BACHELOR'S DEGREE IN AEROSPACE VEHICLE ENGINEERING

Study of feasibility of Attitude Control System for a 3U cubesat based on gravity-boom [ANNEXES]

Author: Ana Cambón Periscal

Director: David González Díez

Codirector: Miquel Sureda Anfres

Bachelor of Engineering Thesis

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

June 30, 2020

Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

Contents

1 Introduction **2**

Annex A: maincode.m **3**

Annex B: inertia_c.m **8**

Annex C: dcm2quat.m **12**

Annex D: dynamic.m **15**

Annex E: unwinding.m **18**

Bibliography **20**

1 Introduction

In this report the annexes that have been referred in the main document will be presented.

Annex A: maincode.m

```
%ATTITUDE SIMULATOR

%-------------------------------------------------------------
%% %%%%%%%%%%%%%%%%%%%% INITIALIZE SCRIPT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Close all matlab figures, clear workspace variables and command window.

close all; clear all; clc;

% Start simulation execution timer.
tic;
time = 0;

% Add all the subfolders (and its own subfolders) within the current
% folder to the path.
GeneralPath = genpath(pwd);
addpath(GeneralPath);

%%%%%%%%%%%%%%%%%%%%%% CONFIGURATION FILES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% The configuration files needed for the simulation will be loaded in this
% section.

inertia_c;

 %% Mass Moments of Inertia and Stability Chart

% The global values for the MOI are generated:

global Ix
global Iy

% The global values for the Smelt parameters are generated:

 global K1
 K1 = ((Ix) - (Iy))/(Iz)

 global K2
 K2 = ((Iy) - (Iz))/(Ix)
```

```
% The stability chart used in the report (Kane, Likinsand Levinson's
% Spacecraft Dynamics) is generated in order to check that the satellite
% needs its stability requirements.

x1 = linspace (-1,1,1000);
for i=1:length(x1)
    y1(i)=-x1(i);
end

% The chart is plotted taking into account the Smelt parameters values:
figure(1)
plot(x1, y1, K1, K2, 'xr', 'MarkerSize', 10)
axis([-1 1 -1 1])
grid off
hold on
line([-1, 1], [0, 0])
line([0, 0], [-1, 1])
hold off
title('Stability Chart')
xlabel('K1')
ylabel('K2')
```

```
%% Orbital Parameters

% The orbital parameters for the Earth are defined (could be changed in
% case anothe celestial body was to be used as a center):

R = 6378 + 500; %Earth radius [km]

mu = 398600; % Gravitational parameter [km^3/s^2]

omega = sqrt(mu/(R^3)); % Mean motion

T = ((2*pi)/sqrt(mu))*(R^(3/2)); %Calculation of the period [s].
```

```
%% Initial position (Euler Angles)

% In this section, in order to obatain the quaternions given the Euler
% angles, the function previusly defined will be used:
```

```
phi = 0; % Phi initial value.
theta = 0; %Theta initial value.
psi = 0;  %Psi initial value.

% The function is used:
[q_si, q_xi, q_y3i, q_z4i, A, E] = dcm2quat (phi, theta, psi);


%% Time interval
%In this section the time interval for the simulation will be stablished.

ti = 0; % Initial time.
tf = 16 * T; %This propagation will take about 24 hours.

%% Gravity Gradient simulation

%Angular velocity + quaternion matrix.
x = [0.01*omega, 0.01*omega, omega, q_si, q_xi, q_y3i, q_z4i];


options = odeset('RelTol', 1e-10);

[t, y] = ode45('dynamic', [ti, tf], x, options); %Diferential equation
% solver for the variations on time.

for i = 1:length(t)
    q_sn = y(i, 4); %Quaternion q_s variation w/ time.
    q_xn = y(i, 5); %Quaternion q_x variation w/ time.
    q_yn = y(i, 6); %Quaternion q_y variation w/ time.
    q_zn = y(i, 7); %Quaternion q_z variation w/ time.

    %Validity check caculation for the quaternions.
    A(i) = sqrt(((y(i,4))^2)+((y(i,5))^2)+((y(i,6))^2)+((y(i,7)^2)));

    %Quaternions matrix calculation.
    E = [(1 - 2*(q_xn^2 + q_yn^2)), 2*(q_sn*q_xn + q_yn*q_zn),...
        2*(q_sn*q_yn - q_xn*q_zn); 2*(q_sn*q_xn - q_yn*q_zn), ....
        (1 -2*(q_sn^2 + q_yn^2)), 2*(q_xn*q_yn + q_sn*q_zn);...
        2*(q_sn*q_yn + q_xn*q_zn), 2*(q_xn*q_yn - q_sn*q_zn),...
        (1 - 2*(q_sn^2 + q_xn^2))];
```

```
    yaw(i,1) = acos(E(3,3)) * 180/pi; %Yaw (RAM) calcution trough time.

    pitch(i,1) = acos(E(2,2)) * 180/pi; %Pitch (Crosstrack) calculation
                                        %trough time.

    roll(i,1) = acos(E(1,1)) * 180/pi; %Roll (Nadir) calculation
                                       %trough time.
end

% Yaw variation trough time plot.

figure(2)
plot(t,yaw)
title('Yaw (RAM)')
xlabel('Time [s]')
ylabel('Degrees')

% Pitch variation trough time plot.

figure(3)
plot(t,pitch)
title('Pitch (Crosstrack)')
xlabel('Time [s]')
ylabel('Degrees')

% Roll variation trough time plot.

figure(4)
plot(t,roll)
title('Roll (Nadir)')
xlabel('Time [s]')
ylabel('Degrees')

%Validity check for quaternions plot.

figure(5)
plot(t,A)
axis([0 tf .99 1.01])
title('Validity check for quaternions')
xlabel('Time [s]')
ylabel('Sum of the squares')

toc;
```

