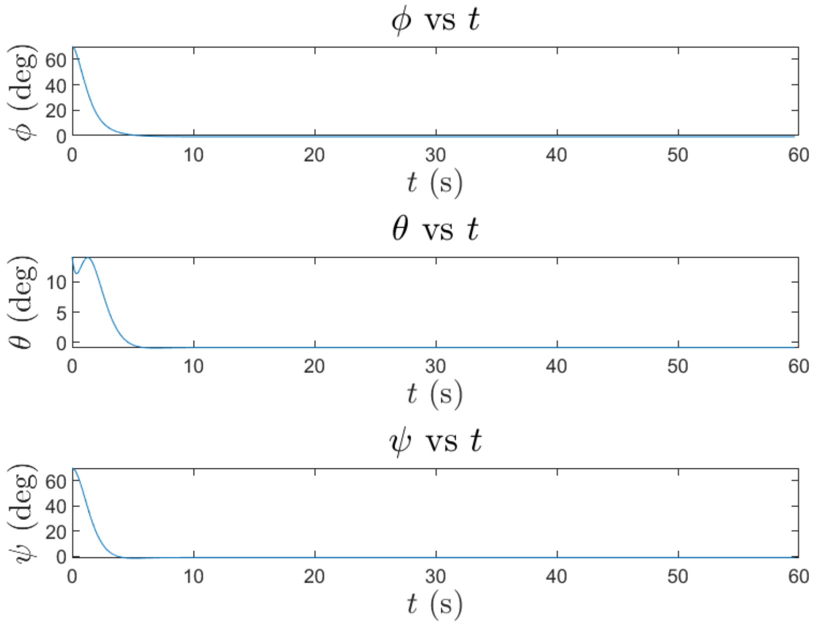

Figure 5: Spacecraft euler angles for a 3-2-1 euler sequence wrt time

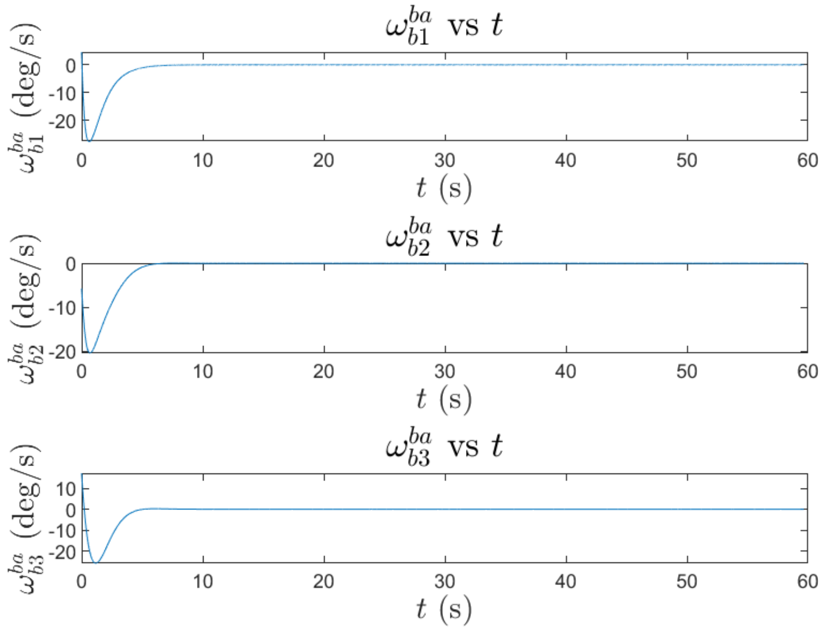

Figure 6: Spacecraft angular rates wrt time

Both control laws were effective at reducing the angular rates to zero and aligning the spacecraft body frame with the inertial frame. When simulating the control laws, values of $k_p = 1.2$ and $k_d = 0.8$ where used for the control gains, which seemed to be sufficient for getting the angular rates to converge to zero or close to it quickly. In practice, the size of the control gains you are able to apply to the control law is limited to the environment you operate in and the actuators you use. An actuator may not be able to physically apply the control effort needed in theory to stabilize the system in a way that is the most desirable. When this happens, an actuator may saturate, or be "wound up" by the controller, causing it to actuate constantly at large efforts or overly high speeds. This can lead to degradation of the actuator and an inability to stabilize the system.

3 Additional Attitude Control/Estimation Simulations

The following sections contain the two additional estimation/control techniques I decided to try.

3.1 Complimentary Filter Attitude Estimation with Bias Estimation

The code for this simulation can be found in the appendix. Below are the results of using the Complimentary Filter with Bias Estimation.

