Booklet

1

SPACECRAFT DYNAMICS SIMULATION

Russell Hawkins, Oxon Hill High School

Dynamics Simulation Procedures

RUSSELL HAWKINS OF OXON HILL HIGH SCHOOL

Dynamics Simulation Procedures

I ask that you please respect my work and keep this booklet (as well as any of my other materials) clean and return them to me or wherever my poster is located when you are finished reading.
Thank You for being interested!

Russell Hawkins
8241 Surratts Rd
Clinton, MD 20735
Phone: 301.433.3297
Email: Russell.aspirations@gmial.com

Table of Contents

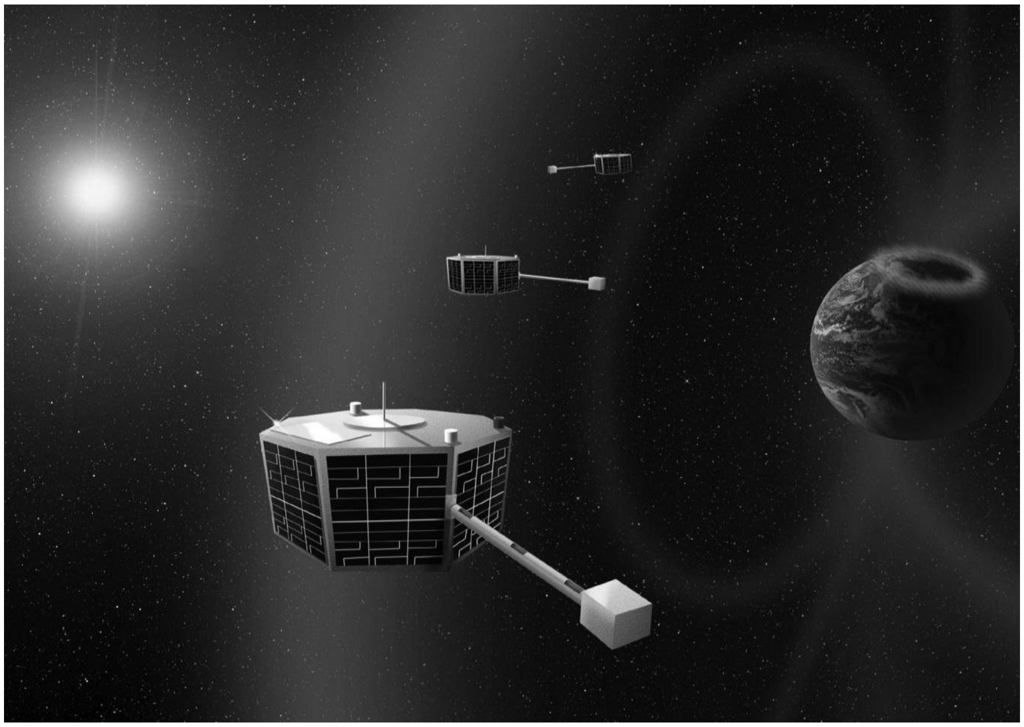
Chapter

1

How This Booklet Is Organized

This chapter is meant to explain what is in this booklet and how this information is laid out through the pages.

This booklet contains the steps that I took to complete the different sections of my project. This booklet is split up by **Section Headers**, which will be written in the same style as this chapter's heading; these headers show which part of the project the following procedures pertain to. Following the headers are the guidelines provided to me by my mentor Richard Harman, along with what it was I was meant to accomplish in each section. Each section is somewhat of an overview of what I would be trying to represent in my code. ***DISCLAIMER:*** This is not an 'academic' text, and therefor does not follow any strict formatting. Please Enjoy!



Chapter 2

01_Quantize Sun Sensor Measurements

For a spinning spacecraft, a slit sun sensor is commonly used to derive a sun angle (referenced from the spacecraft spin axis) and time the sun crossed the slit. The sun data transmitted to the ground are the count representation of the angles. Each count is a fraction of a radian (or degree). In other words, the data is quantized.

Project A: Quantize Sun Angles

The input sun data is in radians. The formula is as follows:

Bias=0.0625 degrees
Scale_factor=0.125 degrees/count

1. Either convert scale factor and bias from degrees to radians or the data to degrees
2. counts=fix((input_sun_data – Bias) / Scale_factor)

fix is a matlab command that converts data to the lower interger (i.e. 1.9 goes to 1).