Annex B: inertia_c.m

```
% INERTIA CALCULATIONS
%
% The objective of this function is to calculate the inertial matrix for
% each one of the elements that are part of the CubeSat. Once this MOIs
% are calculated for them, the parallel axis will be used in order to
% obtain the final inertia matrix, which will be needed in other scripts.
%
% INPUT:
% · mb, hb, wb, db -> Cubesat body parameters.
%
% · hcb, rcb, mcb -> Cubesat cable connection parameters.
%
% · mgb, rgb -> Cubesat gravity boom parameters parameters.
%
% · sat_m -> Total cubesat mass.
%
% OUTPUT:
%  · Ib/Icc_tras -> Cubesat body MOI (second one is traslated).
%
%  · Icb/Icb_tras -> Cubesat connection cable MOI
%    (second one is traslated).
%
%  · Igb/Igb_tras -> Cubesat gravity boom MOI (second one is traslated).
%
%  · Itotal -> Total inertia matrix (obtained w/parallel axis theorem).
%-----------------------------------------------------------------------

sat_m = 4; %Spacecraft mass (maximum 3U cubesat mass) [kg]

%CUBESAT BODY MOI CALCULATION [kg/m^2]

mb = 2;   %cubesat mass [kg]
hb = 0.1; %cubesat height [m]
wb = 0.3; %cubesat width [m]
db = 0.3405; %cubesat depth [m]

Ib = [(1/12)*mb*(hb^2+db^2) 0 0; 0 (1/12)*mb*(wb^2+hb^2) 0;
    0 0 (1/12)*mb*(wb^2+db^2)];     % Inertia matrix (body)
```

```
%This MOI wont be changing since no parameters will be modified during
%calculations.

%CONNECTION CABLE MOI CALCULATION [kg/m^2]

hcb = 0.5; %Initial cable length [m]
rcb = 0.0175; %Cable radius [m]
mcb = 0.045; %Cable mass [kg]

Icb=[(1/12)*mcb*(3*rcb^2+hcb^2) 0 0; 0 (1/12)*mcb*(3*rcb^2+hcb^2) 0;
    0 0 (1/2)*mcb*rcb^2]; %Inertia matrix.



%GRAVITY BOOM MOI CALCULATION [kg/m^2]

mgb = 0.25; %Initial gravity boom mass [kg]
rgb = 0.5; %Gravity Boom radius [m]

Igb=(2/5)*[mgb*rgb^2 0 0; 0 mgb*rgb^2 0; 0 0 mgb*rgb^2]; %Inertia matrix.



%COORDINATES OF THE CENTER OF EACH OBJECT

%For this calculation x=0 and y=0 are always considered at the center:

cc_pos = [0 0 0]'; %Body cube coordinates
cb_pos = [0 0 -(db/2)-(hcb/2)]'; %Connection cable coordinates
cgb_pos = [0 0 -(db/2)-hcb-rgb]'; %Gravity boom coordinates

%CENTER OF MASS OF EACH OBJECT [m]

sat_cm = (mb*cc_pos+mcb*cb_pos+mgb*cgb_pos)/sat_m; %Spacecraft CoM
cc_cm = sat_cm-cc_pos; %Body cube CoM
cb_cm = sat_cm-cb_pos; %Connecton cable CoM
cgb_cm = sat_cm-cgb_pos; %Gravity boom CoM
```

```
%PARALLEL AXIS THEOREM:

%Considering that the MoI of the main body (cube) suffers no changes:

Icc_tras = Icb+mcb*(dot(cc_cm, cc_cm)*eye(3)-cc_cm*cc_cm');
%Translated inertia body

Icb_tras = Icb+mcb*(dot(cb_cm, cb_cm)*eye(3)-cb_cm*cb_cm');
%Translated inertia connection cable

Igb_tras = Igb+mgb*(dot(cgb_cm, cgb_cm)*eye(3)-cgb_cm*cgb_cm');
%Translated inertia gravity boom

Itotal=Icc_tras+Icb_tras+Igb_tras; %Total inertia matrix.

%MOI for each axis:

Ix = Itotal(1,1);
Iy = Itotal(2,2);
Iz = Itotal(3,3);
```

