Instituto de Sistemas e Robótica

Pólo de Lisboa

Stabilisation and Control of Small Satellites

Small Satellite Attitude Closed Loop Control using a Kalman Filter Estimator

RT-403-99

Robert Clements

Supervisor: Prof. Pedro Lima

Maio de 1999

ISR – Torre Norte

Av. Rovisco Pais

1096 Lisboa CODEX

PORTUGAL

This document is the Final Year Project Report of Robert Clements, a student of Aeronautical Engineering at Imperial College, London, UK. The work has been done with the support of Imperial College.

Abstract

This report describes the enhancement of satellite sensor simulations within an existing satellite simulator and the development of an attitude estimator algorithm. Sensor models have been constructed for magnetorquers, Sun sensors and Earth horizon sensors. An estimation algorithm using an Extended Kalman Filter to estimate quaternions and angular velocities from the available sensors has been written. Methods of tuning the Kalman filter, including a maunal approach, an automatic method and a genetic algorithm method, have been implemented and tested. The estimator algorithm has been tested with two control algorithms, a predictive controller and an energy controller, and results are compared to bench mark controllers.

Key Words: Attitude Dynamics Simulation; Sun Sensor; Earth Horizon Sensors; Kalman Filtering; Attitude Estimation; Attitude Control; Genetic Algorithms.

Contents

1. Notation

1.1. Co-ordinate Systems

Three Co-ordinate Systems (CS) are used within this report. These are described below:

Satellite Co-ordinate System (SCS) - This CS is a right orthogonal CS with its origin at the centre of gravity. The z axis is parallel to the boom direction and points toward the boom tip. The x axis is perpendicular to the shortest edge of the satellite's base, and points away from the boom canister. The y axis is perpendicular to the longest edge of the satellite's base. It is the reference CS for attitude measurements and the magnetorquers.

Orbital Co-ordinate System (OCS) - This CS is a right orthogonal CS fixed at the centre of mass of the satellite. The z axis points at zenith (is aligned with the Earth's centre and points away from Earth), the x axis points in the orbit plane normal direction and its sense coincides with the sense of the orbital angular velocity vector. The Orbit CS is the reference for the attitude control system.

Inertial Co-ordinate System (ICS) - This CS is an inertial right orthogonal CS with its origin at the Earth's centre of mass. The z axis is parallel to the Earth's rotation axis and points toward the North Pole. The x axis is parallel to the line connecting the centre of the Earth with Vernal Equinox and points towards Vernal Equinox (Vernal Equinox is the point where ecliptic crosses the Earth equator going from South to North on the first day of spring).

1.2. Vectors and Matrices

${}^{s}v,\ {}^{o}v,\ {}^{I}v$	vector **v** resolved in Satellite CS, Orbit CS or Inertial CS respectively,
v_o^x, v_o^y, v_o^z	x, y, and z components of vector $\mathbf{v}_o$.
$\hat{v}$	predicted value of vector v.
$\dot{v}$	time derivative of vector v.
$\lvert v \rvert$	magnitude of vector v.

1.3. List of Symbols

$\boldsymbol{\Omega_{si}}$	angular velocity of Satellite CS w.r.t. inertial CS,
$\boldsymbol{\Omega_{so}}$	angular velocity of Satellite CS w.r.t. Orbit CS,
$\mathbf{q}$	attitude quaternion representing rotation of Satellite CS w.r.t. Orbit CS,
q_4	fourth component of vector q,

$\mathbf{A}(\mathbf{q})$	transformation matrix from Orbital CS to Satellite CS (direct cosine matrix),
$\mathbf{i}_o, \mathbf{j}_o, \mathbf{k}_o$	unit vector along x-, y-, z-axis of Orbit CS,
ω_o	orbital rate,
T	period of orbit,
I	inertia matrix of the satellite,
I_x, I_y, I_z	moments of inertia about x-, y-, z-principal axis,
$\mathbf{n}_{ctrl}$	control torque,
$\mathbf{n}_{gg}$	gravity gradient torque,
m	magnetic moment generated by set of coils,
b	magnetic field of Earth (geomagnetic field),
s	vector from the satellite to the Sun
ψ, θ, φ	Roll, Pitch an Yaw angles respectively,
γ	angle between the satellite's boom and the local vertical,
Δt	time step,
ϕ	Angle of Radius of Earth when view from the satellite,
R_Earth	Radius of the Earth.
$I_{(i\times i)}$	i by i Identity Matrix

2. Introduction

The report forms part of a ongoing project called ConSat. Project ConSat aims to study the dynamic of bodies under the influence of gravitational, aerodynamic and control moments in the particular case of small satellites. The project started in 1997, the first year of the project was dedicated to the study of the satellite's dynamics and the development of the satellite simulator. The second year of the project was dedicated to the development, implementation and testing of multiple attitude control algorithms. These algorithms assumed that the current attitude and current motion of the satellite was known. The purpose of this report is to develop an attitude determination strategy and to test this strategy with the control algorithms that have been developed. This was done in three parts; 1) the development of simulations of the satellite's sensors; 2) the development of an attitude estimator algorithm; 3) the intergration of the attitude controller and estimator in the closed loop. The complete system is shown in Figure 2.1. The following section gives a brief description of the satellite simulator that already exists. A detailed description of these is given in references [Tabuada-98], [Tavares-98a] and [Tavares-98b].

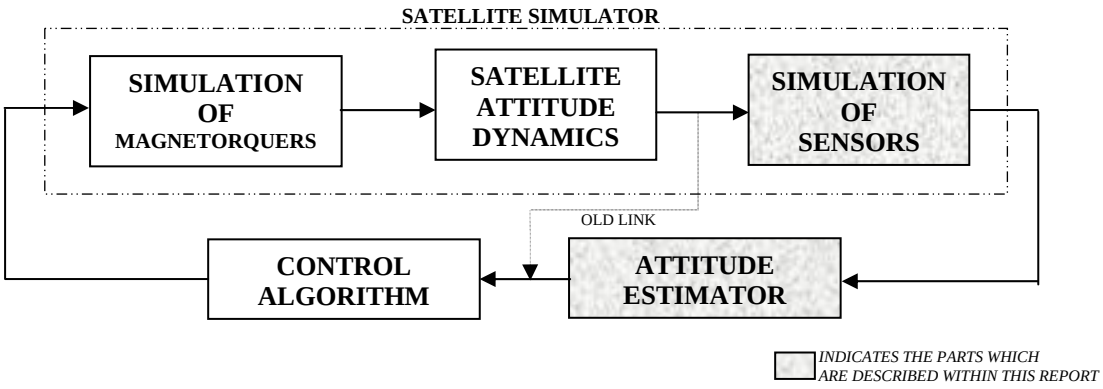

Figure 2.1 - System Architecture

This system uses a specific satellite called PoSat-1 as a case study. PosSat-1 is a 50kg micro-satellite launched in 1993 as a technology demonstration. Throughout this work it is assumed that the system is being developed for this satellite, however, the ideas and method can be easily adapted to other satellite configurations.

2.1. ConSat Simulator

The simulator calculates the attitude of the satellite as it travels around a specific orbit. The orbits are generated prior to the attitude simulation. The orbit used within the simulator is that of PoSat-1 which is a polar orbit with slight eccentricity at an altitude of approximately 800km., however, this can be changed by varying the input parameters. The orbit can start from any date and time specific by the user. The orbit can then be generated prior to running the simulation using the USSPACECOM SGP4 mathematical model. The

geomagnetic field throughout the orbit is also pre-calculated using a 10^{TH} order spherical harmonic IGRF model.

The satellite dynamics are modelled using Euler's equations for rigid body motion under the influence of external moments. The only moments considered are those from the magnetorquers and those due to gravity gradient. Within the simulator the attitude is always expressed in quaternions (also known as Euler symmetric parameters). As mentioned above the simulator uses magnetorquers to control attitude, similar, to those used on PoSat-1 and on most small satellites. The magnetorquers are restricted to only be able to fire for a duration of one, two or three seconds. The coils can only be fired in sequence once every ten seconds, in the first three seconds the x-axis coil can be fired, in the second three seconds the y-axis coil and in the last three the z-axis coil (note, on the satellite this order can be changed, this is currently not possible within the simulation). The simulations can be starting from any initial attitude and angular velocity specified by the user.