The sun_angle_project.mat contains the sun angle data for the project. To load it, do the following:

```
>> load sun_angle_project
```

Then arrays that will appear in memory will be the following:

sun_angles_radians(1,28801)
sun_angles_times(28801,1)

Project B: Convert the program to a matlab function with the following:

INPUTS:
Bias
Scale_factor
Sun Angles

OUTPUTS:
Quantized Sun Angles

Lastly, develop a script to do the following:

a. load in the data
b. define scale factor and bias
c. call the function which quantizes the data
d. plot the following data on the same plot
    i. the input sun data (either in radians or degrees) on y-axis
    ii. the quantized sun data (same units as the input sun data) on y-axis
    iii. time in seconds on the x-axis

Chapter 3

02_Time

For this project, time is stored as julian date + fractions of a day. The following algorithms enable you to convert from calendar time to julian date and vice versa.

Calendar Format = YYYYMMDD.HHMMSSmmm

Where YYYY=year
MM=month
DD=day
HH=hours
MM=minutes
SS=seconds
mmm=milli-seconds

For 19700101, YYYY=1970, MM=01, DD=01, HH=00, MM=00, SS=00, mmm=000

jd = 2440588 yields cd = 19700101

Julian Date is the number of days since Noon (12:00 UT) on January 1, 4713 BC.

<u>PART 1 of 2 (Calculate Julian date form input calendar format time)</u>

It can be calculate as follows:

Input:
Calendar_time (YYYYMMDD.HHMMSSmmm)

Output:
JD

Calculate JD as follows

I= fix(I)
J= fix(J)
K= fix(K)

Where

I=Year
J=Month
K=Day

L = fix((J-14)/12)

JD=K – 32075 + fix(1461*(I + 4800 + L)/4)

+ fix(367*(J – 2 - L*12)/12)

- fix(3*fix(((I + 4900 + L)/100)/4))

JD = JD + fraction_of_a_day

Where fraction_of_a day is calculated from HHMMSSmmm

PART 2 of 2 (Calculate Calendar time from Julian Date)

Julian Date can be converted back to calendar format as follows:

Input:

JD=julian date

Output:

Calendar_time (YYYYMMDD.HHMMSSmmm)

$p = \mathrm{fix}(JD) + 68569$

$q = \mathrm{fix}(4*p/146097)$

$r = p - \mathrm{fix}((146097*q + 3)/4)$

$s = \mathrm{fix}(4000*(r+1)/1461001)$

$t = r - \mathrm{fix}(1461*s/4) + 31$

$u = \mathrm{fix}(80*t/2447)$

$v = \mathrm{fix}(u/11)$

$Y = 100*(q-49)+s+v$

$M = u + 2 - 12*v$

$D = t - \mathrm{fix}(2447*u/80)$

Where Y=year, M=month, and D=day and
HHMMSSmmm are calculated from fraction of julian day

Chapter 4

03_Sun Position Calculation

Orbital elements of the Sun (GCI MN2000)

```
N = 0.0
i = 0.0
w = 282.9404 + 4.70935E-5 * d     (degrees)
a = 1.000000  (AU)
e = 0.016709 - 1.151E-9 * d
M = 356.0470 + 0.9856002585 * d   (degrees)
```

(Make sure you convert w & M to radians, *pi/180)

One *Astronomical Unit (AU)* is the Earth's mean distance to the Sun, or 149.6 million km.

rs = 149.6e6

d = jd_in – jd(19991231.0);

where jd_in is the julian date for your input time.

The position of the Sun

The position of the Sun is computed just like the position of any other planet, but since the Sun always is moving in the ecliptic, and since the eccentricity of the orbit is quite small, a few simplifications can be made. Therefore, a separate presentation for the Sun is given.

Of course, we're here really computing the position of the Earth in its orbit around the Sun, but since we're viewing the sky from an Earth-centered perspective, we'll pretend that the Sun is in orbit around the Earth instead.

First, compute the eccentric anomaly E from the mean anomaly M and from the eccentricity e (E and M in radians):

```
      E = M + e * sin(M) * ( 1.0 + e * cos(M) )
```

Note that the formulae for computing E are not exact; however they're accurate enough here.