Annex C: dcm2quat.m

```
% EULER ANGLES TO QUATERNION

% The objective of this function is to convert the given Euler Angles
% (phi, theta and psi) to quaternions (q_s, q_x, q_y, q_z), following the
% 'X Convention'
%
% INPUT:
%  · Phi -> First rotation angle about the z-axis (pitch).
%
%  · Theta -> Second rotation angle between [0, pi] about the y-axis.
%    (roll)
%
%  · Psi -> Third rotation angle about the x-axis (yaw).
%
% OUTPUT:
%  · (q_s) -> First quaternion value.
%
%  · (q_x) -> Second quaternion value.
%
%  · (q_y) -> Third quaternion value.
%
%  · (q_z) -> Fourth quaternion value.
%
%  · A -> Validity check for quaternions.
%
%  · E -> Quaternion matrix.
%
%-------------------------------------------------------------------------

function [q_s, q_x, q_y, q_z, A, E] = dcm2quat(phi, theta, psi)

% Construction of the direction cosine matrix (DCM) baed on Euler angles
% following the 'X convention':

DCM1 = [cosd(phi)*cosd(psi)-sind(phi)*cosd(theta)*sind(psi),
    cosd(phi)*cosd(theta)*sind(psi)+sind(phi)*cosd(psi),
    sind(theta)*sind(psi)];
DCM2 = [-cosd(phi)*sind(psi)-sind(phi)*cosd(theta)*cosd(psi),
    cosd(phi)*cosd(theta)*cosd(psi)-sind(phi)*sind(psi),
    sind(theta)*cosd(psi)];
```

```
DCM3 = [sind(phi)*sind(theta), -cosd(phi)*sind(theta),
    cosd(theta)];


%Obtaining the matrix:

DCM = [DCM1; DCM2; DCM3];

% Extraction of the quaternion values from the DCM:

q_z = (1/2)*sqrt(DCM(1,1)+DCM(2,2)+DCM(3,3)+1);
q_s = (DCM(2,3) - DCM(3,2))/(4*q_z);
q_x = (DCM(3,1) - DCM(1,3))/(4*q_z);
q_y = (DCM(1,2) - DCM(2,1))/(4*q_z);

% Check of the validity of the quaternions, giving that 'A' must
% have a value equal to 1 or really close to it:

A = (q_s)^2 + (q_x)^2 + (q_y)^2 + (q_z)^2;

% Construction of the quaternion matrix given the values e1, e2, e3 and
% e4 (MATRIX E):


E1 = [(1 - 2*(q_x^2+q_y^2)), 2*(q_s*q_x + q_y*q_z), 2*(q_s*q_y - q_x*q_z)];
E2 = [2*(q_s*q_x - q_y*q_z),(1 - 2*(q_s^2+q_y^2)), 2*(q_x*q_y + q_s*q_z)];
E3 = [2*(q_s*q_y + q_x*q_z), 2*(q_x*q_y - q_s*q_z), (1 - 2*(q_s^2+q_x^2))];

E = [E1; E2; E3];

end
```

