CubeSat
Simulation

Princeton Satellite Systems, Inc.
6 Market Street, Suite 926
Plainsboro, NJ 08536

Phone: 609-275-9606
Fax: 609-275-9609
Email: map@psatellite.com
Web: www.psatellite.com

1 Introduction

We can't test a CubeSat completely on the ground so we must use simulations. If you have ever played a computer game in which you fly a plane or drive a car you have used a simulation.

2 Differential Equations

Differential equations are used to model the motion of the spacecraft, the motion of the reaction wheels and the charge in the battery.

A simple example is linear motion at a constant velocity. This is a model of a car moving at a constant speed for example. The differential equation is

$$\frac{dx}{dt} = v_x \tag{2-1}$$

$$x(0) = x_0 \tag{2-2}$$

d means "tiny change" This says that the change in the position along the x-axis with respect to time equals the velocity. The second line says that x at time 0 is x_0. We solve this by integrating

$$dx = v_x dt \tag{2-3}$$

$$\int dx = \int v_x dt \tag{2-4}$$

$$x(0) = x_0 \tag{2-5}$$

Since v_x is constant we can bring it outside the integrals

$$\int dx = v_x \int dt \tag{2-6}$$

$$x(0) = x_0 \tag{2-7}$$

$\int dx = x$ and $\int dt = t$ so

$$x(t) = v_x t + x_0 \tag{2-8}$$

In this case the $\int$ makes the d go away. This says the $x(t)$ starts at x_0 and grows linearly with time, t.

The CubeSat equations are more complex and can't be solved analytically as we just did. They must be numerically integrated. In the simplest case we say that

$$\Delta x = v_x \Delta t \tag{2-9}$$

$$x_{k+1} - x_k = v_x(t_{k+1} - t_k) \tag{2-10}$$

$$x_{k+1} = x_k + v_x(t_{k+1} - t_k) \tag{2-11}$$

where k is the step. $k = 1$ for step 1 and $k = 2$ for step 2.

3 The Spacecraft Model

The spacecraft model consists of the 3 position coordinates

$$r = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} \tag{3-12}$$

The three velocity components are

$$v = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \tag{3-13}$$

The orientation is a quaternion which is a 4 element set that uniquely determines the orientation

$$q = \begin{bmatrix} q_s \\ q_x \\ q_y \\ q_z \end{bmatrix} \tag{3-14}$$

You can think of the 4 elements as 3 representing a vector of length 1 (also called a “unit vector”) along the axis of rotation and one representing the angle about that axis.

We need three angular rates

$$\omega = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{3-15}$$

These are like the spin rate of a top.

We need the three angular rates of the reaction wheels

$$\Omega = \begin{bmatrix} \Omega_x \\ \Omega_y \\ \Omega_z \end{bmatrix} \tag{3-16}$$

and finally the battery charge b. We collect them all into a vector

$$x = \begin{bmatrix} r \\ v \\ q \\ \omega \\ \Omega \\ b \end{bmatrix} \tag{3-17}$$

This is the CubeSat state vector. Is is a column of numbers with 17 rows and one column. The differential equations for the CubeSat are

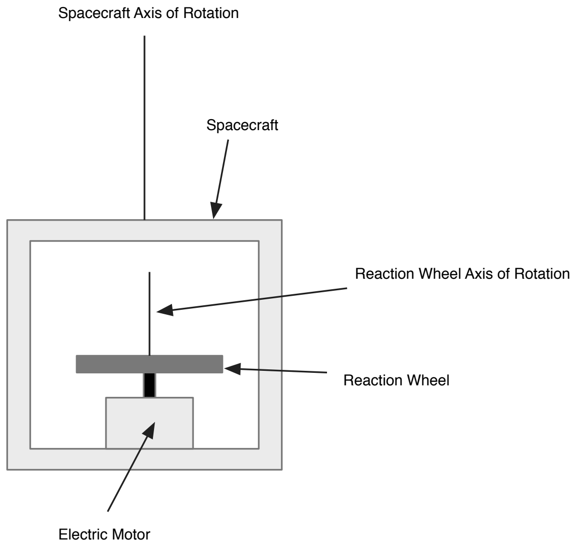
$$\frac{dx}{dt} = f(x, t, u) \tag{3-18}$$

where u are the inputs due to external forces and torques. $f(x, t, u)$ means “a function of” x, u and t.

4 Reaction Wheels

The spacecraft is controlled by reaction wheels. Figure **??** shows the reaction wheel attached to a spacecraft. The reaction wheel is rotated by an electric motor that is attached to the spacecraft. The wheel rotates in one direction and the spacecraft rotates in the opposite direction. The spacecraft “reacts”. Thus the name reaction wheel. Cats and divers twist in the air using the same principal.