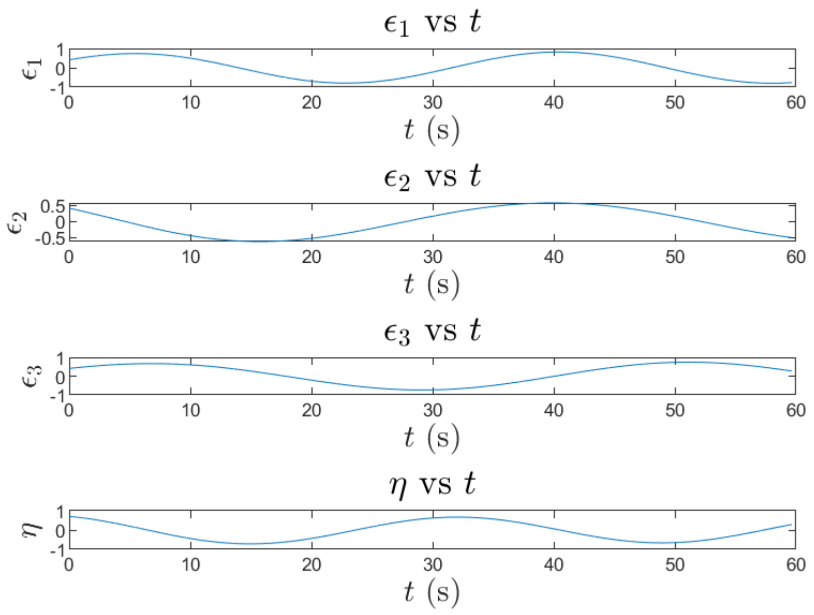

Figure 7: Spacecraft attitude quaternion parameters wrt time

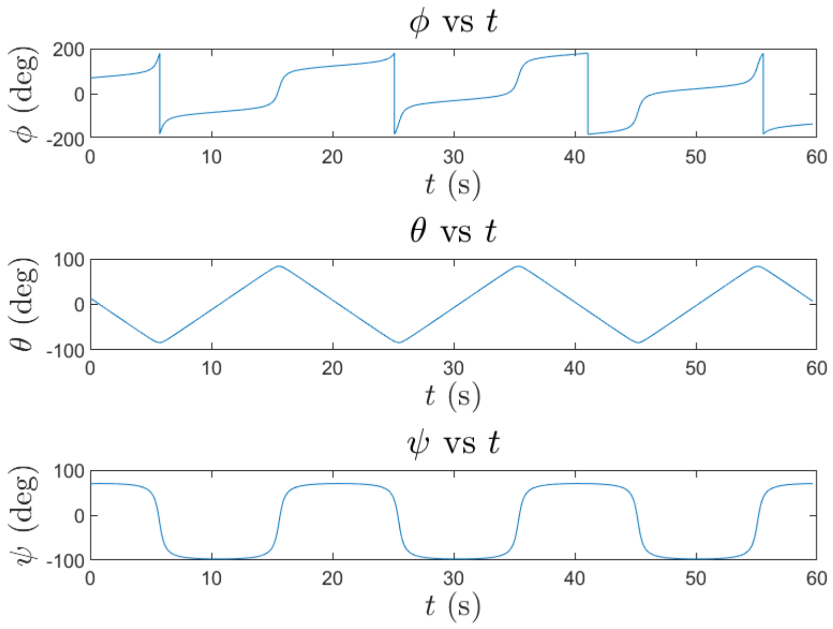

Figure 8: Spacecraft euler angles for a 3-2-1 euler sequence wrt time

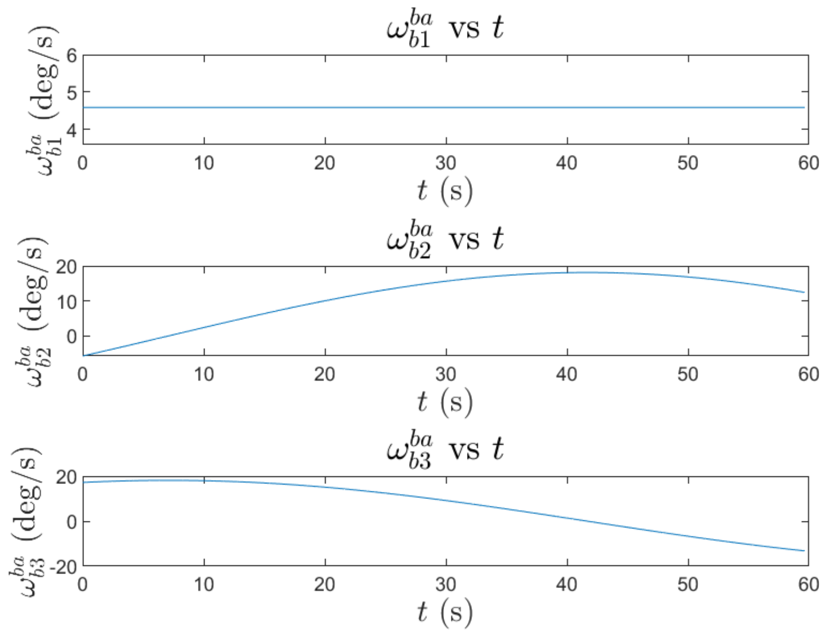

Figure 9: Spacecraft angular rates wrt time

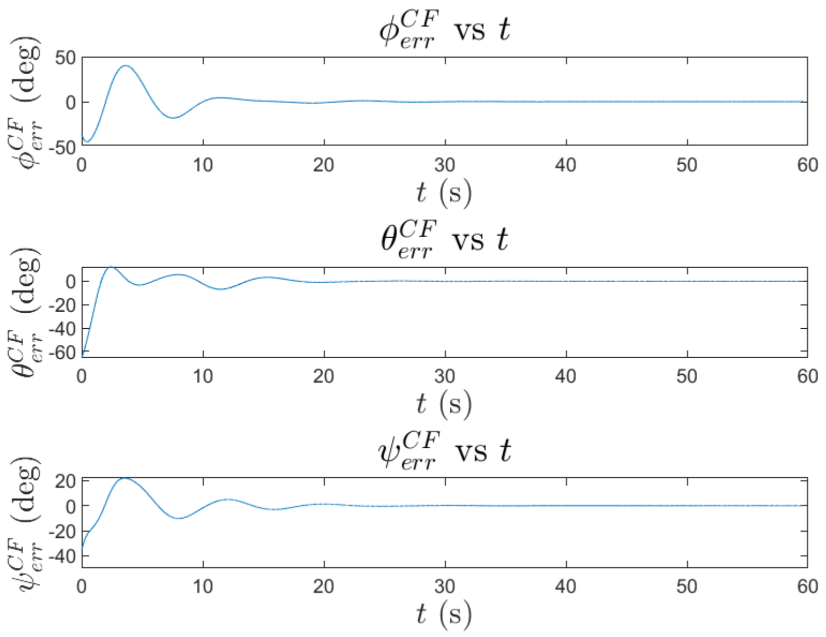

Figure 10: Spacecraft euler angle estimation error for a 3-2-1 euler sequence wrt time

Next, the Complimentary filter was used with active PD control using the control law from problem 1 part (b). Below are the simulation results.

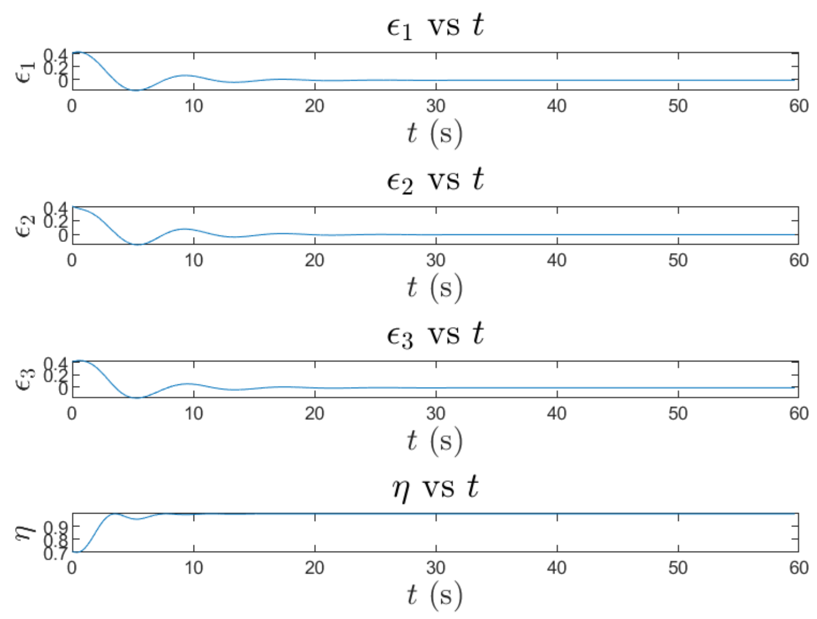

Figure 11: Spacecraft attitude quaternion parameters wrt time

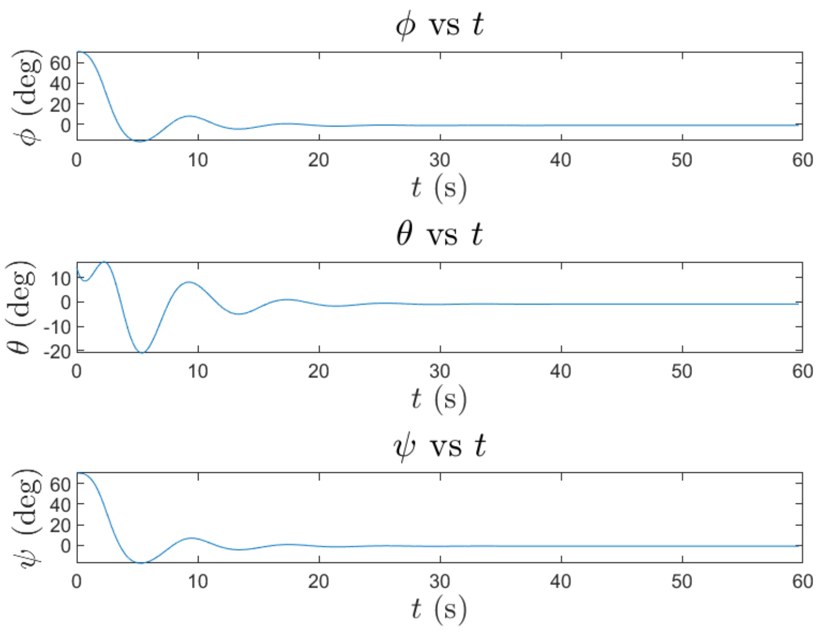

Figure 12: Spacecraft euler angles for a 3-2-1 euler sequence wrt time

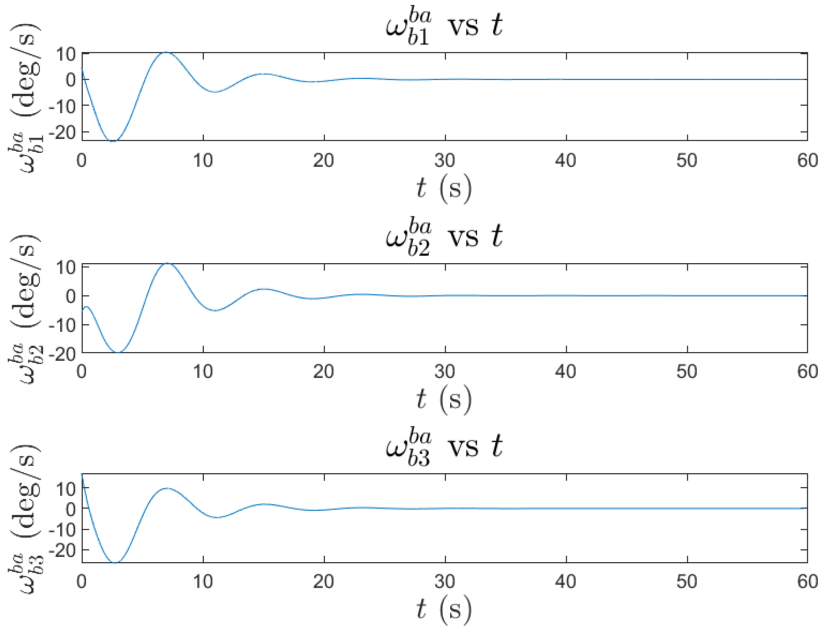

Figure 13: Spacecraft angular rates wrt time

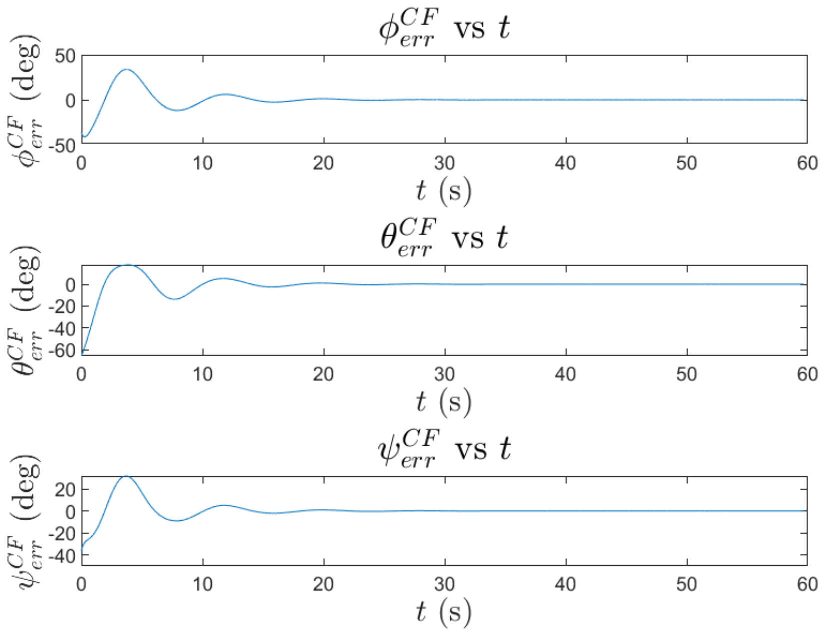

Figure 14: Spacecraft euler angle estimation error for a 3-2-1 euler sequence wrt time

As one can see, the Complimentary Filter is still able to return a good estimate of the attitude of the spacecraft, thought the time to reach steady state error is a bit longer.

3.2 De-Tumble Control with Magnetic Torquers

First, an angular rate feedback de-tumble control law was implemented with the assumption that the true angular rates of the spacecraft were known. In can be seen from the plots generated that the control law starts to reduce the angular rates when simulated over one orbit.