The simulator is controller by a user-interface which can display animations of the satellites motion and plots of all important parameters through out each orbit. Figure 2.1.1 shows the graphical interface.

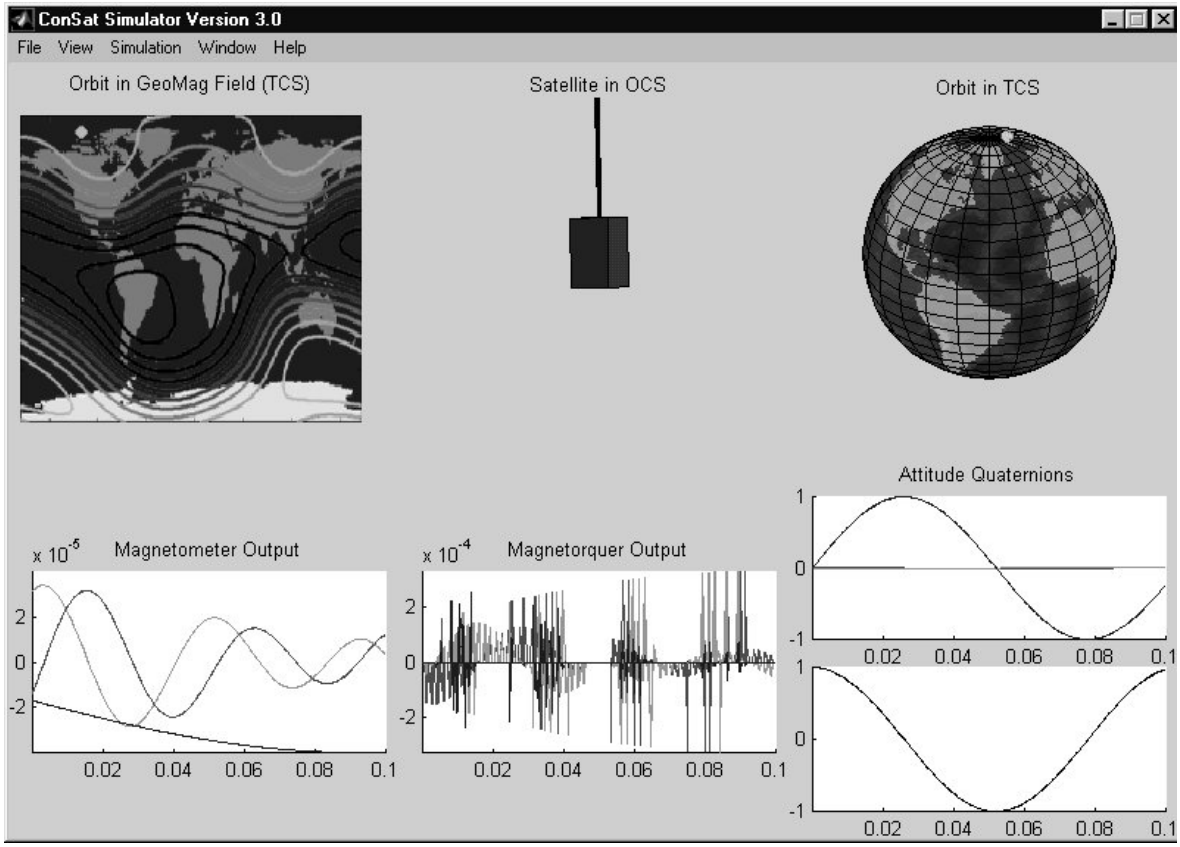

Figure 2.1.1 - ConSat Simulation Graphical Interface

3. Satellite Sensors

3.1. Introduction

To enable an attitude estimator to be tested the simulator needs to accurately model the sensors available on the satellite. The three key sensors used for attitude determination in this work are Magnetometers, Sun Sensors, and Earth Horizon Sensors. The purpose of this section is to develop simulations of each of these sensors to produce realistic sensor information. Telemetry data from PoSat-1 will be used to check the simulations are realistic.

3.2. Magnetometer

The magnetometer is a simple sensor to model because within the simulator the magnetic field is pre-calculated in the orbital co-ordinate system. Therefore, to determine the output of the sensors the magnetic field vector in OCS can be rotated to SCS using the directional cosine matrix

3.3. Sun Sensors

The purpose of this part of the simulator is to accurately model the output of the Sun sensors used on PoSat-1. PoSat has two Sun sensors with each sensor having two channels. Both sensor's field of view lie in the x-y plane, one looking in the positive y direction, the other in the negative x direction (in SCS). The exact design of the sensors is unknown so to produce a simulation the best guess of the design had to be made.

A number of common designs are given in [Wertz], considering all the information available about the sensors on PoSat-1 and the telemetry from them, the most likely layout is that shown in Figure 3.3.1. Each sensor consists of two light sensitive cells, therefore producing the two channel output. Each cell is angled to the horizontal as shown in the figure. To implement this design it is necessary to determine the geometry of the sensor (i.e. angles α and β). A number of clues can be found in the satellite technical documentation and in the telemetry data.

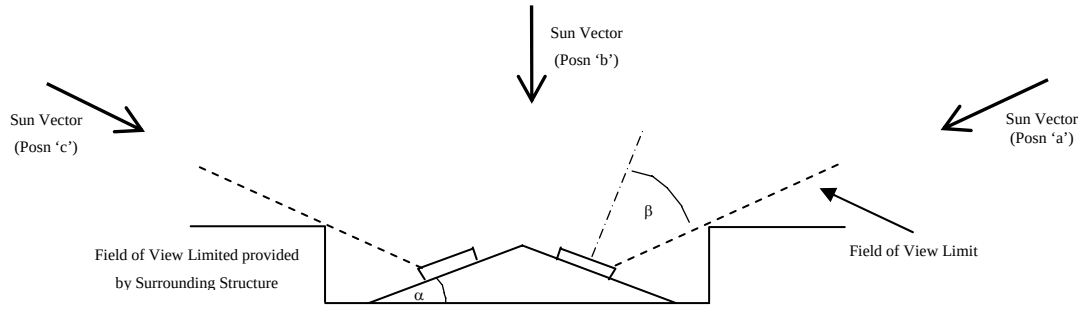

Figure 3.3.1 - New Sun Sensor Design

The technical documentation states that each cell has a total field of view of 120°. It is clear from the telemetry that both channels of the sensor start reading at the same time. Thus, when light first hits one cell it must also hit the other cell. It is also clear from the telemetry that one channel starts reading from zero where as the other starts reading at a positive value. These two facts are shown in the piece of telemetry in Figure 3.3.4 and Figure 3.3.5. The geometry that allows for all these facts is if the angle of inclination of the sensors, $\alpha = 30^o$ and the FOV limits of each cell, $\beta = 30^o$ from the cell's normal axis. If we imagine the Sun passing round the satellite, as the satellite rotates, the sensor will read zero until the Sun reached position 'a' when the Sun first passes into the sensor field of view. At this point channel one will jump to a value just less than maximum and channel two will just start to give a reading. Both readings will increase until the Sun is normal to channel one when it's reading will be a maximum. Once the Sun reaches point 'b' both sensors will give the same reading. The effects are then mirrored as the Sun passes to point 'c'. If the angle 'α' was larger the field of view limits would have to be decreased to maintain the 'both sensors start reading at the same time' condition, then the overall field of view of the each sensor would be reduce below the 120° requirement. If the angle 'α' was reduce the reverse would occur.