Then compute the Sun's distance r and its true anomaly v from:

```
   xv = cos(E) - e
   yv = sqrt(1.0 - e*e) * sin(E)
   v = atan2( yv, xv )
   r = sqrt( xv*xv + yv*yv )
```

(note that the r computed here is later used as rs and is an AU which is defined above in km)

Now, compute the Sun's true longitude:

```
   lonsun = v + w
```

Convert lonsun,r to ecliptic rectangular geocentric coordinates xs,ys:

```
    xs = r * cos(lonsun)
    ys = r * sin(lonsun)
```

(since the Sun always is in the ecliptic plane, zs is of course zero). xs,ys is the Sun's position in a coordinate system in the plane of the ecliptic. To convert this to equatorial, rectangular, geocentric coordinates, compute:

```
    ecl = 23.4393 - 3.563E-7 * d   (degrees, make
sure you convert to radians)

    xe = xs * rs
    ye = ys * cos(ecl) * rs
    ze = ys * sin(ecl) * rs
```

Finally, compute the Sun's Right Ascension (RA) and Declination (Dec):

```
    RA  = atan2( ye, xe )
    Dec = atan2( ze, sqrt(xe*xe+ye*ye) )
```

Chapter 5

04_Orbital Dynamics

If the Earth is assumed to be a point mass, the force due to gravity is:

$$F_g = \frac{G * M_E * m_{sc} * \overline{r}}{\left|\overline{r}\right|^3}$$

where

$G * M_E = 398,600.4418 \ \left(km^3 s^{-2}\right)$
m_{sc} = Spacecraft mass (kg)

From Newton's Second Law:

F = mass*acceleration

$$m_{sc} * acceleration_{sc} = - F_g$$

$$\overline{a}_{sc} = \frac{- G * M_E * \overline{r}}{\left|r^3\right|}$$

where, $\overline{a}_{sc}$ is the spacecraft acceleration.

For the Ordinary Differential Equation (ODE) state, use the following:

X(1:3,1)=position (km)
X(4:6,1)=velocity (km/sec)

Now to solve this equation, I had use a 4th order Runge-Kutta algorithm:

Fourth Order Runge-Kutta Propagator:

Let an initial value problem be specified as follows:

$$y' = f(t,y) \quad \text{and} \quad y(t_0) = y_0$$

In other words, the rate at which y changes is a function of both y and t (time). At time t_0, y equals y_0. The 4x4 Runge-Kutta (RK4) approximation for a problem of this form is given by:

$$y_{i+1} = y_i + \tfrac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)\Delta t$$
$$t_{i+1} = t_i + \Delta t$$

…where y_{i+1} is the RK4 approximation for $y(t_{i+1})$, and:

$$k_1 = f\left[(t_i),(y_i)\right]$$
$$k_2 = f\left[\left(t_i + \tfrac{1}{2}\Delta t\right),\left(y_i + \tfrac{1}{2}k_1\Delta t\right)\right]$$
$$k_3 = f\left[\left(t_i + \tfrac{1}{2}\Delta t\right),\left(y_i + \tfrac{1}{2}k_2\Delta t\right)\right]$$
$$k_4 = f\left[\left(t_i + \tfrac{1}{2}\Delta t\right),\left(y_i + k_3\Delta t\right)\right]$$

Therefore, the next value of (y_{i+1}) is determined by the present value (y_i) plus the weighted average of 4 deltas, where each delta is the product of the size of the interval (Δt) and an estimated slope $\Delta t\,(dy/dt) = \Delta y$.

- k_1 is a delta based on the slope at the beginning of the interval, using y_i (Euler's method).
- k_2 is a delta based on the slope at the midpoint of the interval, using $y_i + \frac{1}{2}k_1$.
- k_3 is also a delta based on the slope at the midpoint of the interval, this time using $y_i + \frac{1}{2}k_2$.
- k_4 is a delta based on the slope at the end of the interval, using $y_i + k_3$.

In averaging the four deltas, greater weight is given to the deltas at the midpoint so that:

$$\Delta y_i = \tfrac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)\Delta t$$

The RK4 method is a fourth-order method, meaning that the error per step is on the order of Δt^5, while to total accumulated error is on the order of Δt^4.