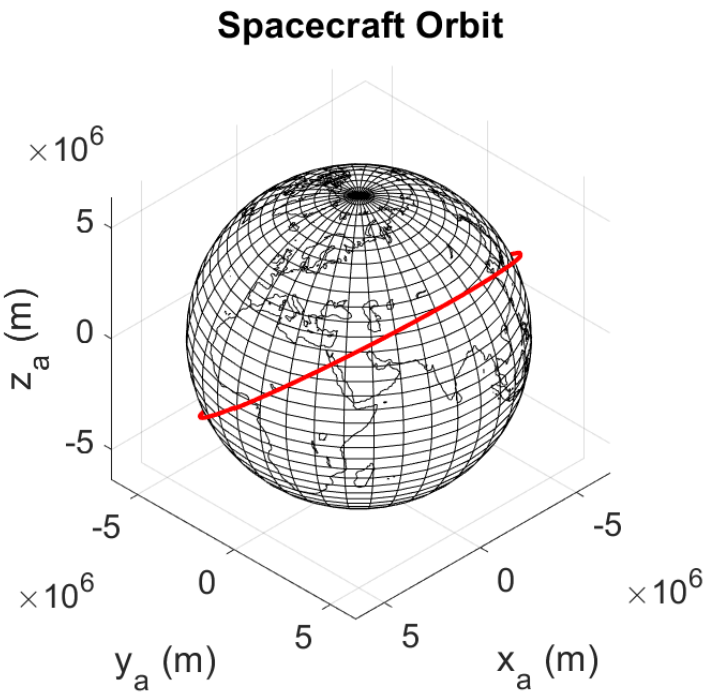

Figure 15: Spacecraft orbit around Earth

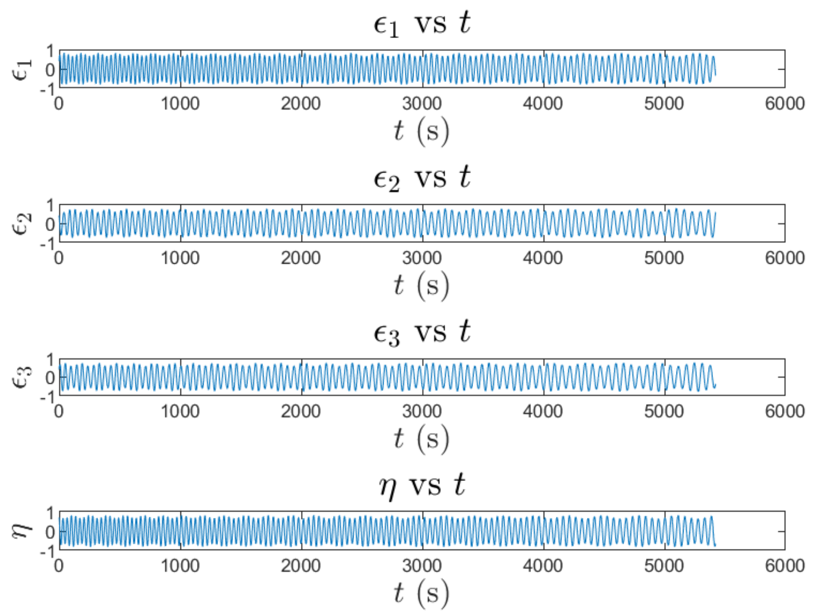

Figure 16: Spacecraft attitude quaternion parameters wrt time

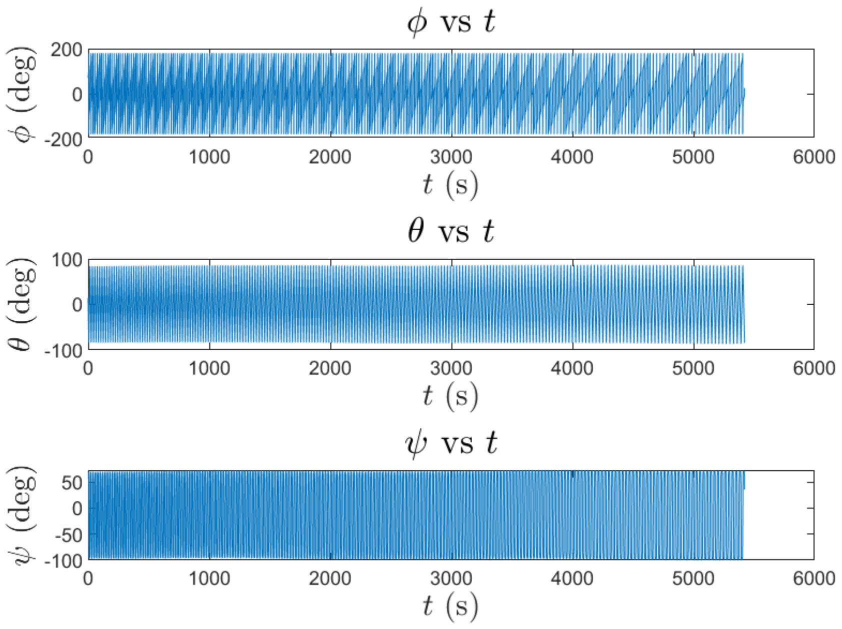

Figure 17: Spacecraft euler angles for a 3-2-1 euler sequence wrt time

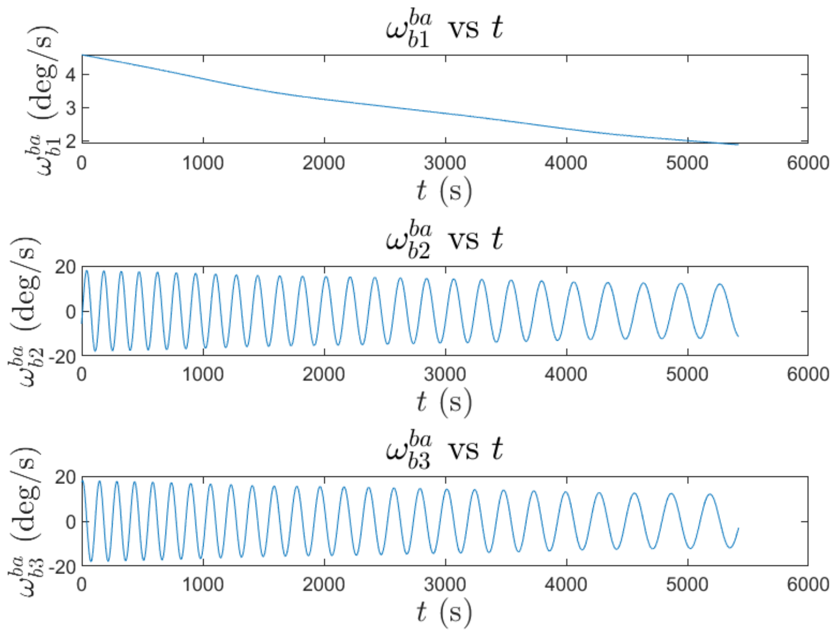

Figure 18: Spacecraft angular rates wrt time

Second, the proportional Bdot control law was used. This controller also successfully reduced the angular rates of a spacecraft and seemed to have near identical performance to the angular rate-feedback law.