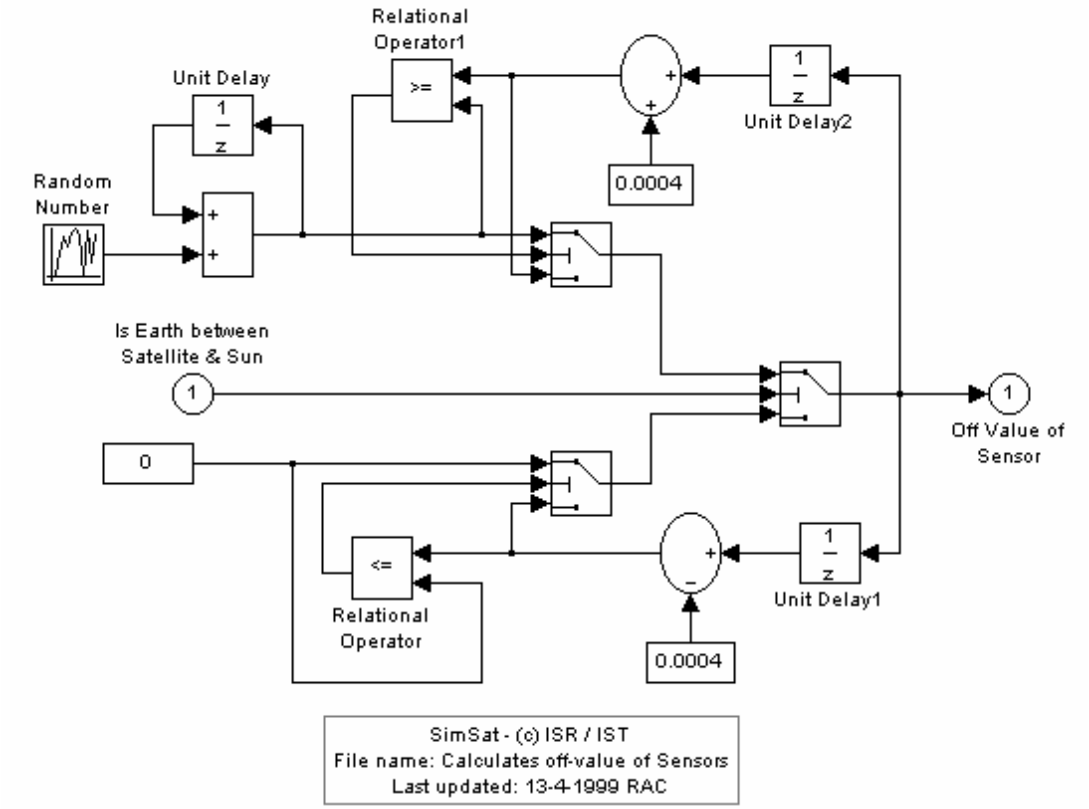

Figure 3.3.2 - Sun Sensor Off-Value Calculation

This analysis assumes that we are in two dimensions, in reality the Sun could be at an angle of elevation to this plane. This would cause the maximum value of sensor to reduce, this is seen in the telemetry (Figure 3.3.4). In Figure 3.3.1 the sensor is shown in a recess, the sides of which cause provide the field of view limits. In three dimensions this recess will actually be a box so there will be upper and lower field of view

limits as well. It is not possible, from the available data, to determine the magnitude of this limit so it has been assumed to be $\pm 60^{o}$. The total field of view of each cell when projected out into space is a rectangle.
The simulator must also consider the possibility of the Earth being between the satellite and the Sun. This case is simply detected by calculating if the angle between the vector to the centre of the Earth and the vector to the Sun is less than the angle of the radius of the Earth as seen from the satellite. The angle of the radius of the Earth is pre-calculated within the simulator from the orbit altitude, so can be considered known at all time.

It is clear from the telemetry data (Figure 3.3.4) that the value of output from the sensors when they can't see the Sun varies. Initial thought might suggest that the 'off' value of the sensor should always be zero. However, it was realised that this variation was due to the sensor picking up Sun light reflected from the Earth. So, if the satellite is over the dark side of the Earth the off value will be zero, if it is over the lit side it will have an slightly increased value. The magnitude of this increase will depend on factors like weather conditions, current altitude and whether the satellite is over land or sea. The 'off' value over the lit side of the Earth has thus been modelled using a random noise signal which gives values similar to those seen in the telemetry. The model to calculate this off-value is shown in Figure 3.3.2, the random noise has a normal distribution with a mean of 0.0 and a standard deviation of $1.0e^{-5}$.

The geometry of the complete Sun Sensor is implemented in Simulink, as shown in Figure 3.3.3.

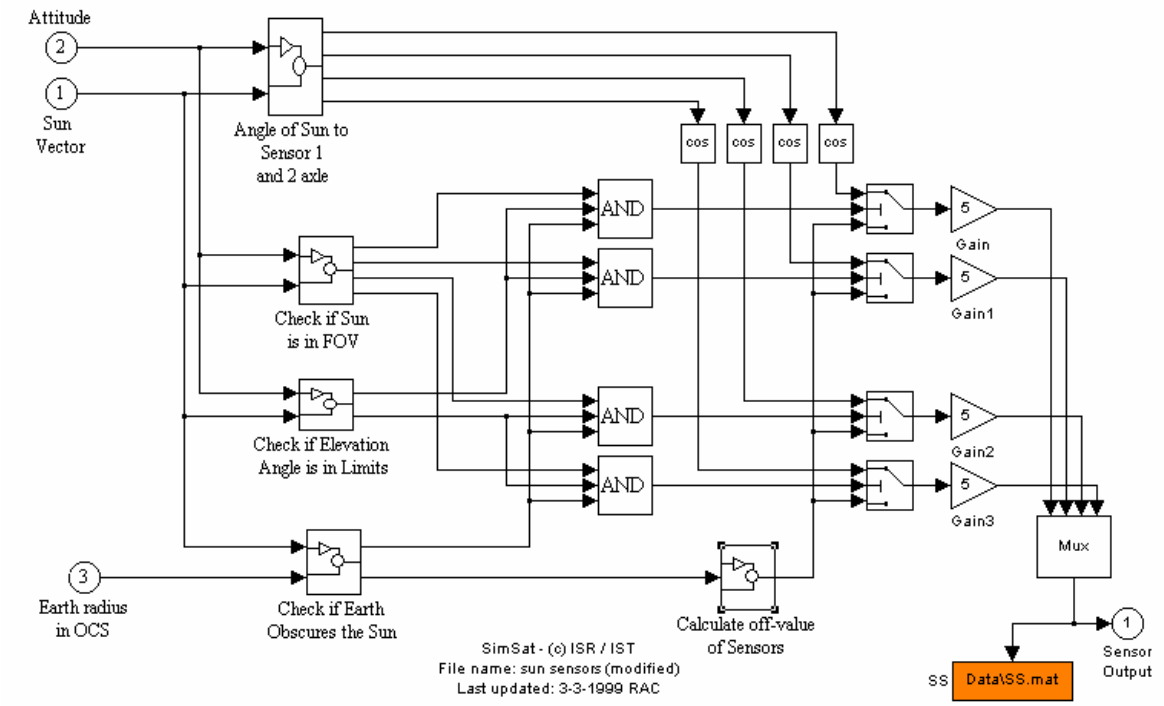

Figure 3.3.3 - Sun Sensor Simulink Block

Comparison of the output from the simulator with telemetry data from PoSat-1 can be seen in Figure 3.3.4 and Figure 3.3.5. Figure 3.3.4 shows a comparison for one orbits (showing just one channel). It can be seen that the two are very similar. The differences in magnitude of each peak is due to differences in attitude of the satellite in the simulation and in reality. The attitude of PoSat-1 is not known exactly so it is not possible to get the simulator to have exactly the same attitude. The difference in number and width of the peaks is due to different spin rates of the simulation and real satellite, again, the exact spin rate of PoSat is not known. The modelling of the off-value can also be clearly seen in this figure. Note, the flattening of the high peaks in this figure is due to saturatiuon of the ADC within the satellite. Figure 3.3.5 shows the comparison of one revolution of the satellite (showing two channels). It can be see that the cut-off of the signals as the Sun passes out of the field of view of each channel are very similar. It is also clear to see that both channels start giving a reading at the same point in time. The spike that appears just before each peak is due to interference on the telemetry communication link and is thus not simulated.