PROJECT: Propagate an initial position and velocity vector using a 4th order Runge-Kutta

For y0 use the following:

```
y0(1:3,1) = [ 1.939171267526330e+03;
              7.053946077269909e+03;
              5.216461164024867e+03];  % initial position in km
y0(4:6,1) = [  7.728790449144201e-01;
              -5.188238121076679e+00;
               3.991379730081099e+00];%  initial velocity in km/sec
```

For the timespan, start at time 0 and go for a 3420 seconds with a delta time of 60 seconds. Compare results to project4_data in project4.mat file where project4_data has the following format:
Column 1 = time (YYMMDD.HHMMSSmmm)
Column 2-4 = position (km)
Column 5-7 = velocity (km/sec)

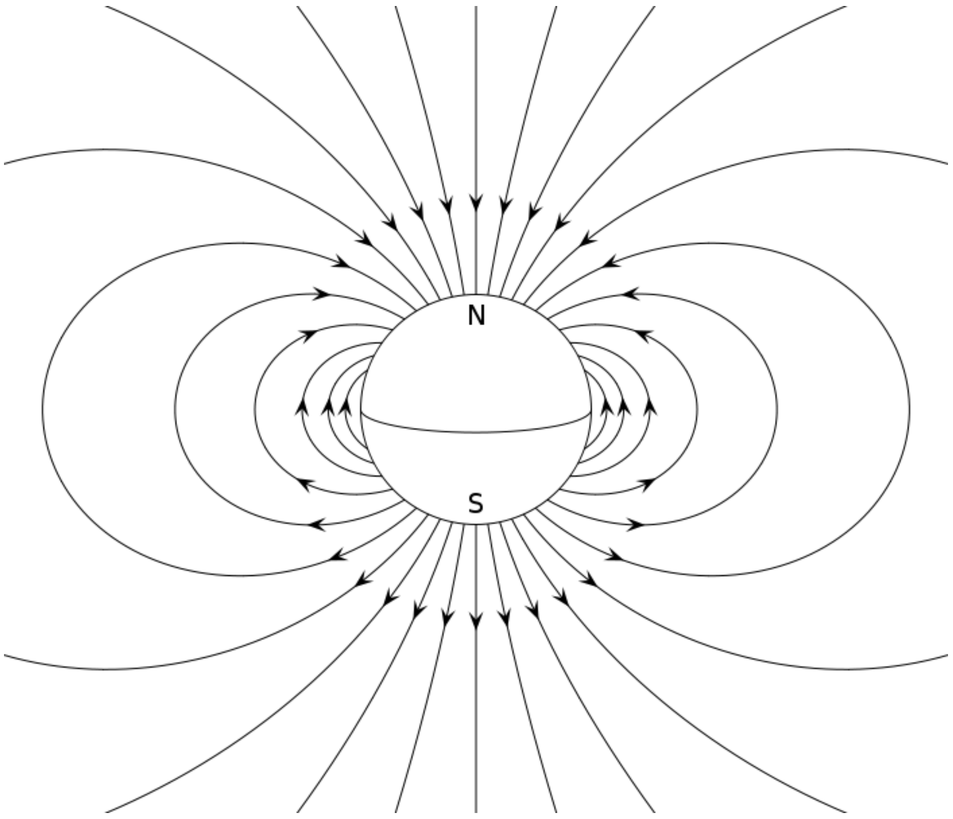
For the comparison, plot computed position and the real position vectors on a plot and plot the position errors on another plot.

Chapter

6

05_Reference Magnetic Field Model

The Earth has a magnetic field, which proves to be useful for a variety of reasons. From a navigation point of view, it allows the use of a compass and a mechanism for estimating the orientation of a satellite. From a health point of view, it protects us from the barrage of charged particles from the Sun. The model of the Earth's magnetic field to first order (80-90%) can be thought of as a dipole such as the figure below:



In order to get the magnetic field vector from the model, you need time and a position vector.

PROJECT: Compute the reference magnetic field vectors for the time and position vectors in Project 04.