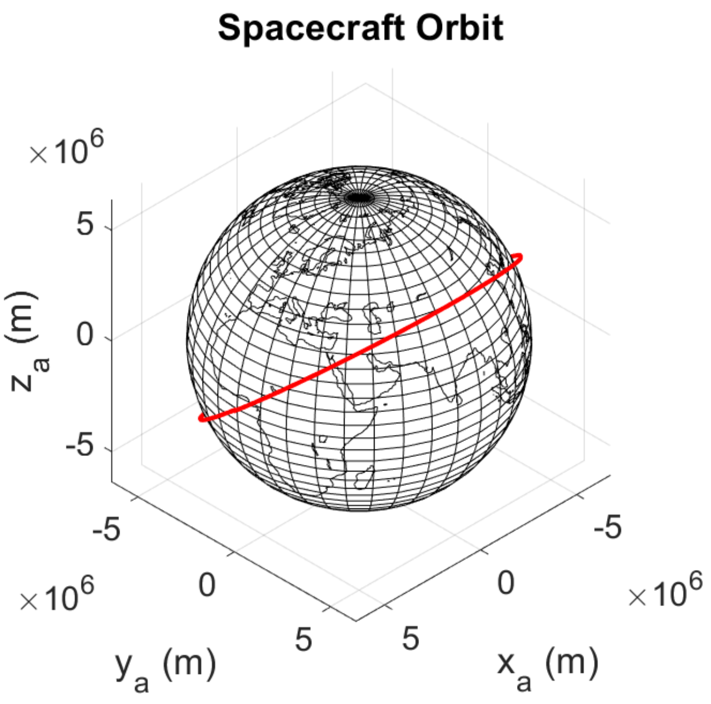

Figure 19: Spacecraft orbit around Earth

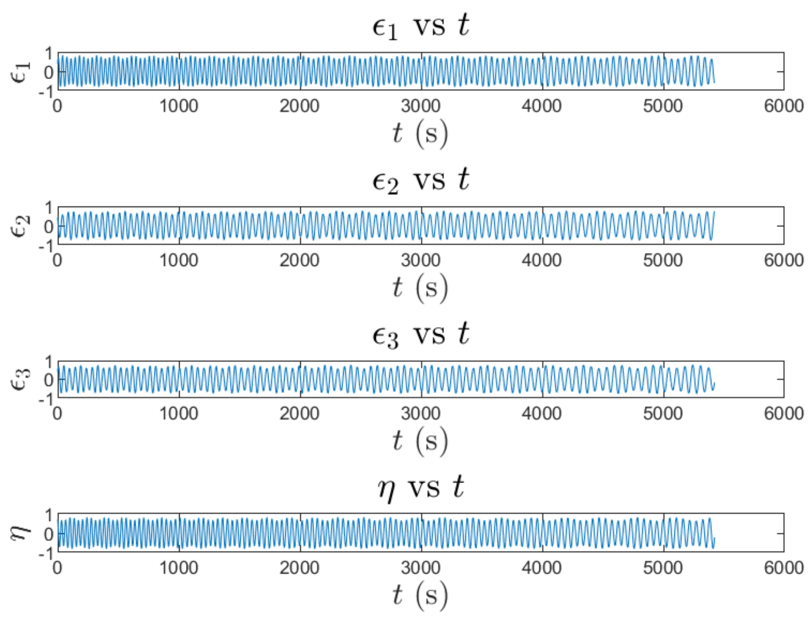

Figure 20: Spacecraft attitude quaternion parameters wrt time

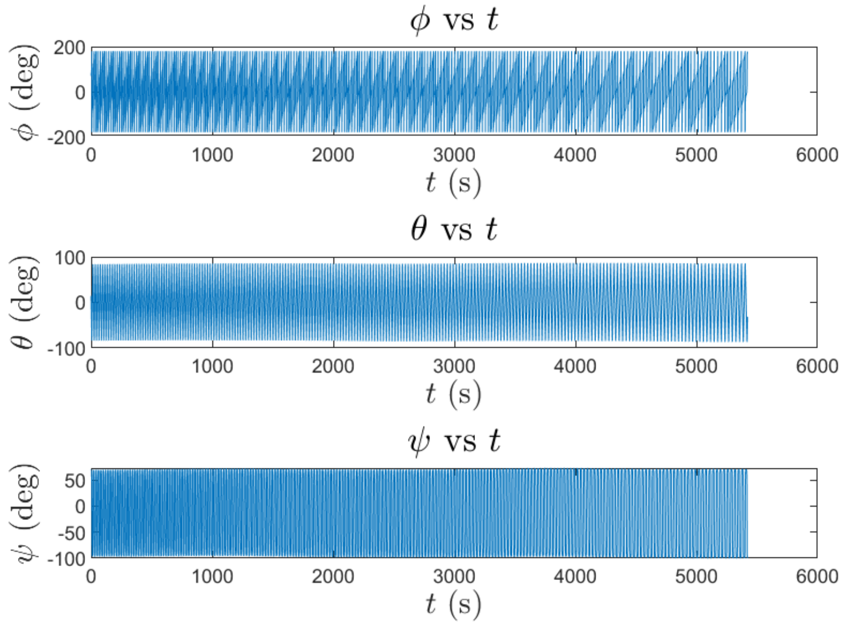

Figure 21: Spacecraft euler angles for a 3-2-1 euler sequence wrt time

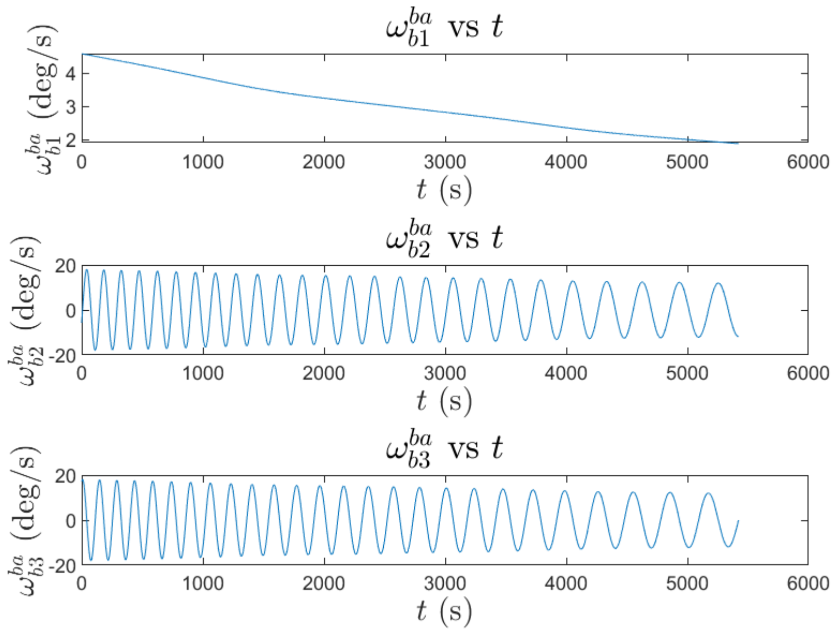

Figure 22: Spacecraft angular rates wrt time

Lastly, a bang-bang implementation of the Bdot control law was implemented. This controller reduced the spacecraft angular rates as well, but was less effective then the previous controllers because the control law takes in to account the limitations of the magnetorquers being used regarding dipole strength .

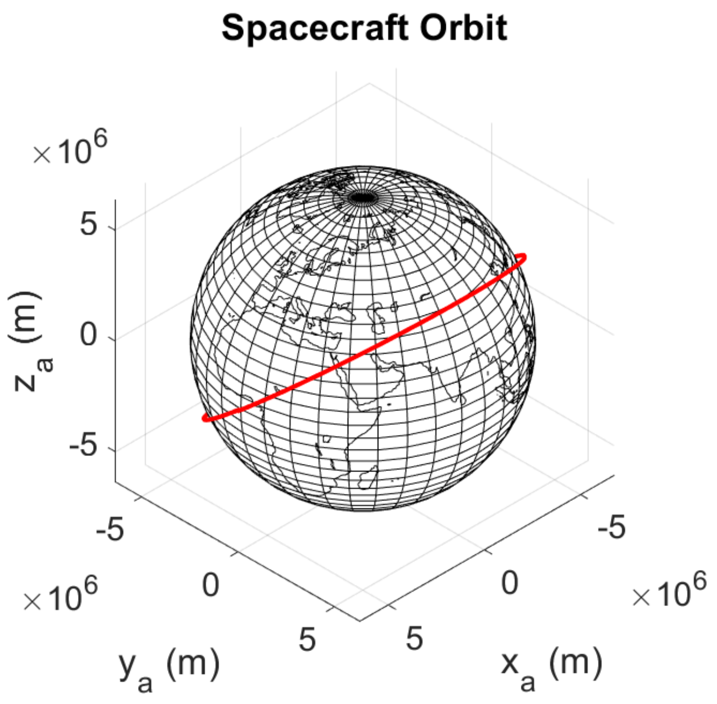

Figure 23: Spacecraft orbit around Earth

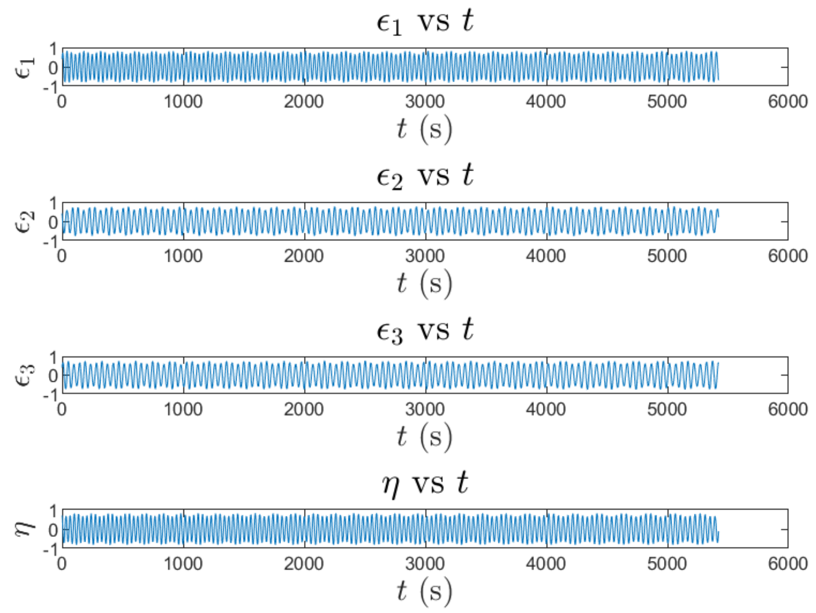

Figure 24: Spacecraft attitude quaternion parameters wrt time

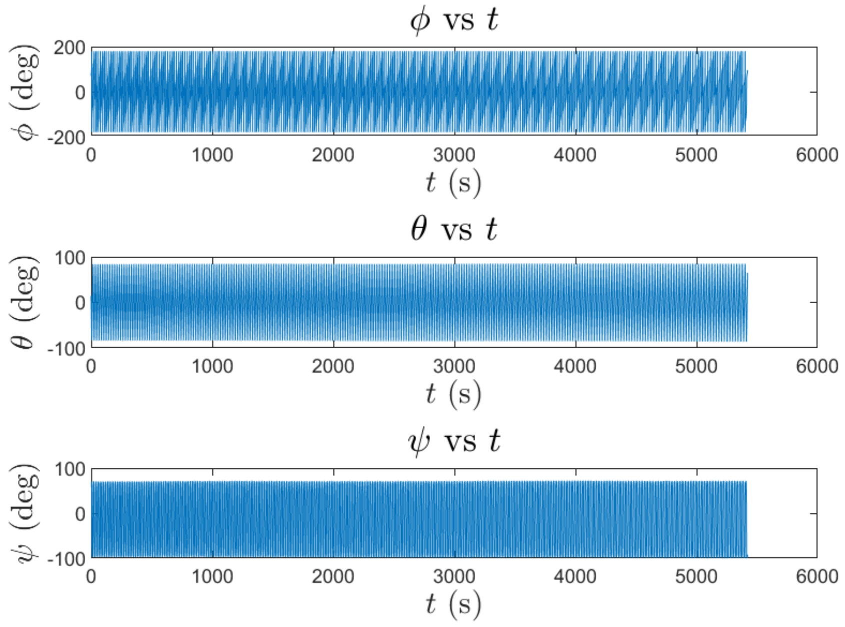

Figure 25: Spacecraft euler angles for a 3-2-1 euler sequence wrt time

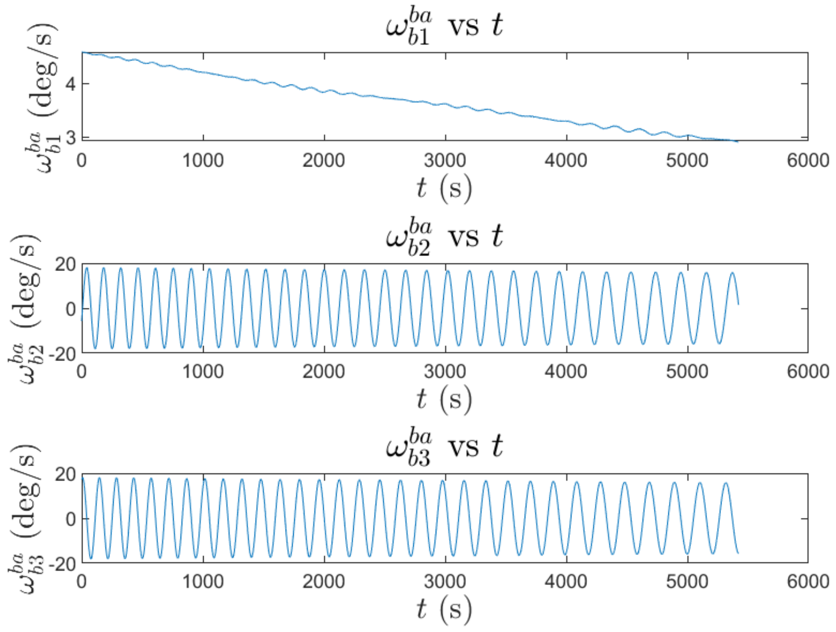

Figure 26: Spacecraft angular rates wrt time

4 Comments and Conclusions

(a) The differential equations of motion of a spacecraft are non-linear, which means that we can't always solve for a single unique solution or even find an analytical solution to the equations of motion for a spaceraft. Using numerical methods, we can perform many computations extremely quickly and model the motion of the spacecraft with fairly high accuracy. The accuracy of this modeled motion depends on the fidelity of the force models we implement and the forces we decide to count for in the model. To include all of these additional forces and solve the differential equations by hand to obtain the same accuracy is virtually impossible. This makes numerical simulation incredibly useful.