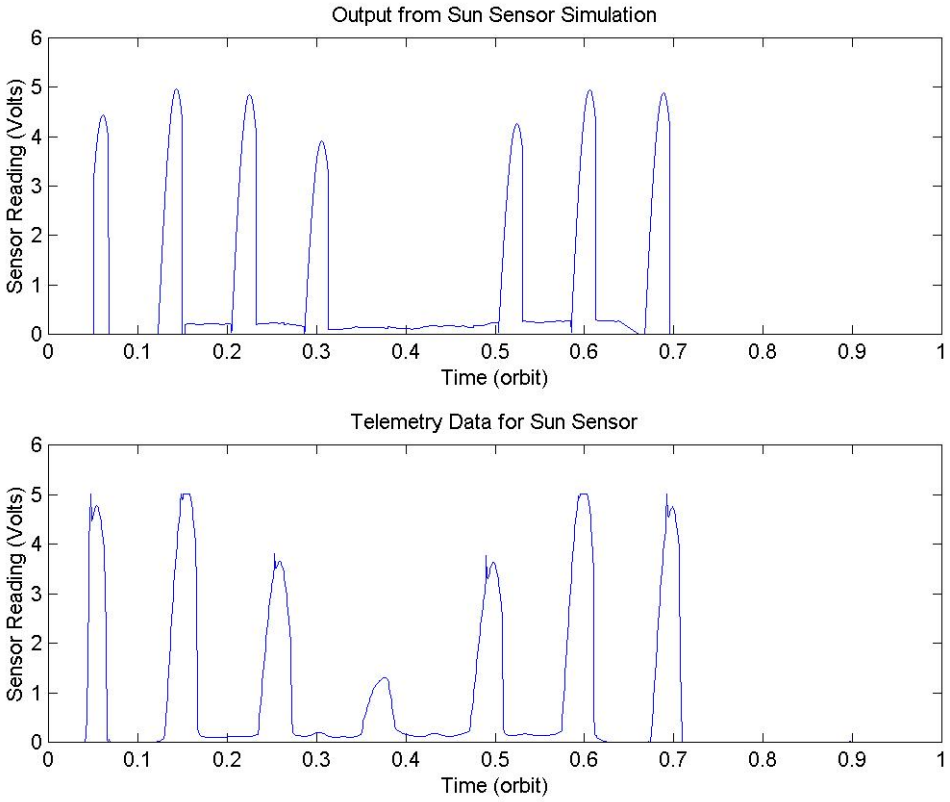

Figure 3.3.4 - Comparison of Sun Sensor Telemetry and Simulation for One Orbit

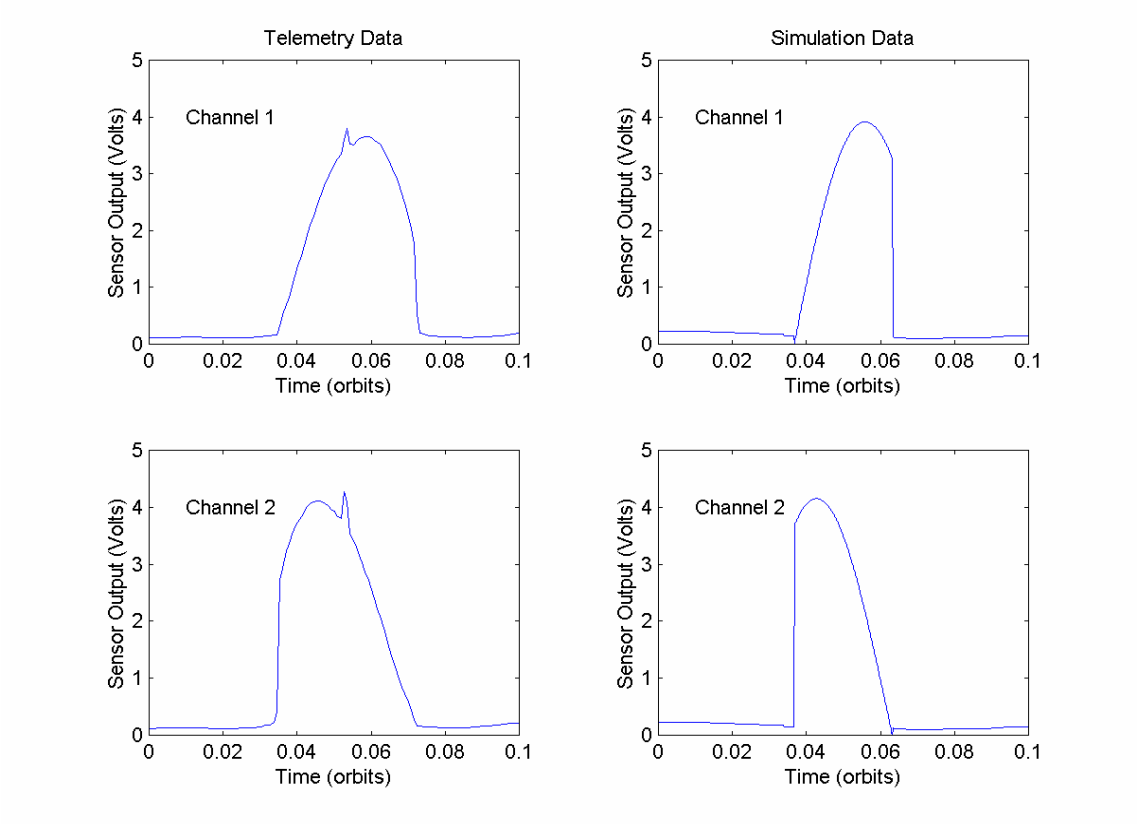

Figure 3.3.5 - Comparison of Sun Sensor Telemetry and Simulation for One Revolution

3.4. Earth Horizon Sensors

PoSat-1 has two Earth Horizon Sensors (EHS), one facing the positive y direction, the other the negative x direction (in SCS). Each EHS is a camera which looks down at the nominal Earth horizon. The view of the camera is a narrow slot, the output of the sensor is related to the amount of that slot that is lit, i.e., the sensor would read 1 if the slot was looking totally at the Earth, 0 if it was looking into space and somewhere in between if the horizon line falls somewhere across that slot. This simulation assumes that the camera focuses on the Earth surface and not any part of the atmosphere.

The main difficulty with an EHS simulation is to determine, given the relative positions of the Sun, Earth and satellite, the current view of the Earth from the satellite (this is like viewing the moon from the Earth). The Earth may appear fully lit, total dark or somewhere in between. Once the view of the Earth is determined the slot that the EHS camera looks through can be projected onto this view and the amount of that slot which is seeing light can be determined.