Algorithm:

$$B(\bar{r}) = \frac{a^3 H_0}{|\bar{r}|^3}\left[3(\hat{m}\cdot\hat{r})\hat{r} - \hat{m}\right]$$

where $\bar{r}$ is the spacecraft position vector in km
$\hat{r}$ is the spacecraft position unit vector
$\hat{m}$ is the dipole unit vector
$a^3 H_0$ is dipole moment with value `7.943e13 (mG-km^3)`

The dipole unit vector is calculated as follows:

$$\hat{m} = \begin{bmatrix} \sin(\theta)\cos(GHA+\phi) \\ \sin(\theta)\sin(GHA+\phi) \\ \cos(\theta) \end{bmatrix}$$

where θ is the coelevation of the dipole (168.6 degrees)
ϕ is the East longitude of the dipole (109.3 degrees)
GHA is the Greenwich Hour Angle

The GHA is calculated as follows:

$$\text{rot} = (\text{t-tref})\Omega;$$
$$\text{GHA=mod(rot,360)}$$

where Ω is the Earth's spin rate (0.00417807462229498 deg/sec)
tref is time when Greenwich crossed through 0 degrees and is 1282384973.0189 seconds since September 1, 1957 0 Hours

Chapter 7

06_Attitude Rate Dynamics

Eulers equation of motion describes the spacecraft angular motion in the presence of an external torque, M.

$$\mathbf{I} \cdot \dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times (\mathbf{I} \cdot \boldsymbol{\omega}) = \mathbf{M}$$

I = Moment of inertia (kg-m^2)
w = angular rate (rad/sec)
M = external torque (N-m)

Project: Propagate rate using the 4th order runge-kutta algorithm with the following initial parameters for 1200 seconds:

```
I = [ 6.400030239e-01    -1.147147624e-01   -1.679751878e-03;
     -1.147147624e-01     9.414219149e-01    3.59946831e-04;
     -1.679751878e-03     3.59946831e-04     1.1155425364e+00];%  kg-m^2

w0=[ 1.682386001340433e-02;
      1.262317498918817e-02;
     -2.832162183099951e+00]';%  rad/sec

M = [0;0;0];
```

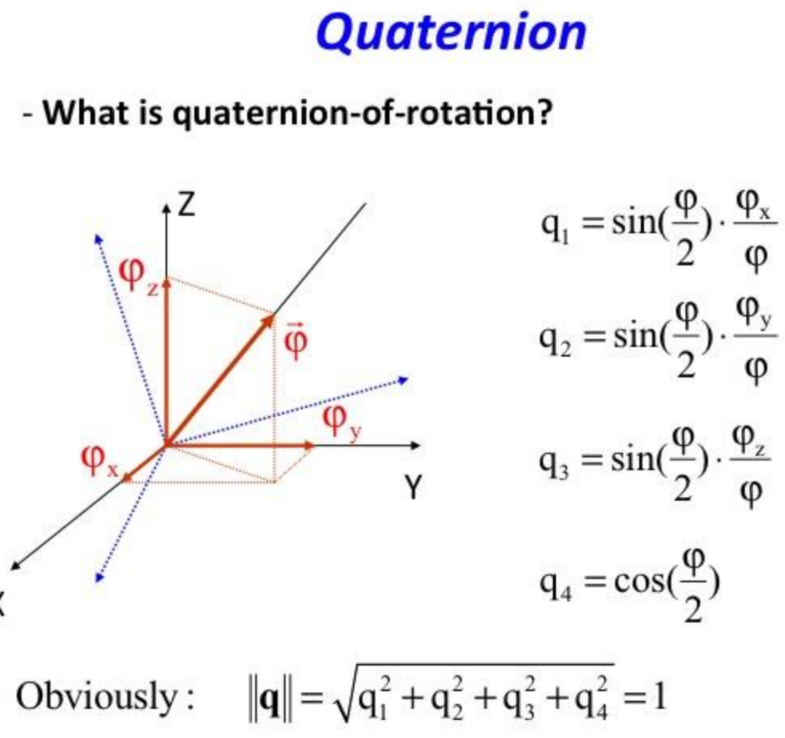
The ODE state will be w (spacecraft rate)

Chapter 8

07_Attitude Quaternion Kinematics

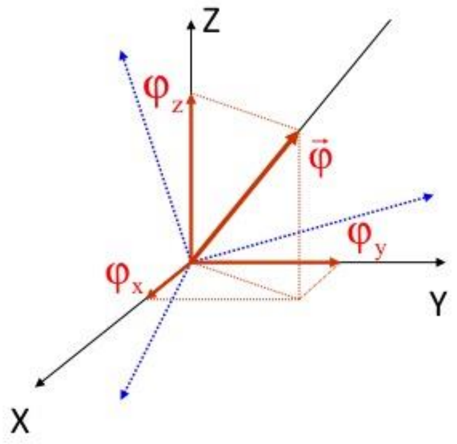
Spacecraft orientation or attitude is defined as the parameters necessary to describe the spacecraft body axes relative to a reference coordinate system. Traditionally, the reference coordinate system is Earth centered where the reference z-axis is the Earth's spin axis and the x & y axes are in the Earth's equatorial plane. As we know, the Earth rotates. For this project (and most spacecraft), the Earth's axes are assumed frozen at January 1, 2000. All spaecraft attitudes are relative to those reference axes. Most of the time, the axes are not a problem as you are given reference data relative to this coordinate system which is called GeoCentric Inertial (GCI) Mean 2000.