(b) Both the total energy and attitude constraint provide ways of checking the validity of your simulation. By turning off all non-conservative forces and torques, the total energy can be used to tell you if the dynamics you are simulating are valid. A spacecraft in orbit in a spherical gravity field should maintain constant total energy ideally. This means, that the change in energy in a numerical simulation with valid dynamics should be of a smaller or equal order of magnitude to the tolerances of the integrator being used. This same idea can be applied to the constraint of the attitude parameterization being used. For a quaternion attitude representation to be valid, it's constraint must remain equal to 1. This means a simulation with a valid attitude representation should have the constraint very on a order of magnitude less than or equal to the magnitude of the integration tolerances being used.

(c) I could add third-body effects from the sun and moon and increase the fidelity of the gravitational model. I actually started to add the Jacchia Atmosphere model to the simulation. This made the computation time quite a bit longer so I did not include the force with any estimation or attitude control method. Here is a graph of the altitude reducing in a pretty eccentric low Earth Orbit.

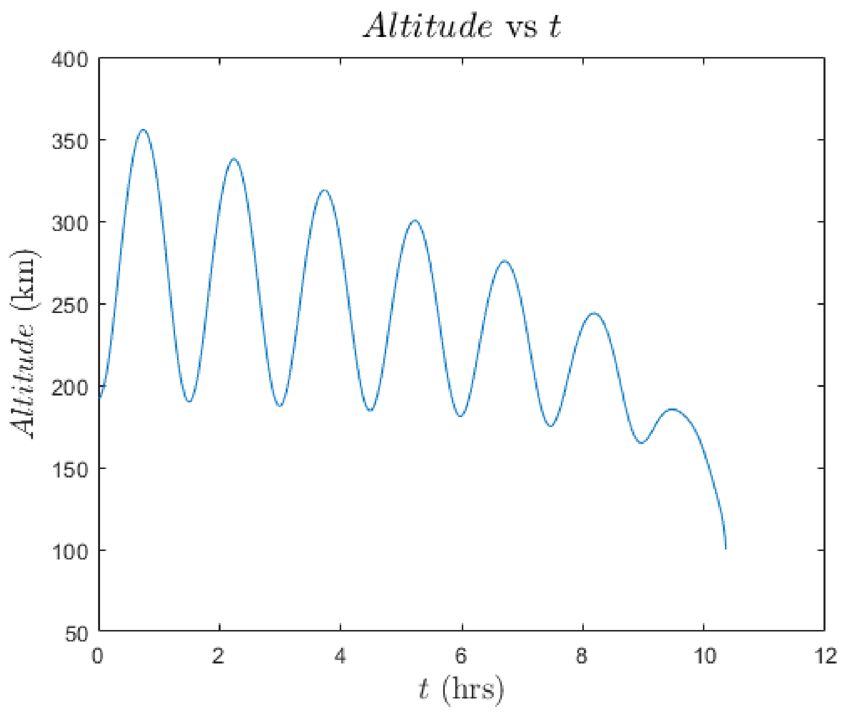

Figure 27: Spacecraft altitude in km vs time in hrs.

This next part is a question so feel free to skip if you don't have the time. I think the altitude is decreasing too quickly, which may be related to how I'm applying the drag force, but I'm unsure. Right now I'm finding the effective wetted area using surface normals and doing the drag calculation as a pressure force. I'm pretty sure the density model is correct. Maybe at altitudes this low, I can't treat the interaction with the atmosphere as Newtonian collisions? Should I be using a spherical drag model with classical aerodynamics?

(d) The error of a TRIAD estimate is based only on the errors present in the sensors and in the models used at the time step the estimate is made. Inertial navigation uses a noisy rate, which means the error is grown during the integration process and compounds. Additionally, gyroscopes have a significant non-zero bias that drifts, where as TRIAD uses sensors that probably have more predictable noise characteristics. The drawback with TRIAD is that the sensors typically used with TRIAD have limited fields of view on what the sense, which means a TRIAD estimate can't always be made. Additionally, TRIAD doesn't deal well with high angular rates, since this can make it hard to get accurate measurements using visual sensors like a star tracker, or sensors like an Earth horizon sensor.

(e) The implementation of the control laws in this course does not take into account the actuator dynamics or the limits on the amount of control effort they can provide. There are also operational limits like actuation speed, lag time, saturation and the tolerances on the expected output to consider.

(f) This project was probably one of the most valuable projects I've ever done in my AEM courses and was by far one of the most fun. Not only did I learn a lot about Spacecraft dynamics and control, but I got to build something pretty substantial with it and that was quite rewarding. I hope to work on spacecraft as a GNC engineer and I will continue to study the topics we were introduced too in this class on my own. I didn't have time with finals, but I plan on doing all the parts in problem 3 that I wasn't able to implement before submitting this. The only suggestion I have is maybe including a little bit about drag modeling for spacecraft since that can be a pretty dominating torque/force in LEO. Other than that, awesome class!

5 Appendix

5.1 Simulation Code for Section 2

Still can't figure out how to make latex include a pdf without skipping a page, sorry.

Table of Contents

.... 1
Constants 1
Extract Parameters 1
Add Spacecraft MOI, Face Objects, and Center of Mass to Params 2
Atmospheric Input Data for Model 2
Assemble Initial Conditions for Spacecraft 2
Simulate 2
Post Process Data 3
Create Plots 3

```
function results = MainSim(params)

% Usage: results = SpacecraftSim(params)
%
% Description: Function takes in a struct of simulation parameters and
% returns a struct containing all of the results of the simulation
%
% Inputs:
%   params  -  struct of parameters for spacecraft, planet, and
 general
%              simaultion parameters
%
% Outputs:
%   results  -  struct of simulation results
%
```

Constants

```
deg2rad = pi/180;
rad2deg = 1/deg2rad;  %#ok<NASGU>
```

Extract Parameters

Sim

```
nOrbits = params.nOrbits;
absTol = params.absTol;
relTol = params.relTol;
atm_model = params.atm_model; %#ok<NASGU>

% Earth
mu = params.Earth.mu_e;
atmDen_mdl = params.Earth.atmDen_mdl;

% Orbit
a = params.sc.sma;
e = params.sc.ecc;
```

```
inc = params.sc.inc*deg2rad;

% Spacecraft Initial Conditions
AttType = params.sc.Attitude_Type;
omega0 = params.sc.omega0;
```

Add Spacecraft MOI, Face Objects, and Center of Mass to Params

```
[IB_b, rcz] = scMOI(params.sc);
params.sc.IB_b = IB_b;
params.sc.rcz = rcz;
params.sc = loadFaces(params.sc);
```

Atmospheric Input Data for Model

```
if strcmp(atmDen_mdl,'Jacchia70')
    LoadJacchia70;
    params.Earth.jacchiaInput = indata;
end
```

Assemble Initial Conditions for Spacecraft

```
OMEGA = 0; % assume perigee at equator with RAAN = 0;
omega = 0;
theta = 0;

[r,v,~] = orbEl2rv(a, e, theta, OMEGA, omega, inc, mu); % transform
                                                        % orbital
 elements
                                                        % to pos. and
 vel.

if strcmp(AttType,'quaternion')
    x0 = [r; v; params.sc.qba0; omega0; params.sc.qbahat0];
elseif strcmp(AttType,'DCM')
    x0 = [r; v; params.sc.Cba0(:); omega0; params.sc.Cbahat0(:)];
else
    error('Incorrect attitude type!\n');
end
```

Simulate

```
event_func = @(t,x) event_function(t,x,params);
options = odeset('AbsTol',absTol,'RelTol',relTol,'Events',event_func);

T = 2*pi*sqrt(a^3/mu);
tspan = [0 nOrbits*T];

[tout,xout] = ode45(@(t,x) dynamics_PD(t,x,params),tspan,x0,options);
```

Post Process Data

```
Post_Process_v4;
results.tout = tout;
results.xout = xout;
results.E = E;
results.eulerAngs = eulerAngs;
results.constraint = constraint;
results.x0 = x0;
```

Create Plots

```
Plotter_v4;
```

Published with MATLAB® R2019b

Table of Contents

.... 1
Extract Parameters from Struct 1
Useful Values 2
Check for Earth Impact and J2 Inclusion 2
Forces and Moments 2
Sensors 3
Navigation 3
Control 4
Orbital Dynamics 4
Attitude Dynamics 4
Package Dynamics Together 5

```
function xdot = dynamics_PD(t,x,params)

% Usage: [tout,xout] = ode45(@(t,x) Coupledyn(t,x,params),tspan,x0,...
%
% Written by Garrett Ailts
%
% Description: Function takes in the current time, state, and a struct
 of
% simulation parameters for a continuouse rigid body's (CRB) angular
% dynamics using the quaternion or DCM representation and returns the
% derivative of the state vector
%
% Inputs:
%   t       -  time since t0 (s)
%   x       -  17 x 1 (quaternion) or 27 x 1 (DCM) state vector
%              representing CRB's attitude and angular rates
%   params  -  struct containing CRB and simulation parameters
%
% Outputs:
%   xdot    -  17 x 1 or 27 x 1 vector containing the rates of change
 for the state
%              paramters
%
```