The calculation of the EHS output is performed in the following manner:

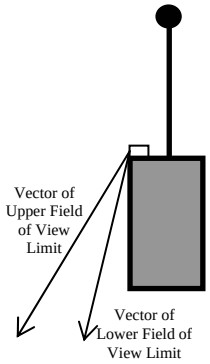

Figure 3.4.2 - Upper & Lower Field of View Limits

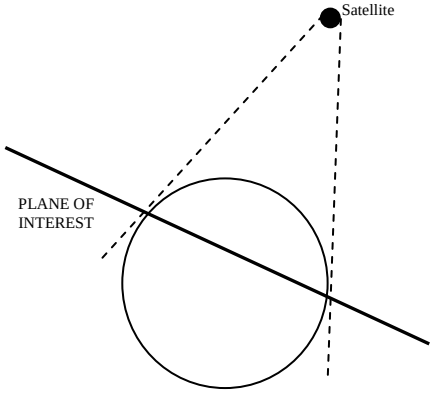

Figure 3.4.1 - Showing Plane of Interest

1) Calculate the vectors of the upper and lower limits of the sensor field of view in the orbit co-ordinate system, as shown in figure 3.4.2. The calculation is given in (3.4.1), where A(q) is the direction cosine matrix (attitude), α is the look down angle of the sensor (α=32^o at 800km altitude), β is field of view limits (β=6^o for PoSat-1).

$$ {}^{O}V_{Upper\_Limit} = \mathbf{A}(\mathbf{q}) \times \begin{bmatrix} -\cos(\alpha-\beta) \\ 0 \\ -\sin(\alpha-\beta) \end{bmatrix}, \qquad {}^{O}V_{Lower\_Limit} = \mathbf{A}(\mathbf{q}) \times \begin{bmatrix} -\cos(\alpha+\beta) \\ 0 \\ -\sin(\alpha+\beta) \end{bmatrix} \tag{3.4.1} $$

2) The satellites view of the Earth is effectively two dimensional, so the following calculation can be performed in one plane, this plane is shown in figure 3.4.1. It is at right-angle to the vector between the satellite and the centre of the Earth and is located at the point of maximum field of view of the Earth from the satellite. The equation of this plane is given in (3.4.2) (in OCS), where a and b parameterise the plane and ALT is the satellite altitude above this plane which is given by (3.4.3) where ϕ is the angle of Earth's radius (a known value within the simulator) and R_Earth is the radius of the Earth.

$$ \begin{bmatrix} {}^{o}x \\ {}^{o}y \\ {}^{o}z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -ALT \end{bmatrix} + a \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + b \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{3.4.2} $$

$$ ALT = \frac{R\_Earth}{\tan\phi} \cos\phi \tag{3.4.3} $$

3) The equation of the Earth seen in this plane can be calculated and will be of the form $x^2+y^2=H^2$ where H is the apparent radius of the Earth and is given by (3.4.4) (note x and y are in OCS where z = -ALT).

$$ H = R\_Earth \times \cos\phi \tag{3.4.4} $$

4) The vectors describing the field of view can also be projected to points in this plane, points A and B. This is done by calculating the intersection of the vector in (3.4.1) with the plane in (3.4.2). The points A and B are thus given by (3.4.5).

$$ {}^{o}A = \frac{-R\_Earth \times \cos\phi}{\tan\phi\left({}^{o}V^{Z}_{Upper\_Limit}\right)} \times {}^{o}V_{Upper\_Limit}, \qquad {}^{o}B = \frac{-R\_Earth \times \cos\phi}{\tan\phi\left({}^{o}V^{Z}_{Lower\_Limit}\right)} \times {}^{o}V_{Lower\_Limit} \tag{3.4.5} $$

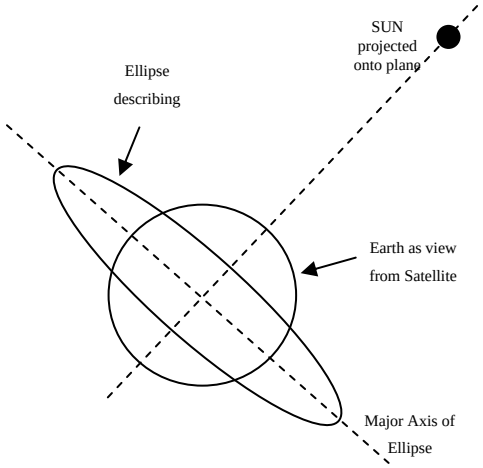

Figure 3.4.3 - Showing the Ellipse that describes the Horizon Line

5) It is now necessary to calculate the horizon line across the Earth. This can be described by part of an ellipse. If the Sun's position is projected into this plane then the major axis of this ellipse will be at right angles to the vector between the centre of the Earth and the projected Sun position, as shown in figure 3.4.3. To make the equation of the ellipse simpler the co-ordinate system can now be rotated to align with this major axis (new system u,v). The rotation from x-y space to u-v space is given by (3.4.6)

$$ R = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \tag{3.4.6} $$

where α is determined from the position of the Sun and is given by (3.4.7).

$$ \alpha = 2\pi - \cos^{-1}\left({}^{o}S^{x}\right) \tag{3.4.7} $$

where S is the vector from the satellite to the Sun.

6) The points A and B can now be determined in the new co-ordinate system using the rotation matrix in (3.4.6). (Note, everything, can now be consider into 2-D space because we are only interest in things on the plane).

7) The equation of the ellipse is thus $\left(\frac{u}{a}\right)^2 + v^2 = (R\_Earth)^2$ where R_Earth is the radius of the Earth and 'a' relates to the size of the ellipse. This value 'a' is determined by the angle of elevation of the Sun from this plane. If the elevation angle was 90^o or 270^o then a must be 1, if the elevation angle was 0^o or 180^o then a must be 0. The value of 'a' is, thus, given by (3.4.8).

$$ a = \left|{}^{O}S^{Z}\right| \tag{3.4.8} $$

If 'a' is less than 0.01 then the ellipse is assumed to be a straight line with equation v = 0

8) All the equations above can now be scaled by R_Earth to avoid rounding errors. In summary the important points are now the eqn of the Earth's apparent radius (3.4.9), the eqn if the ellipse representing the horizon line (3.4.10) and the points A and B (3.4.11)

$$u^2 + v^2 = \cos^2\phi \tag{3.4.9}$$

$$\frac{u^2}{\left|{}^{O}S^{Z}\right|} + v^2 = 1 \qquad \text{or} \qquad v = 0 \text{ if } \left|{}^{O}S^{Z}\right| \le 0.01 \tag{3.4.10}$$

$$A = R \times \left[\frac{-\cos\phi}{\tan\phi\left({}^{o}V^{Z}_{Upper\_Limit}\right)} \times {}^{o}V_{Upper\_Limit} \right]^{X,Y}$$

$$B = R \times \left[\frac{-\cos\phi}{\tan\phi\left({}^{o}V^{Z}_{Lower\_Limit}\right)} \times {}^{o}V_{Lower\_Limit} \right]^{X,Y} \tag{3.4.11}$$

9) The current picture is shown in figure 3.4.5. The Earth is now divided into three parts by the ellipse. It is necessary to determine which of these parts are lit and which are in darkness. This is depended on whether the Sun is above of below the plane. If the Sun, in figure 3.4.5, was above the plane then parts (a) and (b) would be lit, however, if the Sun was below the plane only part (a) would be lit. If the sign of ${}^{O}S^{Z}$ is negative the Sun is below the plane, if positive the Sun is above the plane.

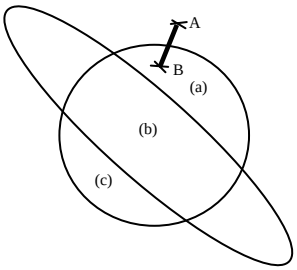

Figure 3.4.5 - Showing the Light and Dark parts of the Earth

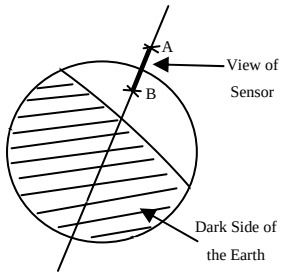

Figure 3.4.4 - Showing View of Sensor with Horizon Line

10) The equation of the line connecting points A and B can easily be found in the form v=mu+c, or u=mv+c (it is necessary to use both forms to avoid the situation when m, the gradient, becomes infinite). With this equation, the equation of the circle and ellipse intersection points can be calculated. There will be a maximum of four intersection points. An example of one possibility is shown in figure 3.4.5. Knowing the positions of A and B and each of these intersections it is relatively simple to calculate if the line between A and B is full lit, full dark, or if it crosses a horizon line. In the last case the output value can be found as the percentage of the line that is lit.

To implement the sensor facing the positive y-direction the x and y component are swapped and the sign changed in the field of view vectors in (3.4.1).

This method has been implemented at a MatLab m-file, the full program listing is given in Appendix A.