Figure 4-1. Reaction wheel attached to a spacecraft. The electric motor turns the wheel and the spacecraft "reacts" and rotates in the opposite direction.



5 Power

The battery is charged by solar pressure. here is the battery equation.

$$\frac{db}{dt} = P_s - P_c \tag{5-19}$$

b is the battery charge, P_s is the power from the solar arrays and P_c is the power consumed by everything on the CubeSat. This says that the change in power stored in the battery equals the power from the solar cells minus all of he power consumed on the CubeSat. If you have any portable electronics devices they work just like this. For example when you are using a cell phone $P_s = 0$. When you hook it into a charger $P_c = 0$. b can never be less than zero and it will always have a maximum charge, b_{max}.

6 The Simulation Script

The script **??** simulates the CubeSat attitude (orientation) dynamics, orbit dynamics, reaction wheels and the battery. Any line starting with at "%" is a comment line. The remaining lines area all code. At the beginning of the script we define constants such as a constant to convert radians to degrees

```
radToDeg = 180/pi;
```

Letters with a "." after them denote a datastructure. This is a convenient way of collecting variables. The following is an excerpt of the data structure `d`.

```
d       = struct;
d.mu = 3.98600436e5; % km3/sec2
d.inertiaRWA = (mass/2)*radius^2; % Polar inertia
```

For example we can pass all the variables in `d` to

```
[xDot, tMag, power] = RHSCubeSatRWA( x, t, d );
```

by just passing the letter `d`.

In MATLAB code we can define variables that are arrays or matrices. For example

```
d.dipole      = [0.0;0;0]
```

is the vector

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{6-20}$$

Semi-colons mean split by row. Commas or spaces mean split by column;

```
q      = [0 0 0]
```

is the vector

$$\begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \tag{6-21}$$

MATLAB uses the notation

```
a = 1.23e10
```

to mean

$$a = 1.23 \times 10^{10} \tag{6-22}$$

The notation

```
z = MyFunction( x, y )
```

means pass the variables `x` and `y` to the function `MyFunction` and return the variable `z`. `MyFunction` can contain large amounts of code. This is a way of encapsulating frequently used code.