Extract Parameters from Struct

sim

```
gg_model = params.gg_model;
mag_model = params.mag_model;
atm_model = params.atm_model;
SRP_model = params.SRP_model;
J2on = params.J2on;

% Earth
mu = params.Earth.mu_e;
```

```
R = params.Earth.Rmean;
J2 = params.Earth.J2const;
mag_epoch = params.Earth.mag_epoch;

% spacecraft
I = params.sc.IB_b;
start_epoch = params.sc.start_epoch;
mb = params.sc.mom_b;
est_method = params.sc.est_method;
ctrl_method = params.sc.ctrl_method;
kp = params.sc.kprop;
kd = params.sc.kderiv;
```

Useful Values

```
day2sec = 86400;
I3 = [0 0 1]';
r = norm(x(1:3));
if length(x)==27
    Cba = reshape(x(7:15),[3 3]);
    wba = x(16:18);
else
    Cba = Quat2DCM(x(7:10));
    wba = x(11:13);
end
wbaX = crossMatrix(wba);
```

Check for Earth Impact and J2 Inclusion

```
if r<=R
    warning('Earth impact!')
end
% Check For J2 Inclusion
if ~J2on
    J2 = 0;
end
```

Forces and Moments

gravity gradient torque

```
if gg_model
    tau_gg = (3*mu/r^5)*crossMatrix(Cba*x(1:3))*I*Cba*x(1:3);
else
    tau_gg = 0;
end

% magnetic moment
if mag_model
    telapsed = t+day2sec*(start_epoch-mag_epoch);
    b_a = EarthMagField(x(1:3),telapsed);
    tau_mag = crossMatrix(mb)*Cba*b_a;
```

```
else
    tau_mag = 0;
end

% atmospheric pressure force and torque
if atm_model
    [f_atm, tau_atm] = atmosphereMdl(t,x,Cba,params);
else
    f_atm = 0;
    tau_atm = 0;
end

% solar radiation pressure force
if SRP_model
    f_srp = 0; % placeholder for solar radiation pressure model
               % implementation
else
    f_srp = 0;
end

force = f_atm+f_srp; % sum of forces besides gravity of primary body
mom = tau_gg+tau_mag+tau_atm; % sum of moments imparted on spacecraft
```

Sensors

rate gyroscope

```
wbahat = RateGyroNoisy(wba,t);

% Earth horizon sensor
rpc_b = EarthSensorNoisy(-x(1:3),Cba,t);

% magnetometer
% if ~exist('b_a','var')
%     telapsed = t+day2sec*(start_epoch-mag_epoch);
%     b_a = EarthMagField(x(1:3),telapsed);
% end
b_b = MagnetometerNoisy(b_a,Cba,t);
```

Navigation

```
if strcmp(est_method,'inertial')
    if length(x)==27
        % pull estimate from state vector and ensure its normalized
        C1hat = x(19:21)/norm(x(19:21));
        C2hat = x(22:24)/norm(x(22:24));
        C3hat = crossMatrix(C1hat)*C2hat;
    else
        qhat = x(14:17)/norm(x(14:17));
    end
elseif strcmp(est_method, 'TRIAD')
    Cea = TRIAD(-x(1:3),b_a,rpc_b,b_b);
    if length(x)~=27
```

```
            qhat = DCM2Quat(Cea);
        end

    elseif strcmp(est_method,'none')
        if length(x)==27
            % pull estimate from state vector and ensure its normalized
            C1hat = zeros(3,1);
            C2hat = C1hat;
            C3hat = C1hat;
        else
            qhat = zeros(4,1);
        end
    end
```

Control

```
    if strcmp(ctrl_method,'PD ideal')
        % method a)
        if length(x)==27
            q = DCM2Quat(Cba);
        else
            q = x(7:10);
        end
        tau_ctrl = -kp*q(1:3)-kd*wba;
    elseif strcmp(ctrl_method,'PD true')
        % method b)
        tau_ctrl = -kp*qhat(1:3)-kd*wbahat;
    elseif strcmp(ctrl_method,'none')
        tau_ctrl = 0;
    else
        error('Not a valid control method for this function!\n');
    end

    mom = mom+tau_ctrl;
```

Orbital Dynamics

Calculate xdot1 with gravity and other forces

```
    xdot1 = [x(4:6); -mu*x(1:3)/r^3 + (3*mu*J2*R^2/2/r^5)*(((5/r^2)* ...
                     (x(1:3)'*I3)-1)*x(1:3)-2*(x(1:3)'*I3)*I3) + force];
```

Attitude Dynamics

```
    if length(x)==27
        xdot2 = [-wbaX*x(7:9); -wbaX*x(10:12); -wbaX*x(13:15); ...
                 I\(mom-wbaX*I*wba); -wbahatX*C1hat; -wbahatX*C2hat; ...
                 -wbahatX*C3hat];
        % DCM calc
    else
        xdot2 = [GammaQuat(x(7:10))*[wba;0]; I\(mom-wbaX*I*wba); ...
                 GammaQuat(qhat)*[wbahat;0]];
    % quaternion calculation
end
```

Package Dynamics Together

```
xdot = [xdot1; xdot2];
```

```
end
```

Published with MATLAB® R2019b

Post_Process_v4

Table of Contents

Constants 1
Extract Necessary Parameters 1
Calculate Total Energy, Euler Angles, and Attitude Constraint 1

Written by Garrett Ailts

Constants

```
day2sec = 86400;
```

Extract Necessary Parameters

```
mag_epoch = params.Earth.mag_epoch;
start_epoch = params.sc.start_epoch;
R = params.Earth.Rmean;
```

Calculate Total Energy, Euler Angles, and Attitude Constraint

```
xout = xout';
E = zeros(1,length(tout));
eulerAngs = zeros(3,length(tout));
constraint = zeros(1,length(tout));
q = zeros(4,length(tout));
h = zeros(1,length(tout));

I3 = [0 0 1]';
for lv1 = 1:length(tout)
    r = norm(xout(1:3,lv1));
    telasped = tout(lv1)+day2sec*(start_epoch-mag_epoch);
    h(lv1) = norm(r)-R;
    ba = EarthMagField(xout(1:3,lv1),telasped);
    if strcmp(AttType,'DCM')
        Cba = reshape(xout(7:15,lv1),[3 3]);
        q(:,lv1) = DCM2Quat(Cba);
        constraint(lv1) = det(Cba)-1;
    else
        Cba = Quat2DCM(xout(7:10,lv1));
        constraint(lv1) =
 xout(7:9,lv1)'*xout(7:9,lv1)+xout(10,lv1)^2-1;

    end

    E(lv1) = Etot(xout(:,lv1),r,Cba,ba,params);
    [phi, theta, psi] = DCM2Euler321(Cba);
    eulerAngs(:,lv1) = [phi; theta; psi];
end
if strcmp(AttType,'DCM')
    wba = xout(16:18,:);
else
    q = xout(7:10,:);
    wba = xout(11:13,:);
end
```

Published with MATLAB® R2019b

5.2 Simulation Code for Section 3.1

Still can't figure out how to make latex include a pdf without skipping a page, sorry.

Table of Contents

.... 1
Constants 1
Extract Parameters 1
Add Spacecraft MOI, Face Objects, and Center of Mass to Params 2
Atmospheric Input Data for Model 2
Assemble Initial Conditions for Spacecraft 2
Simulate 2
Post Process Data 3
Create Plots 3

```
function results = CFSim(params)

% Usage: results = SpacecraftSim(params)
%
% Description: Function takes in a struct of simulation parameters and
% returns a struct containing all of the results of the simulation
%
% Inputs:
%   params  -  struct of parameters for spacecraft, planet, and
 general
%              simaultion parameters
%
% Outputs:
%   results  -  struct of simulation results
%
```

Constants

```
deg2rad = pi/180;
rad2deg = 1/deg2rad;  %#ok<NASGU>
```

Extract Parameters

Sim

```
nOrbits = params.nOrbits;
absTol = params.absTol;
relTol = params.relTol;
atm_model = params.atm_model; %#ok<NASGU>

% Earth
mu = params.Earth.mu_e;
atmDen_mdl = params.Earth.atmDen_mdl;

% Orbit
a = params.sc.sma;
e = params.sc.ecc;
inc = params.sc.inc*deg2rad;
```

```
% Spacecraft Initial Conditions
AttType = params.sc.Attitude_Type;
omega0 = params.sc.omega0;
bgyro0 = params.sc.bgyro0;
```

Add Spacecraft MOI, Face Objects, and Center of Mass to Params

```
[IB_b, rcz] = scMOI(params.sc);
params.sc.IB_b = IB_b;
params.sc.rcz = rcz;
params.sc = loadFaces(params.sc);
```

Atmospheric Input Data for Model

```
if strcmp(atmDen_mdl,'Jacchia70')
    LoadJacchia70;
    params.Earth.jacchiaInput = indata;
end
```

Assemble Initial Conditions for Spacecraft

```
OMEGA = 0; % assume perigee at equator with RAAN = 0;
omega = 0;
theta = 0;

[r,v,~] = orbEl2rv(a, e, theta, OMEGA, omega, inc, mu); % transform
                                                        % orbital
 elements
                                                        % to pos. and
 vel.

if strcmp(AttType,'quaternion')
    x0 = [r; v; params.sc.qba0; omega0; params.sc.qbahat0; bgyro0];
elseif strcmp(AttType,'DCM')
    x0 = [r; v; params.sc.Cba0(:); omega0; params.sc.Cbahat0(:);
 bgyro0];
else
    error('Incorrect attitude type!\n');
end
```

Simulate

```
event_func = @(t,x) event_function(t,x,params);
options = odeset('AbsTol',absTol,'RelTol',relTol,'Events',event_func);

T = 2*pi*sqrt(a^3/mu);
tspan = [0 nOrbits*T];
```

```
[tout,xout] = ode45(@(t,x)
 dynamics_CFwB(t,x,params),tspan,x0,options);
```