The EHS is very sensitive to attitude. Perturbations greater that $\pm 5^{o}$ will saturate the sensor. For this reason to get a good comparison between telemetry and simulation it is important that the attitude is very similar. The attitude of the satellite is unknown for the period of the telemetry data so accurate comparison is not possible. Figure 3.4.6 shows a comparison with different attitude. Although they are quite different whilst the sensor is 'on' it shows the peaks that occur as the satellite passes over the Earth horizon. i.e. as the satellite is over the horizon line, during one revolution the sensor will see a dark Earth on one half and a lit Earth on the other half of the revolution. This gives the single peak either side of the main 'on' part which is seen in both the telemetry and the simulation. It also shows the sensor switches off whilst over the dark side of the Earth.

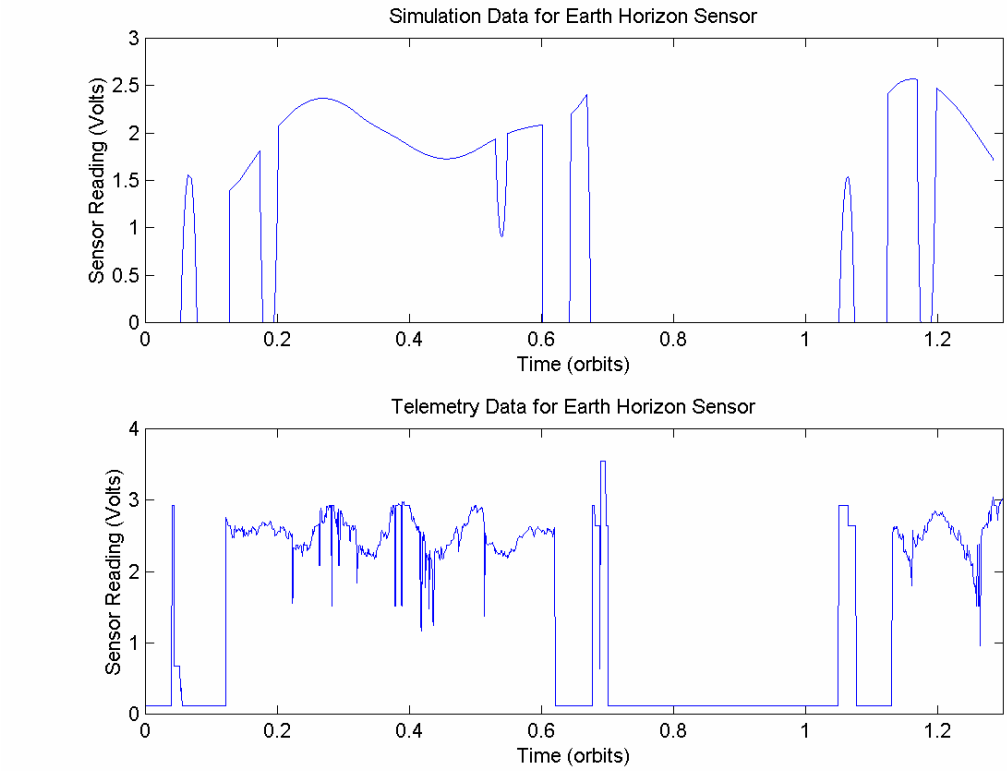

Figure 3.4.6 - Comparison of Simulation and Telemetry for one channel of the Earth Horizon Sensor

It can be seen in the telemetry that during the 'on' period there is lot of noise on the signal, this is not seen in the simulation. The cause of this is not known, however, there are a number of possibilities:

- The variation is caused by attitude variations. The variations do appear to be quite high frequency so attitude is unlikely to be the sole cause of this noise.

- Variations in the weather conditions would change the view of the Earth. If the camera is looking at a cloud it will give a low signal than if it is looking at clear sky.

- There could be some interference in the telemetry transmission (although this does not occur on the other telemetry channels) or there is some fault with the horizon sensor.

The most likely explanation is weather conditions. Figure 3.4.7 shows the simulation data for an 'artificially' created attitude (this was done by manually entering the attitude matrix, rather than feeding it from the satellite dynamics). The first plot has the satellite spinning with the boom axis offset by 0.5^{o}. The second plot shows the same simulation with small random perturbations added to the offset angle at a frequency of ones per revolution and 20 times per revolution. It can be seen that this gives very similar results to the telemetry data in figure 3.4.6. Although the variations seen in the telemetry data can be mimicked by attitude variations, it is unlikely that the satellites is actually vibrating at this higher frequency. The conclusion is, thus, that the variations are probably caused by weather or atmospheric conditions. This is not currently modelled by the simulation.

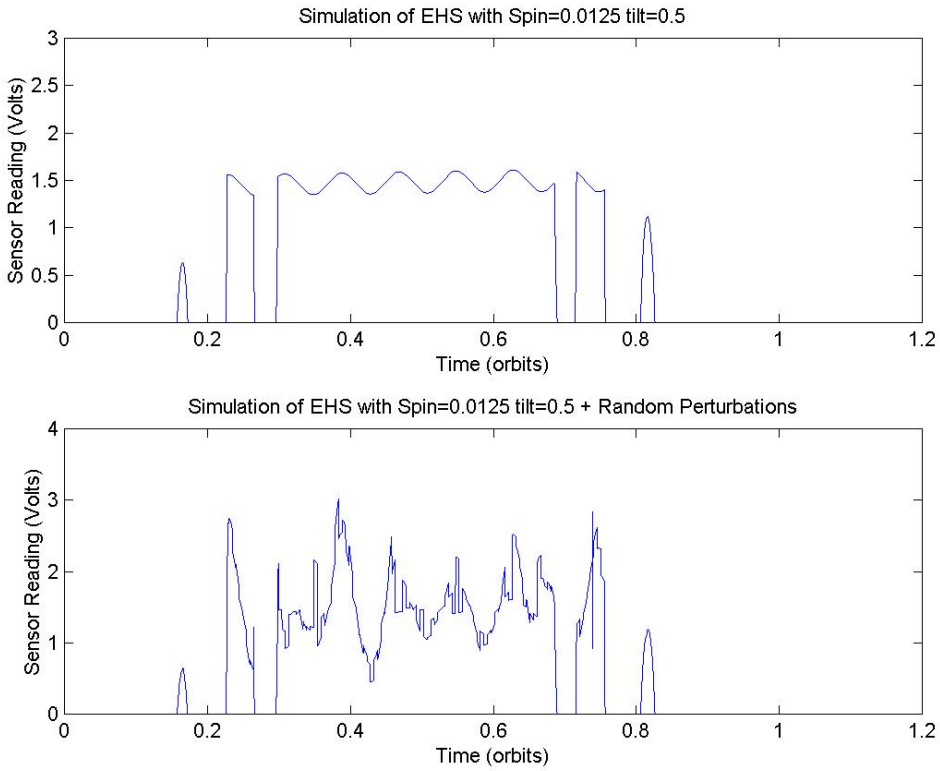

Figure 3.4.7 - Simulation of EHS with manually generated attitude

4. Attitude Estimator

4.1. Introduction

The purpose of this Attitude Estimator is to calculate the attitude quaternions and angular velocity of the satellite given the sensors readings and knowledge of the satellites dynamics. The principle difficulty with this is that some of the sensors are not available all of the time, and there reliability depends of the attitude of the satellite. This work focuses on the use of a Kalman Filter to estimate the attitude. The first version of the estimator only used the magnetometer reading, later versions also incorporated the Sun sensor.