As mentioned above, spacecraft attitude takes many forms. Tradionally with aircraft, attitude has been defined as roll, pitch, and yaw where roll is a rotation about the x-axis, pitch is a rotation about the y-axis, and yaw is a rotation about the z-axis. This form of attitude is easier to visualize. However, it involves trigametric functions, is dependent on the order of the rotation (roll/pitch/yaw,yaw/pitch/roll, etc) and has singularities for certain angles. Due to the these issues and the complexities of implementing trigametric functions in early computers, quaternions were selected to represent spacecraft attitude. Quaternion arithmetic is algebraic and more efficient for computation. A quaternion is defined as follows:



Quaternion

- What is quaternion-of-rotation?



$$q_1 = \sin(\frac{\varphi}{2}) \cdot \frac{\varphi_x}{\varphi}$$

$$q_2 = \sin(\frac{\varphi}{2}) \cdot \frac{\varphi_y}{\varphi}$$

$$q_3 = \sin(\frac{\varphi}{2}) \cdot \frac{\varphi_z}{\varphi}$$

$$q_4 = \cos(\frac{\varphi}{2})$$

$$\text{Obviously}: \quad \|\mathbf{q}\| = \sqrt{q_1^2 + q_2^2 + q_3^2 + q_4^2} = 1$$

1

where $\vec{j}$ is an angular vector fixed to your reference coordinate system (i.e. GCI Mean 2000). If you rotate about that vector by angle j, the GCI axes will line up with the spacecraft body axes.

Attitude Motion is as follows:

$$\dot{\bar{q}} = \frac{1}{2} W \bar{q}$$

where

$$W = \begin{bmatrix} 0 & w_z & -w_y & w_x \\ -w_z & 0 & w_x & w_y \\ w_y & -w_x & 0 & w_z \\ -w_x & -w_y & -w_z & 0 \end{bmatrix}$$

$\bar{w}$ is the spacecraft body rate vector

$\bar{q}$ is the spacecraft quaternion (4x1 column vector)

Project: Augment (add to) the attitude rate dynamics equation (in Project 06) using the quaternion kinematics equation above and propagate using your 4th order Runge-Kutta.

Assume the following initial quaternion and utilize the rates calculated before.

```
q0 = [-6.236273220692253e-01;
       -7.496449331864782e-01;
       -1.594698559301466e-01;
        1.539181677587992e-01];
```

Chapter

9

08_Sun Sensor Model

In order to model the sun sensor, the following information is necessary:

1. Attitude (which for us will be in the form of a quaternion)
2. Sun sensor mounting angle, g, from the spacecraft x-axis (61.5 degrees)
3. Sun sensor quantization (0.125 deg/count)
4. Sun sensor noise (0.05 degrees)

For a quaternion q=[q1;q2;q3;q4], the 3x3 attitude matrix, A, is computed as follows:

```
A(1,1)=  q1q1 - q2q2 - q3q3 + q4q4;
A(2,1)=    2*(q1q2 - q3q4);
A(3,1)=    2*(q1q3 + q2q4);
A(1,2)=    2*(q1q2 + q3q4);
A(2,2)= -q1q1 + q2q2 - q3q3 + q4q4;
A(3,2)=    2*(q2q3 - q1q4);
A(1,3)=    2*(q1q3 - q2q4);
A(2,3)=    2*(q2q3 + q1q4);
A(3,3)= -q1q1 - q2q2 + q3q3 + q4q4;
```