Post Process Data

```
Post_Process_CF;
results.tout = tout;
results.xout = xout;
results.E = E;
results.eulerAngs = eulerAngs;
results.constraint = constraint;
results.x0 = x0;
```

Create Plots

```
Plotter_CF;
```

Published with MATLAB® R2019b

Table of Contents

.... 1
Extract Parameters from Struct 1
Useful Values 2
Check for Earth Impact and J2 Inclusion 2
Forces and Moments 2
Sensors 3
Navigation 3
Control 4
Orbital Dynamics 4
Attitude Dynamics 4
Package Dynamics Together 5

```
function xdot = dynamics_CFwB(t,x,params)

% Usage: [tout,xout] = ode45(@(t,x) Coupledyn(t,x,params),tspan,x0,...
%
% Written by Garrett Ailts
%
% Description: Function takes in the current time, state, and a struct
 of
% simulation parameters for a continuouse rigid body's (CRB) angular
% dynamics using the quaternion or DCM representation and returns the
% derivative of the state vector
%
% Inputs:
%   t       -  time since t0 (s)
%   x       -  20 x 1 (quaternion) or 30 x 1 (DCM) state vector
%              representing CRB's attitude and angular rates
%   params  -  struct containing CRB and simulation parameters
%
% Outputs:
%   xdot    -  20 x 1 or 30 x 1 vector containing the rates of change
 for the state
%              paramters
%
```

Extract Parameters from Struct

sim

```
gg_model = params.gg_model;
mag_model = params.mag_model;
atm_model = params.atm_model;
SRP_model = params.SRP_model;
J2on = params.J2on;

% Earth
mu = params.Earth.mu_e;
```

```
R = params.Earth.Rmean;
J2 = params.Earth.J2const;
mag_epoch = params.Earth.mag_epoch;

% spacecraft
AttType = params.sc.Attitude_Type;
I = params.sc.IB_b;
start_epoch = params.sc.start_epoch;
mb = params.sc.mom_b;
est_method = params.sc.est_method;
ctrl_method = params.sc.ctrl_method;
kp = params.sc.kprop;
kd = params.sc.kderiv;
ke = params.sc.kerror;
kb = params.sc.kbias;
```

Useful Values

```
day2sec = 86400;
I3 = [0 0 1]';
r = norm(x(1:3));
if strcmp(AttType,'DCM')
    Cba = reshape(x(7:15),[3 3]);
    wba = x(16:18);
else
    Cba = Quat2DCM(x(7:10));
    wba = x(11:13);
end
wbaX = crossMatrix(wba);
```

Check for Earth Impact and J2 Inclusion

```
if r<=R
    warning('Earth impact!')
end
% Check For J2 Inclusion
if ~J2on
    J2 = 0;
end
```

Forces and Moments

gravity gradient torque

```
if gg_model
    tau_gg = (3*mu/r^5)*crossMatrix(Cba*x(1:3))*I*Cba*x(1:3);
else
    tau_gg = 0;
end

% magnetic moment
if mag_model
```

```
    telapsed = t+day2sec*(start_epoch-mag_epoch);
    b_a = EarthMagField(x(1:3),telapsed);
    tau_mag = crossMatrix(mb)*Cba*b_a;
else
    tau_mag = 0;
end

% atmospheric pressure force and torque
if atm_model
    [f_atm, tau_atm] = atmosphereMdl(t,x,Cba,params);
else
    f_atm = 0;
    tau_atm = 0;
end

% solar radiation pressure force
if SRP_model
    f_srp = 0; % placeholder for solar radiation pressure model
               % implementation
else
    f_srp = 0;
end

force = f_atm+f_srp; % sum of forces besides gravity of primary body
mom = tau_gg+tau_mag+tau_atm; % sum of moments imparted on spacecraft
```

Sensors

rate gyroscope

```
wbahat = RateGyroNoisy(wba,t);

% Earth horizon sensor
rpc_b = EarthSensorNoisy(-x(1:3),Cba,t);

if mag_model
    b_b = MagnetometerNoisy(b_a,Cba,t);
end
```

Navigation

```
if strcmp(est_method,'CF')
    CbaTRIAD = TRIAD(-x(1:3),b_a,rpc_b,b_b);
    bgyro = x(end-2:end); % estimate of gyro bias
    if strcmp(AttType,'DCM')
        % pull estimate from state vector and ensure its normalized
        Cea = reshape(x(19:27),[3 3]);
        CbeTRIAD = CbaTRIAD*Cea';
        e = uncross(CbeTRIAD-CbeTRIAD');
        CeaDot = -crossMatrix(wbahat-bgyro-ke*e)*Cea;
        bgyroDot = -kb*e;
    else
        qhat = x(14:17);
        qTRIAD = DCM2Quaternion(CbaTRIAD);
        qerr = 2*GammaQuat(qhat)'*qTRIAD;
        qhatDot = GammaQuat(qhat)*[(wbahat-bgyro
+ke*qerr(4)*qerr(1:3)); 0];
        bgyroDot = -kb*qerr(4)*qerr(1:3);
    end
elseif strcmp(est_method,'none')
    if strcmp(AttType,'DCM')
        % pull estimate from state vector and ensure its normalized
        CeaDot = zeros(3);
    else
        qhatDot = zeros(4,1);
    end
end
```

Control

```
if strcmp(ctrl_method,'PD ideal')
    % method a)
    if strcmp(AttType,'DCM')
        q = DCM2Quat(Cba);
    else
        q = x(7:10);
    end
    tau_ctrl = -kp*q(1:3)-kd*wba;
elseif strcmp(ctrl_method,'PD true')
    if strcmp(AttType,'DCM')
        qhat = DCM2Quat(Cea);
    end
    % method b)
    tau_ctrl = -kp*qhat(1:3)-kd*wbahat;
elseif strcmp(ctrl_method,'none')
    tau_ctrl = 0;
else
    error('Not a valid control method for this function!\n');
end

mom = mom+tau_ctrl;
```

Orbital Dynamics

Calculate xdot1 with gravity and other forces

```
xdot1 = [x(4:6); -mu*x(1:3)/r^3 + (3*mu*J2*R^2/2/r^5)*(((5/r^2)* ...
                 (x(1:3)'*I3)-1)*x(1:3)-2*(x(1:3)'*I3)*I3) + force];
```

Attitude Dynamics

```
if strcmp(AttType,'DCM')
    xdot2 = [-wbaX*x(7:9); -wbaX*x(10:12); -wbaX*x(13:15); ...
             I\(mom-wbaX*I*wba); CeaDot(:); bgyroDot];
    % DCM calc
```

```
else
    xdot2 = [GammaQuat(x(7:10))*[wba;0]; I\(mom-wbaX*I*wba); ...
             qhatDot; bgyroDot];
    % quaternion calculation
end
```

Package Dynamics Together

```
xdot = [xdot1; xdot2];

end
```

Published with MATLAB® R2019b

Post_Process_CF

Table of Contents

Constants 1
Extract Necessary Parameters 1
Calculate Total Energy, Euler Angles, and Attitude Constraint 1

Written by Garrett Ailts

Constants

```
day2sec = 86400;
```

Extract Necessary Parameters

```
mag_epoch = params.Earth.mag_epoch;
start_epoch = params.sc.start_epoch;
R = params.Earth.Rmean;
```

Calculate Total Energy, Euler Angles, and Attitude Constraint

```
xout = xout';
E = zeros(1,length(tout));
eulerAngs = zeros(3,length(tout));
eulerEstErr_CF = zeros(3,length(tout));
constraint = zeros(1,length(tout));
constrainthat = zeros(1,length(tout));
q = zeros(4,length(tout));
h = zeros(1,length(tout));

I3 = [0 0 1]';
for lv1 = 1:length(tout)
    r = norm(xout(1:3,lv1));
    telasped = tout(lv1)+day2sec*(start_epoch-mag_epoch);
    h(lv1) = norm(r)-R;
    ba = EarthMagField(xout(1:3,lv1),telasped);
    if strcmp(AttType,'DCM')
        Cba = reshape(xout(7:15,lv1),[3 3]);
        q(:,lv1) = DCM2Quat(Cba);
        Cea_CF = reshape(xout(19:27,lv1),[3 3]);
        constraint(lv1) = det(Cba)-1;
        constrainthat(lv1) = det(Cea_CF)-1;
    else
        Cba = Quat2DCM(xout(7:10,lv1));
        Cea_CF = Quat2DCM(xout(14:17,lv1));
        constraint(lv1) =
 xout(7:9,lv1)'*xout(7:9,lv1)+xout(10,lv1)^2-1;
        constrainthat(lv1) = xout(14:16,lv1)'*xout(14:16,lv1)+ ...

 xout(17,lv1)^2-1;
    end

    E(lv1) = Etot(xout(:,lv1),r,Cba,ba,params);
    [phi, theta, psi] = DCM2Euler321(Cba);
    Ceb_CF = Cea_CF*Cba'; % error DCM between estimated frame and body
 frame
    [phierr_CF, thetaerr_CF, psierr_CF] = DCM2Euler321(Ceb_CF);
    eulerAngs(:,lv1) = [phi; theta; psi];
    eulerEstErr_CF(:,lv1) = [phierr_CF; thetaerr_CF; psierr_CF];
end
if strcmp(AttType,'DCM')
    wba = xout(16:18,:);
else
    q = xout(7:10,:);
    wba = xout(11:13,:);
end
```

Published with MATLAB® R2019b

5.3 Simulation Code for Section 3.2

Still can't figure out how to make latex include a pdf without skipping a page, sorry.