4.2. Modifications to Simulator

The core of a Kalman Filter is a model of the satellite attitude dynamics. The simulation also simulates the motion of the satellite using a model of the satellite attitude dynamics. If the estimator and the simulator were to use exactly the same model, the result would be an extremely actuate estimator, however this would be slightly unrealistic. For this reason it in necessary to alter the simulation model to include some 'realistic' uncertainty. This was done augmenting the satellite inertia matrix. The real satellite's inertia will be slightly different from the nominal values, therefore, the inertia matrix used within the simulation is given by (4.2.1). The matrix used within the estimator and controllers is given by eqn (4.2.2). The ± terms in eqn (4.2.1) are random numbers within the range given. These random numbers can be regenerated before each simulation.

$$I = \begin{bmatrix} I_1 \pm 0.05 & \pm 0.0005 & \pm 0.0005 \\ \pm 0.0005 & I_2 \pm 0.05 & \pm 0.0005 \\ \pm 0.0005 & \pm 0.0005 & I_3 \pm 0.0005 \end{bmatrix} \tag{4.2.1}$$

$$I = \begin{bmatrix} I_1 & 0 & 0 \\ 0 & I_2 & 0 \\ 0 & 0 & I_3 \end{bmatrix} \tag{4.2.2}$$

The estimator uses a model of the Earth's geomagnetic field (in OCS) to determine the error in attitude from the sensor reading. From similar reasons to those explained above it is also important that the GMF model used by the simulator is not the same as the one used by the estimator. Therefore the estimator uses a 4^{TH} Order Geomagnetic Field model where as the simulator uses a 10^{TH} Order Geomagnetic Field model. Note, it would also be too computationally expensive to implement a 10^{TH} order model on the satellite's on-board computer, so a 4^{TH} order model is more realistic.

4.3. Kalman Filter

The method used here is the same as the version of the filter proposed by [Steyn], except, the disturbance model has been removed. This report does not attempt to explain or justify the method used, however, the following summary lists the equation used.

Extended Kalman Filter Algorithm

1) Calculation of Kalman Gain

$$K_{k+1} = P_{k+1/k} H_{k+1/k}^T \left[H_{k+1/k} P_{k+1/k} H_{k+1/k}^T + R \right]^{-1} \tag{4.3.1}$$

where K_{k+1} is the Kalman Gain Matrix (7x3)

$P_{k+1/k}$ is the perturbation covariance matrix (7x7) at time k+1 given the measurements at time k.

R is the measurement error covariance matrix (3x3)

$H_{k+1/k}$ is a (3x7) matrix defined as follows:

$$H_{k+1/k} = \begin{bmatrix} 0_{3x3} & h_1 & h_2 & h_3 & h_4 \end{bmatrix} \tag{4.3.2}$$

$$h_1 = 2\begin{bmatrix} \hat{q}_{1,k} & \hat{q}_{2,k} & \hat{q}_{3,k} \\ \hat{q}_{2,k} & -\hat{q}_{1,k} & \hat{q}_{4,k} \\ \hat{q}_{3,k} & -\hat{q}_{4,k} & -\hat{q}_{1,k} \end{bmatrix} b_{orb,k+1} \qquad h_3 = 2\begin{bmatrix} -\hat{q}_{3,k} & \hat{q}_{4,k} & \hat{q}_{1,k} \\ -\hat{q}_{4,k} & -\hat{q}_{3,k} & \hat{q}_{2,k} \\ \hat{q}_{1,k} & \hat{q}_{2,k} & \hat{q}_{3,k} \end{bmatrix} b_{orb,k+1}$$

$$h_2 = 2\begin{bmatrix} -\hat{q}_{2,k} & \hat{q}_{1,k} & -\hat{q}_{4,k} \\ \hat{q}_{1,k} & \hat{q}_{2,k} & \hat{q}_{3,k} \\ \hat{q}_{4,k} & -\hat{q}_{3,k} & -\hat{q}_{2,k} \end{bmatrix} b_{orb,k+1} \qquad h_4 = 2\begin{bmatrix} \hat{q}_{4,k} & \hat{q}_{3,k} & -\hat{q}_{2,k} \\ -\hat{q}_{3,k} & \hat{q}_{4,k} & \hat{q}_{1,k} \\ \hat{q}_{2,k} & -\hat{q}_{1,k} & \hat{q}_{4,k} \end{bmatrix} b_{orb,k+1}$$

$\hat{q}_{i,k}$ is the iTH estimated component of quaternion q at time k

$b_{orb,k+1}$ is the model magnetic field vector (in OCS) at time k+1.

2) Update the State

$$d\hat{z}_{k+1/k+1} = K_{k+1}(b_{meas,k+1} - D \times b_{orb,k+1}) \tag{4.3.3}$$

$$\hat{z}_{k+1/k+1} = \hat{z}_{k+1/k} + d\hat{z}_{k+1/k+1} \tag{4.3.4}$$

where $\hat{z}_{k+1/k}$ is the estimated state vector at time k+1 given the measurements at k,

(note z is used as the state vector rather than x because x is pre-defined within Simulink S-functions)

$$z = \begin{bmatrix} \boldsymbol{\Omega}_{si}^x & \boldsymbol{\Omega}_{si}^y & \boldsymbol{\Omega}_{si}^z & q_1 & q_2 & q_3 & q_4 \end{bmatrix}^T$$

$b_{meas,k+1}$ is the measurement magnetic field vector (in SCS) at time k+1.

$$D = \begin{bmatrix} \hat{q}_{1,k}^2 - \hat{q}_{2,k}^2 - \hat{q}_{3,k}^2 + \hat{q}_{4,k}^2 & 2(\hat{q}_{1,k}\hat{q}_{2,k} + \hat{q}_{3,k}\hat{q}_{4,k}) & 2(\hat{q}_{1,k}\hat{q}_{3,k} - \hat{q}_{2,k}\hat{q}_{4,k}) \\ 2(\hat{q}_{1,k}\hat{q}_{2,k} - \hat{q}_{3,k}\hat{q}_{4,k}) & -\hat{q}_{1,k}^2 + \hat{q}_{2,k}^2 - \hat{q}_{3,k}^2 + \hat{q}_{4,k}^2 & 2(\hat{q}_{2,k}\hat{q}_{3,k} + \hat{q}_{1,k}\hat{q}_{4,k}) \\ 2(\hat{q}_{1,k}\hat{q}_{3,k} + \hat{q}_{2,k}\hat{q}_{4,k}) & 2(\hat{q}_{2,k}\hat{q}_{3,k} - \hat{q}_{1,k}\hat{q}_{4,k}) & -\hat{q}_{1,k}^2 - \hat{q}_{2,k}^2 + \hat{q}_{3,k}^2 + \hat{q}_{4,k}^2 \end{bmatrix}$$

D is the rotation matrix from OCS to SCS (Directional Cosine Matrix)

3) Update the Covariance Matrix

$$P_{k+1/k+1} = \left[I_{(7x7)} - K_{k+1}H_{k+1/k+1}\right]P_{k+1/k}\left[I_{(7x7)} - K_{k+1}H_{k+1/k+1}\right]^T + K_{k+1}R\,K_{k+1}^T \tag{4.3.5}$$

where $H_{k+1/k+1}$ is calculated from eqn 6.3.2 except that the quaternions at time k+1 are used instead of at time k.

4) Propagate State Vector

$$\hat{z}_{k+2/k+1} = \int_{k+1}^{k+2} f_{k+1}(\hat{z}_{k+1/k+1}, k+1)dt + \hat{z}_{k+1/k+1} \tag{4.3.6}$$

where $f_{k+1}(\hat{z}_{k+1/k+1}, k+1) = \begin{bmatrix} \dot{\boldsymbol{\Omega}}_{si} \\ \dot{q} \end{bmatrix}$

$\dot{\boldsymbol{\Omega}}_{si} = I^{-1}\left[n_{gg} + n_{ctrl} - \boldsymbol{\Omega}_{si} \times \left(I\boldsymbol{\Omega}_{si}\right)\right]$

$\dot{q} = \underline{\boldsymbol{\Omega}}q$

n_{ctrl} is the control moments

$$n_{gg} = 3\omega_o^2(I_x - I_z)D_{33}\begin{bmatrix} -D_{23} \\ D_{13} \\ 0 \end{bmatrix} \qquad \underline{\boldsymbol{\Omega}} = \frac{1}{2}\begin{bmatrix} 0 & \boldsymbol{\Omega}_{so}^z & -\boldsymbol{\Omega}_{so}^y & \boldsymbol{\Omega}_{so}^x \\ -\boldsymbol{\Omega}_{so}^z & 0 & \boldsymbol{\Omega}_{so}^x & \boldsymbol{\Omega}_{so}^y \\ \boldsymbol{\Omega}_{so}^y & -\boldsymbol{\Omega}_{so}^x & 0 & \boldsymbol{\Omega}_{so}^z \\ -\boldsymbol{\Omega}_{so}^x & -\boldsymbol{\Omega}_{so}^y & -\boldsymbol{\Omega}_{so}^z & 0 \end{bmatrix}$$

D_{ij} is the element of D in the i^{TH} row and j^{TH} column.