Listing 1. Models a CubeSat with reaction wheels and a battery. *CubeSatRWASimulation.m*

```
%-------------------------------------------------------------------------
%   Demonstrate a CubeSat attitude and power system dynamics.
%   The model includes 3 orthogonal reaction wheels.
%-------------------------------------------------------------------------

%-------------------------------------------------------------------------
%   Copyright (c) 2009-2010 Princeton Satellite Systems, Inc.
%   All rights reserved.
%-------------------------------------------------------------------------

% CubeSat type
%-------------
type = '3U';

% Constant
%---------
radToDeg = 180/pi;

% Simulation duration
%--------------------
days  = 0.2;%0.2;
tEnd  = days*86400;

% Time step
%----------
dT    = 1;
nSim  = ceil(tEnd/dT);

% Gravitational parameter for the earth
%--------------------------------------
d    = struct;
d.mu = 3.98600436e5; % km^3/sec^2

% Reaction wheel design
%----------------------
```

```
radius       = 0.040;
density      = 2700; % Aluminum
thickness    = 0.002; % This is 2 mm
mass         = pi*radius^2*thickness*density; % Mass from density x volume
d.inertiaRWA = (mass/2)*radius^2; % Polar inertia

% Add power system model
%-----------------------
d.power.solarCellNormal    = [1 1 -1 -1 0 0 0 0;0 0 0 0 1 1 -1 -1;0 0 0 0 0 0 0 0];
d.power.solarCellEff       = 0.295; % 27% to 29.5% Emcore http://www.emcore.com/solar_photovoltaics/
d.power.effPowerConversion = 0.8;
d.power.solarCellArea      = 0.1*0.116*ones(1,8);
d.power.consumption        = 4;
d.power.batteryCapacity    = 36000; % 360000 J/kg http://en.wikipedia.org/wiki/Lithium-ion_battery

% Initial state vector for a circular orbit
%-----------------------------------------
x    = 6387.165+800; % km
v    = sqrt(d.mu/x);

r    = [x;0;0];                  % Position vector
v    = [0;0;v];                  % Velocity vector
q    = [1;0;0;0];                % Quaternion
w    = [0;0;0];                  % Angular rate of spacecraft
c    = [0;0;0];                  % Reaction wheel rate
b    = 6000;                     % Battery state of charge

% State is [position;velocity;quaternion;angular velocity; battery charge]
% CubeSats are 1 kg per U
%-----------------------------------------------------------------------
x    = [r;v;q;w;c;b];

% Start Julian date
%------------------
d.jD0  = Date2JD([2012 4 5 0 0 0]);
d.rP   = 6378.165;
d.mass = 3; % kg

% CubeSat inertia
%----------------
d.inertia = InertiaCubeSat( type, d.mass );

% Design the PID Controller
%--------------------------
[p.a, p.b, p.c, p.d] = PIDMIMO( 1, 1, 0.01, 200, 0.1, dT );

p.inertia            = d.inertia;
p.max_angle          = 0.01;
p.accel_sat          = [100;100;100];
p.mode               = 1;
p.l                  = [0;0];
p.x_roll             = [0;0];
p.x_pitch            = [0;0];
p.x_yaw              = [0;0];
p.q_target_last      = x(1:4);
p.q_desired_state    = [0;0;0;1];
p.reset              = 0;
p.body_vector        = [0;0;1];

% Planet we are orbiting
%-----------------------
d.planet = 'earth';

% Initialize the plotting aray to save time
%-----------------------------------------
qECIToBody   = x(7:10);
bField       = QForm( qECIToBody, BDipole( x(1:3), d.jD0 ) );
p.eci_vector = Unit(bField);
```

```
angleError   = acos(Dot(p.eci_vector,QTForm(qECIToBody,p.body_vector)))*radToDeg;

xPlot        = [[x;0;0;0;0;angleError;bField;0;0;0] zeros(length(x)+11,nSim)];

% Run the simulation
%-------------------
t = 0;

for k = 1:nSim

    % Quaternion
    %-----------
    qECIToBody   = x(7:10);

    % Magnetic field - the magnetometer output is proportional to this
    %-----------------------------------------------------------------
    bField       = QForm( qECIToBody, BDipole( x(1:3), d.jD0+t/86400 ) );

    % Control system momentum management
    %-----------------------------------
    d.dipole     = [0.0;0;0]; % Amp-turns m^2

    % Reaction wheel control
    %-----------------------
    p.eci_vector = Unit(bField);
    angleError   = acos(Dot(p.eci_vector,QTForm(qECIToBody,p.body_vector)))*radToDeg;
    [torque, p]  = PID3Axis( qECIToBody, p );
    d.tRWA       = -torque;

    % A time step with 4th order Runge-Kutta
    %---------------------------------------
    x            = RK4( @RHSCubeSatRWA, x, dT, t, d );

    % Get the power
    %--------------
    [xDot, tMag, power] = RHSCubeSatRWA( x, t, d );

    % Update plotting and time
    %-------------------------
    hRWA         = x(14:16)*d.inertiaRWA;
    xPlot(:,k+1) = [x;power;torque;angleError;bField;hRWA];
    t            = t + dT;

end

% Plotting
%---------
[t, tL] = TimeLabl( (0:nSim)*dT );

% Y-axis labels
%--------------
yL = {’r_x (km)’ ’r_y (km)’ ’r_z (km)’ ’v_x (km/s)’ ’v_y (km/s)’ ’v_z (km/s)’...
      ’q_s’ ’q_x’ ’q_y’ ’q_z’ ’\omega_x (rad/s)’ ’\omega_y (rad/s)’ ’\omega_z (rad/s)’ ...
      ’\omega_x (rad/s)’ ’\omega_y (rad/s)’ ’\omega_z (rad/s)’ ’b (J)’ ’Power (W)’ ...
      ’T_x (Nm)’ ’T_y (Nm)’ ’T_z (Nm)’ ’Angle Error (deg)’ ’B_x’ ’B_y’ ’B_z’,...
      ’H_x (Nms)’ ’H_y (Nms)’ ’H_z (Nms)’};

% Plotting utility
%-----------------
Plot2D( t, xPlot( 1: 3,:), tL, yL(  1: 3), ’CubeSat Orbit’ );
Plot2D( t, xPlot( 7:10,:), tL, yL(  7:10), ’CubeSat ECI To Body Quaternion’ );
Plot2D( t, xPlot(11:13,:), tL, yL( 11:13), ’CubeSat Attitude Rate (rad/s)’ );
Plot2D( t, xPlot(14:16,:), tL, yL( 14:16), ’CubeSat Reaction Wheel Rate (rad/s)’ );
Plot2D( t, xPlot(17:18,:), tL, yL( 17:18), ’CubeSat Power’ );
Plot2D( t, xPlot(19:22,:), tL, yL( 19:22), ’CubeSat Control Torque’ );
Plot2D( t, xPlot(23:25,:), tL, yL( 23:25), ’CubeSat Magnetic Field’ );
Plot2D( t, xPlot(26:28,:), tL, yL( 26:28), ’CubeSat RWA Momentum’ );
```

CubeSatRWASimulation.m