From the definition of an attitude:

$$\bar{S}_{body} = A \times \bar{S}_{GCI}$$

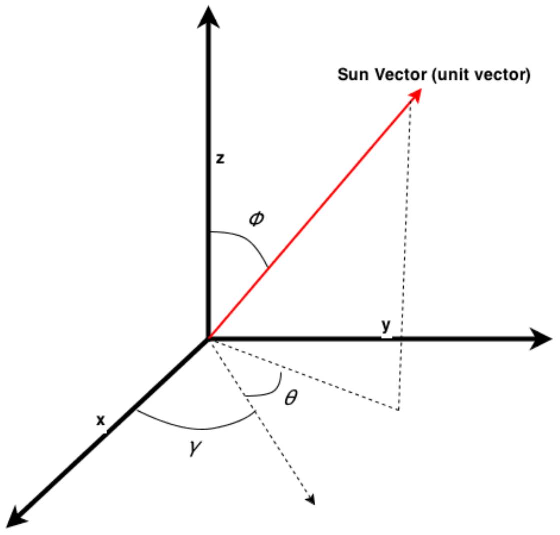


Figure 1: Sun Vector in Spacecraft Body Coordinates and Sun Sensor Angle Definitions

PROBLEM: Model a Sun Sensor as follows:

1. Using your propagated attitudes from Project 07, compute sun vectors in GCI using times in Project 07, and then compute the sun vectors in body coordinates.
2. Start Time is: `20060401.08505992969`
3. Compute the azimuth angle, q, from the mounting angle, g
4. Compute the angle from the x-y plane to the sun vector in body coordinates (sun angle), f
5. Add noise to the sun angle (HINT: use randn(1,1)*noise and make sure your units are correct)
6. Convert the sun angle to counts using the scale factor from Project 01 (ignore bias term)
7. Output the following:
   a. Time
   b. Quaternion
   c. Sun vector in gci
   d. Sun vector in body
   e. Angle in x-y plane from mounting angle
   f. Sun Angle with noise
   g. Sun Angle counts

Chapter

10

09_Magnetometer Model

In order to model the magnetometer, the following information is necessary:

1. Attitude (which for us will be in the form of a quaternion)
2. Magnetometer in GCI coordinates at attitude time
3. Magnetometer quantization (0.3 mG/count)
4. Magnetometer noise (0.1 mG)

For a quaternion $\bar{q} = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 \end{bmatrix}$, the 3x3 attitude matrix is computed as follows:

$$A(\bar{q}) = \begin{bmatrix} q_1^2 - q_2^2 - q_3^2 + q_4^4 & 2*(q_1q_2 + q_3q_4) & 2*(q_1q_3 - q_2q_4) \\ 2*(q_1q_2 - q_3q_4) & -q_1^2 + q_2^2 - q_3^2 + q_4^4 & 2*(q_2q_3 + q_1q_4) \\ 2*(q_1q_3 + q_2q_4) & 2*(q_2q_3 - q_1q_4) & -q_1^2 - q_2^2 + q_3^2 + q_4^4 \end{bmatrix}$$

From the definition of an attitude:

$$\bar{M}_B = A \times \bar{M}_R$$

PROJECT A:

1. `Start time is:  20060401.08505992969`
2. Use the following position and velocity:
   r0 = [1.939171267526330e+03;
   7.053946077269909e+03;
   5.216461164024867e+03]; % initial position in km
   v0 = [7.728790449144201e-01;
   -5.188238121076679e+00;
   3.991379730081099e+00];% initial velocity in km/sec
3. Using propagated attitudes from Project 07, the orbital dynamics routines from Project 04, the magnetic dipole model from Project 05, and the corresponding magnetic field vectors in GCI, compute the magnetometer in body coordinates

4. Add noise to each component of the magnetic field(HINT: use randn(1,1)*noise and make sure your units are correct)
5. Convert the magnetometer values to counts using the equation from Project 1 using bias of 0 and .3 mG/count).
6. Output the following:
    a. Time
    b. Quaternion
    c. Magnetic Field vector in gci
    d. Magnetometer vector in body
    e. Magnetometer in Counts
    f. Magnetometer converted from counts to mG

PROJECT B:

Magnetometers have a significant number of error sources along with the reference magnetic field model. In order to confirm that your magnetometer model is actually consistent with the reference magnetic field, compute the magnitude of the magnetometer measurements and subtract from that the magnitude of the reference magnetic field vectors. The result should have a mean close to 0 and with white noise in the neighborhood of your input noise.