Table of Contents

.... 1
Constants 1
Extract Parameters 1
Add Spacecraft MOI, Face Objects, and Center of Mass to Params 2
Atmospheric Input Data for Model 2
Assemble Initial Conditions for Spacecraft 2
Simulate 2
Post Process Data 3
Create Plots 3

```
function results = detumbleSim(params)

% Usage: results = SpacecraftSim(params)
%
% Description: Function takes in a struct of simulation parameters and
% returns a struct containing all of the results of the simulation
%
% Inputs:
%   params  -  struct of parameters for spacecraft, planet, and
 general
%              simaultion parameters
%
% Outputs:
%   results  -  struct of simulation results
%
```

Constants

```
deg2rad = pi/180;
rad2deg = 1/deg2rad;  %#ok<NASGU>
```

Extract Parameters

Sim

```
nOrbits = params.nOrbits;
absTol = params.absTol;
relTol = params.relTol;
atm_model = params.atm_model; %#ok<NASGU>

% Earth
mu = params.Earth.mu_e;
atmDen_mdl = params.Earth.atmDen_mdl;

% Orbit
a = params.sc.sma;
e = params.sc.ecc;
inc = params.sc.inc*deg2rad;
```

```
% Spacecraft Initial Conditions
AttType = params.sc.Attitude_Type;
omega0 = params.sc.omega0;
x_f = params.sc.x_filter;
```

Add Spacecraft MOI, Face Objects, and Center of Mass to Params

```
[IB_b, rcz] = scMOI(params.sc);
params.sc.IB_b = IB_b;
params.sc.rcz = rcz;
params.sc = loadFaces(params.sc);
```

Atmospheric Input Data for Model

```
if strcmp(atmDen_mdl,'Jacchia70')
    LoadJacchia70;
    params.Earth.jacchiaInput = indata;
end
```

Assemble Initial Conditions for Spacecraft

```
OMEGA = 0; % assume perigee at equator with RAAN = 0;
omega = 0;
theta = 0;

[r,v,~] = orbEl2rv(a, e, theta, OMEGA, omega, inc, mu); % transform
                                                         % orbital
 elements
                                                         % to pos. and
 vel.

if strcmp(AttType,'quaternion')
    x0 = [r; v; params.sc.qba0; omega0; x_f];
elseif strcmp(AttType,'DCM')
    x0 = [r; v; params.sc.Cba0(:); omega0; x_f];
else
    error('Incorrect attitude type!\n');
end
```

Simulate

```
event_func = @(t,x) event_function(t,x,params);
options = odeset('AbsTol',absTol,'RelTol',relTol,'Events',event_func);

T = 2*pi*sqrt(a^3/mu);
tspan = [0 nOrbits*T];

[tout,xout] = ode45(@(t,x) dynamicsBdot(t,x,params),tspan,x0,options);
```

Post Process Data

```
Post_Process_v4
results.tout = tout;
results.xout = xout;
results.E = E;
results.eulerAngs = eulerAngs;
results.constraint = constraint;
results.x0 = x0;
```

Create Plots

```
Plotter_v4;
```

Published with MATLAB® R2019b

Table of Contents

.... 1
Extract Parameters from Struct 1
Useful Values 2
Check for Earth Impact and J2 Inclusion 2
Forces and Moments 2
Sensors 3
Navigation 3
Control 3
Orbital Dynamics 4
Attitude Dynamics 4
Package Dynamics Together 4

```
function xdot = dynamicsBdot(t,x,params)

% Usage: [tout,xout] = ode45(@(t,x) Coupledyn(t,x,params),tspan,x0,...
%
% Written by Garrett Ailts
%
% Description: Function takes in the current time, state, and a struct
 of
% simulation parameters for a continuouse rigid body's (CRB) angular
% dynamics using the quaternion or DCM representation and returns the
% derivative of the state vector
%
% Inputs:
%   t       -  time since t0 (s)
%   x       -  20 x 1 (quaternion) or 30 x 1 (DCM) state vector
%              representing CRB's attitude and angular rates
%   params  -  struct containing CRB and simulation parameters
%
% Outputs:
%   xdot    -  20 x 1 or 30 x 1 vector containing the rates of change
 for the state
%              paramters
%
```

Extract Parameters from Struct

sim

```
gg_model = params.gg_model;
mag_model = params.mag_model;
atm_model = params.atm_model;
SRP_model = params.SRP_model;
J2on = params.J2on;

% Earth
mu = params.Earth.mu_e;
R = params.Earth.Rmean;
```

```
J2 = params.Earth.J2const;
mag_epoch = params.Earth.mag_epoch;

% spacecraft
AttType = params.sc.Attitude_Type;
I = params.sc.IB_b;
start_epoch = params.sc.start_epoch;
mb_residual = params.sc.mom_b;
est_method = params.sc.est_method;
ctrl_method = params.sc.ctrl_method;

km = params.sc.kmag;
a = params.sc.f_cutoff;
mtorquer = params.sc.mtorquer;
mcoil = params.sc.mcoil;
```

Useful Values

```
day2sec = 86400;
I3 = [0 0 1]';
r = norm(x(1:3));
if strcmp(AttType,'DCM')
    Cba = reshape(x(7:15),[3 3]);
    wba = x(16:18);
else
    Cba = Quat2DCM(x(7:10));
    wba = x(11:13);
end
wbaX = crossMatrix(wba);
```

Check for Earth Impact and J2 Inclusion

```
if r<=R
    warning('Earth impact!')
end
% Check For J2 Inclusion
if ~J2on
    J2 = 0;
end
```

Forces and Moments

gravity gradient torque

```
if gg_model
    tau_gg = (3*mu/r^5)*crossMatrix(Cba*x(1:3))*I*Cba*x(1:3);
else
    tau_gg = 0;
end

% magnetic moment
if mag_model
```

```
    telapsed = t+day2sec*(start_epoch-mag_epoch);
    b_a = EarthMagField(x(1:3),telapsed);
    tau_mag = crossMatrix(mb_residual)*Cba*b_a;
else
    tau_mag = 0;
end

% atmospheric pressure force and torque
if atm_model
    [f_atm, tau_atm] = atmosphereMdl(t,x,Cba,params);
else
    f_atm = 0;
    tau_atm = 0;
end

% solar radiation pressure force
if SRP_model
    f_srp = 0; % placeholder for solar radiation pressure model
               % implementation
else
    f_srp = 0;
end

force = f_atm+f_srp; % sum of forces besides gravity of primary body
mom = tau_gg+tau_mag+tau_atm; % sum of moments imparted on spacecraft
```

Sensors

rate gyroscope

```
%wbahat = RateGyroNoisy(wba,t);

% Earth horizon sensor
%rpc_b = EarthSensorNoisy(-x(1:3),Cba,t);

if mag_model
    b_b = MagnetometerNoisy(b_a,Cba,t);
end
```

Navigation

```
if strcmp(est_method,'LDF')
   x_f = x(end-2:end);
   xdot_f = -a*eye(3)*x_f+a*eye(3)*b_b;
   bhatDot = xdot_f;
elseif strcmp(est_method,'none')
   xdot_f = zeros(3,1);
   bhatDot = zeros(3,1);
end
```

Control

```
if strcmp(ctrl_method,'det1')
```

```
    mb = km*wbaX*b_b/sqrt(b_b'*b_b);
elseif strcmp(ctrl_method,'Bdot')
    mb = -km*bhatDot/sqrt(b_b'*b_b);
elseif strcmp(ctrl_method,'bang-bang')
    m_max = [mcoil; mtorquer; mtorquer]; % axes have different max
 dipoles
    m_max(bhatDot<1e-12) = 0; % dead zone
    mb = -m_max.*sign(bhatDot);
elseif strcmp(ctrl_method,'none')
    mb = 0;
else
    error('Not a valid control method for this function!\n');
end
tau_ctrl = -crossMatrix(b_b)*mb;
mom = mom+tau_ctrl;
```

Orbital Dynamics

Calculate xdot1 with gravity and other forces

```
xdot1 = [x(4:6); -mu*x(1:3)/r^3 + (3*mu*J2*R^2/2/r^5)*(((5/r^2)* ...
                 (x(1:3)'*I3)-1)*x(1:3)-2*(x(1:3)'*I3)*I3) + force];
```

Attitude Dynamics

```
if strcmp(AttType,'DCM')
    xdot2 = [-wbaX*x(7:9); -wbaX*x(10:12); -wbaX*x(13:15); ...
             I\(mom-wbaX*I*wba); xdot_f];
    % DCM calc
else
    xdot2 = [GammaQuat(x(7:10))*[wba;0]; I\(mom-wbaX*I*wba);xdot_f];
    % quaternion calculation
end
```

Package Dynamics Together

```
xdot = [xdot1; xdot2];

end
```

Published with MATLAB® R2019b

Post_Process_v4

Table of Contents

Constants 1
Extract Necessary Parameters 1
Calculate Total Energy, Euler Angles, and Attitude Constraint 1

Written by Garrett Ailts

Constants

```
day2sec = 86400;
```

Extract Necessary Parameters

```
mag_epoch = params.Earth.mag_epoch;
start_epoch = params.sc.start_epoch;
R = params.Earth.Rmean;
```

Calculate Total Energy, Euler Angles, and Attitude Constraint

```
xout = xout';
E = zeros(1,length(tout));
eulerAngs = zeros(3,length(tout));
constraint = zeros(1,length(tout));
q = zeros(4,length(tout));
h = zeros(1,length(tout));

I3 = [0 0 1]';
for lv1 = 1:length(tout)
    r = norm(xout(1:3,lv1));
    telasped = tout(lv1)+day2sec*(start_epoch-mag_epoch);
    h(lv1) = norm(r)-R;
    ba = EarthMagField(xout(1:3,lv1),telasped);
    if strcmp(AttType,'DCM')
        Cba = reshape(xout(7:15,lv1),[3 3]);
        q(:,lv1) = DCM2Quat(Cba);
        constraint(lv1) = det(Cba)-1;
    else
        Cba = Quat2DCM(xout(7:10,lv1));
        constraint(lv1) =
 xout(7:9,lv1)'*xout(7:9,lv1)+xout(10,lv1)^2-1;

    end

    E(lv1) = Etot(xout(:,lv1),r,Cba,ba,params);
    [phi, theta, psi] = DCM2Euler321(Cba);
    eulerAngs(:,lv1) = [phi; theta; psi];
end
if strcmp(AttType,'DCM')
    wba = xout(16:18,:);
else
    q = xout(7:10,:);
    wba = xout(11:13,:);
end
```

Published with MATLAB® R2019b