Annex D: dynamic.m

```
% DYNAMIC ANALYSIS

% The objective of this function is to calculate the dynamic analysis
% of the satellite, in order to obtain the equations of motion, which
% means the angular rates for the cubesat and the variation of the
% quaternions with time, using supplied mass moments of inertia.
%
% INPUT:
% · K1, K2, K3 -> Smelt parameters (three constants of proportionality)
%
%
% OUTPUT:
%  · w1, w2, w3 (as wdot) -> Angular rates for each one of the axis of the
%                            reference frame (body).
%
%  · e1, e2, e3, e4 (as wdot) -> Quaternions variation w.r.t. to time and
%                                orientation.
%
%-----------------------------------------------------------------------

function wdot = dynamic(t,x)

% Global paramenters needed from the main file:


 global K1
 global K2
 global K3


% Orbital parameters (assuming Earth, if another celestial
% body was to be used as reference, this parameters should be
% changed):

R = 6378 + 500; %Earth radius [km]
mu = 398600; % Gravitational parameter [km^3/s^2]

omega = sqrt(mu/(R^3)); % Mean motion
```

```
%Angular velocities and quaternion matrix is generated:

E1 = [(1 - 2*(x(5)^2+x(6)^2)), 2*(x(4)*x(5) + x(6)*x(7)),...
    2*(x(4)*x(6) -(x(5)*x(7))); 2*(x(4)*x(5) - x(6)*x(7)),...
    (1 - 2*(x(4)^2+x(6)^2)), 2*(x(5)*x(6) + x(4)*x(7));...
    2*(x(4)*x(6) + x(5)*x(7)), 2*(x(5)*x(6) - x(4)*x(7)),...
    (1 - 2*(x(4)^2 + x(5)^2))];

% Calculation of the equations of motion for both angular rates and
% quaternions variation w.r.t. time.

wdot(1)= (K1*(x(2)*x(3) - 3*omega^2*E1(2,1) * E1(3,1))); %w1

wdot(2) = (K2*(x(1)*x(3) - 3*omega^2*E1(3,1) * E1(1,1))); %w2

wdot(3) = (K3*(x(1)*x(2) - 3*omega^2*E1(1,1)*E1(2,1))); %w3

wdot(4) = -(1/2) * (-(x(3) + omega)*x(5) + x(2)*x(6) - x(1)*x(7));
%e1 (p_s)

wdot(5) = -(1/2) * (-x(1)*x(6) - x(2)*x(7) + (x(3) + omega)*x(4));
%e2 (p_x)

wdot(6) = -(1/2) * (x(1)*x(5) - x(2)*x(4) - (x(3) - omega)*x(7));
%e3 (p_y)

wdot(7) = -(1/2) * (x(1)*x(4) + x(2)*x(5) + (x(3) - omega)*x(6));
%e4 (p_z)

wdot = wdot';



end
```

Annex E: unwinding.m

```
% UNWINDING PHENOMENA CORRECTION

% The objective of this function is to correct the problem derivated
% from the quaternion unwinding phenomena. As stated in the project
% report, the function will compare two consecutives quaternions,
% and the difference betweeen them is greater than a threshold
% (i.e. 90 degrees), it will be considered that the quaternion has
% suffered the unwinding, so the alternative set of quaternions
% will be chosen.
%
% INPUT:
% · E -> 4xN double matrix, with each column of which containing
%         the quaternion at a time t.
% · E_prev -> 4xN double matrix which in each column contains
%              the quaternion matchning the previous time (t-1).
%
% OUTPUT:
%  · E -> 4xN double matrix, each column containing the quaternion
%         obtained after undergoing the unwinding correction.
%
%------------------------------------------------------------------------

% In order to use this script, both [E] and [E_prev] must be defined.


function [E] = unwinding(E, E_prev)
    ancrit = 90 * pi/180; % Criterio of 90 degrees converted to radians

    vecpart = E(2:4);
    prevvecpart = E_prev(2:4);
    vecpart = vecpart/norm(prevvecpart);
    cosang = dot(vecpart, prevvecpart);
    ang = acos (cosang);

    if ang > ancrit
        E = -E;
    end

end
```

Bibliography

[1] MatLAB Help - MathWorks Webpage [Accesed 24/04/2020]

[2] "Study and development of attitude determination and control simulation software and control algorithms for 3Cat-4 mission" - Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa - Carlos Díez García

[3] "An Analysis of Stabilizing 3U CubeSats Using Gravity Gradient Techniques and a Low Power Reaction Wheel" - California Polytechnic State University - Erich Bender