Chapter 11

10_Attitude Matrix Calculation

The attitude matrix, A, can simply be calculated using the Triad algorithm as follows:

$\hat{S}_b$ is the Sun unit vector in sc body coordinates
$\hat{M}_b$ is the Magnetic Field unit vector in sc body coordinates
$\hat{S}_R$ is the Sun unit vector in GCI coordinates (from model)
$\hat{M}_R$ is the Magnetic Field unit vector in GCI coordinates (from model)

For the B and R matrices as shown below (all vectors are 3x1 column vectors (i.e. 3 rows and 1 column))

$$B = \begin{bmatrix} \hat{S}_B & \hat{M}_B & \hat{S}_B \otimes \hat{M}_B \end{bmatrix}$$

$$R = \begin{bmatrix} \hat{S}_R & \hat{M}_R & \hat{S}_R \otimes \hat{M}_R \end{bmatrix}$$

$$B = A \times R$$

where A is the attitude matrix.

The attitude matrix can be calculated as follows:

$$A = B \times R^{-1}$$

PROJECT A: Perform a sanity check on sun sensor and magnetometer measurement by computing the angle between the observed sun and magnetometer vectors and comparing it to the angle between the reference sun and magnetic field vectors. The difference should be close to 0.

For large angles (>= 2.6 degrees)

$$q_{obs} = \cos^{-1}(\hat{S}_B \cdot \hat{M}_B)$$
$$q_{ref} = \cos^{-1}(\hat{S}_R \cdot \hat{M}_R)$$

For small angles (< 2.6 degrees)

$$q_{obs} = \sin^{-1}(|\hat{S}_B \text{Ä} \hat{M}_B|)$$
$$q_{ref} = \sin^{-1}(|\hat{S}_R \text{Ä} \hat{M}_R|)$$

PROJECT B: Develop a function to compute attitude matrices from sun sensor, magnetometer, reference sun, and reference magnetic field vectors. Check the attitude matrix as follows:

$$\mathrm{D}q_S = \hat{S}_B - A \cdot \hat{S}_R$$
$$\mathrm{D}q_M = \hat{M}_B - A \cdot \hat{M}_R$$

where $\mathrm{D}q_S$ is the sun sensor measurement residual and $\mathrm{D}q_M$ is the magnetometer measurement residual. In the presence of perfect measurements (no noise or systematic errors), the measurement residuals are 0. In the presence of white noise, the measurement residuals have a mean of 0 and a standard deviation in the ballpark of the sensor noise

Chapter

12

11_Attitude Comparison

The .mat file named “intern_project_st5_155.mat” contains the actual ST-5 (155) ground estimated attitude quaternions and rates. The formats of the two arrays are as follows:

q_truth (time(YYMMDD.HHMMSSmmm) q1 q2 q3 q4 flag(0=good))
rate_truth (time(YYMMDD.HHMMSSmmm) wx wy wz (rad/sec) flag(0=good))
sc_position_truth (time(YYMMDD.HHMMSSmmm) rx ry rz (km) vx vy vz (km/sec))

1. Start simulation with the following:

t0, q0, w0, r0, v0 from .mat file

2. Run simulation for 600 seconds from t0 with time increment of 0.05 seconds

3. Compare estimated quaternion calculated from measurements and reference vectors to truth_quaternions as follows:

interp_sim_quaternions=interp1(sim_times,sim_quaternions,truth_quat_times,’spline’);

then normalize interp_sim_quaternions.

4. For propagated quaternions and the truth quaternions, do compute the spin axis vector in GCI coordinates as follows and compare the two:

    a. Compute attitude matrix from quaternion, A
    b. Compute spin axis in GCI as follows: $A^{-1}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$
    c. Compute the angle between the true spin axis and the simulated spin axis
        a. Plot the angle
        b. Compute the mean of the angle
        c. Compute the standard deviation of the angle

5. Compare simulated rate to rate_truth(:,2:4). Interpolate the simulated rates to the times of the true rates using the following:

interp_sim_rates=interp1(sim_times,sim_rates,truth_rate_times,’spline’);

    a. Plot the difference of the two rates (skipping the first 400 seconds of data)
    b. Calculate the mean difference of each axis
    c. Calculate the standard deviation of the difference of each axis.

6. Compare propagated position and velocity to the sc_position_truth(:,2:7)

7. Document why there may be differences.

Chapter 13

Conclusion!

I just want to say thank you very much for taking the time to read this book, I have spent a lot of time working on this project, and even though I truly enjoy it, it is still nice to see others take notice of my work!