5) Propagate the Perturbation Covariance Matrix

$$P_{k+2/k+1} = \boldsymbol{\Phi}_{k+2/k+1}P_{k+1/k+1}\boldsymbol{\Phi}_{k+2/k+1}^T + Q \tag{4.3.7}$$

where Q is the system error covariance matrix

$$\boldsymbol{\Phi}_{k+2/k+1} = I_{(7x7)} + F(\hat{z}_{k+1/k+1}, k+1)\Delta t$$

$$F(\hat{z}_{k+1/k+1}, k+1)\delta z = \begin{bmatrix} \delta\dot{\boldsymbol{\Omega}}_{si} \\ \delta\dot{q} \end{bmatrix}$$

$$\delta\dot{\boldsymbol{\Omega}}_{si} = I^{-1}\left[\delta n_{gg} - \delta\boldsymbol{\Omega}_{si} \times I\boldsymbol{\Omega}_{si} - \boldsymbol{\Omega}_{si} \times I\delta\boldsymbol{\Omega}_{si}\right]$$

$$\delta n_{gg} = 6\omega_o^2(I_x - I_z)\begin{bmatrix} -\hat{D}_{33}\hat{q}_4 + \hat{D}_{23}\hat{q}_1 & -\hat{D}_{33}\hat{q}_3 + \hat{D}_{23}\hat{q}_2 & -\hat{D}_{33}\hat{q}_2 - \hat{D}_{23}\hat{q}_3 & -\hat{D}_{33}\hat{q}_1 - \hat{D}_{23}\hat{q}_4 \\ \hat{D}_{33}\hat{q}_3 - \hat{D}_{13}\hat{q}_1 & -\hat{D}_{33}\hat{q}_4 - \hat{D}_{13}\hat{q}_2 & \hat{D}_{33}\hat{q}_1 + \hat{D}_{13}\hat{q}_3 & -\hat{D}_{33}\hat{q}_2 + \hat{D}_{13}\hat{q}_4 \\ 0 & 0 & 0 & 0 \end{bmatrix}\delta q$$

$$\delta\dot{q} = \underline{\Omega_{si}}\delta q + \underline{\beta}\delta\Omega_{si}$$

$$\underline{\beta} = \frac{1}{2}\begin{bmatrix} \hat{q}_4 & -\hat{q}_3 & \hat{q}_2 \\ \hat{q}_3 & \hat{q}_4 & -\hat{q}_1 \\ -\hat{q}_2 & \hat{q}_1 & \hat{q}_4 \\ -\hat{q}_1 & -\hat{q}_2 & -\hat{q}_3 \end{bmatrix}$$

4.4. Initial Conditions

The estimator requires initial knowledge of the state vector, i.e., the initial quaternions and angular velocity. The estimator also requires the initial value of P the perturbation covariance matrix, which is effectively the covariance of the error in the initial state. Although the initial state is known within the simulator in the real satellite this may not be known. For all tests in this report the initial state has been obtained by adding a random error term to the true initial state (the maximum value of this random error has been set to 20% of the initial state). A suitable initial value of P has been determined by tuning, see Section 4.8. If an initial error greater than 20% was expected P would require to be re-evaluated to minimise transient errors.

4.5. Quaternion Normalisation

A proper quaternion of rotation possesses the quality $q^T q = 1$. It was shown in [Bar-Itzhack] that if this condition is enforced after each state calculation the estimator error will be reduced. This has therefore been included in the estimator. The calculation necessary to normalise the quaternion is explained fully in [Bar-Itzhack]. In summary, the following modifications are made to the method explained in section 4.3.

1) (4.3.3) is changed to (4.5.1):

$$d\hat{z}_{k+1/k+1} = d\hat{z}_{k+1/k} + K_{k+1}(b_{meas,k+1} - D \times b_{orb,k+1} - H_{k+1/k} d\hat{z}_{k+1/k}) \quad (4.5.1)$$

the state is then updated as before

$$\hat{z}_{k+1/k+1} = \hat{z}_{k+1/k} + d\hat{z}_{k+1/k+1} \quad (4.5.2)$$

2) The quaternion can then be normalised using (4.5.3):

$$q^* = q/|q| \quad (4.5.3)$$

3) An additional step is then added after step 5 in Section 4.3 to propagate dz

$$dz_{k+2/k+1} = \begin{bmatrix} 0_{3x1} \\ \underline{\Omega}\left[\left(\hat{q}_{k+1/k+1}\hat{q}^T_{k+1/k+1}\right)d\hat{q}_{k+1/k+1}\right] \end{bmatrix} \quad (4.5.4)$$

where $q_{k+1/k+1}$ is the quaternions before being normalised.

The reason for the additional dz term can be seen by substituting (4.5.2) into (4.5.3) (just considering the quaternion part of the state vector),

$$\hat{q}^*_{k+1/k+1} = \frac{\hat{q}_{k+1/k} + d\hat{q}_{k+1/k+1}}{\left|\left(\hat{q}_{k+1/k} + d\hat{q}_{k+1/k+1}\right)\right|}$$

Expanding this with a first-order Taylor series gives:

$$\hat{q}^{*}_{k+1/k+1} \approx (\hat{q}_{k+1/k} + d\hat{q}_{k+1/k+1})(1 - \hat{q}^{T}_{k+1/k} d\hat{q}_{k+1/k+1})$$

Neglecting second order terms this becomes:

$$\hat{q}^{*}_{k+1/k+1} \approx \hat{q}_{k+1/k} + d\hat{q}_{k+1/k+1} - \hat{q}_{k+1/k}\hat{q}^{T}_{k+1/k} d\hat{q}_{k+1/k+1}$$

There is, thus, an extra term which is subtracted from the quaternions. To maintain the Kalman Filter algorithm this term must be added back. Hence, the additional term in (4.5.1) and the propagation of this additional term in (4.5.4).

4.6. Results Analysis Methods

To enable rapid, consistent assessment of the estimator's performance during any simulation an automatic analysis script has been added to the existing data analysis software. The performance is assessed by two methods, first, the pointing error, and second the spin rate error. The function plots the angle (γ) between the true SCS z-axis and the estimated SCS z-axis, calculated using equation 4.6.1. The mean and standard deviation of this angle, after initial transients, are also calculated.

$$\gamma = \cos^{-1}\left[\begin{bmatrix} 2(\hat{q}_1\hat{q}_3 - \hat{q}_2\hat{q}_4) & 2(\hat{q}_2\hat{q}_3 + \hat{q}_1\hat{q}_4) & -\hat{q}_1^2 - \hat{q}_2^2 + \hat{q}_3^2 + \hat{q}_4^2 \end{bmatrix} \begin{bmatrix} 2(q_1q_3 - q_2q_4) \\ 2(q_2q_3 + q_1q_4) \\ -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{bmatrix}\right] \qquad (4.6.1)$$

The function them plots the percentage error in spin rate, using equation 4.6.2. Then calculates the mean and standard deviation, after initial transients. The output of a standard simulation is shown in Figure 4.6.1., the function is written as a MatLab m-file and is shown in Appendix A.

$$e = 100 \cdot \frac{\left|\boldsymbol{\Omega}^{z}_{si} - \hat{\boldsymbol{\Omega}}^{z}_{si}\right|}{\boldsymbol{\Omega}^{z}_{si}} \qquad (4.6.2)$$

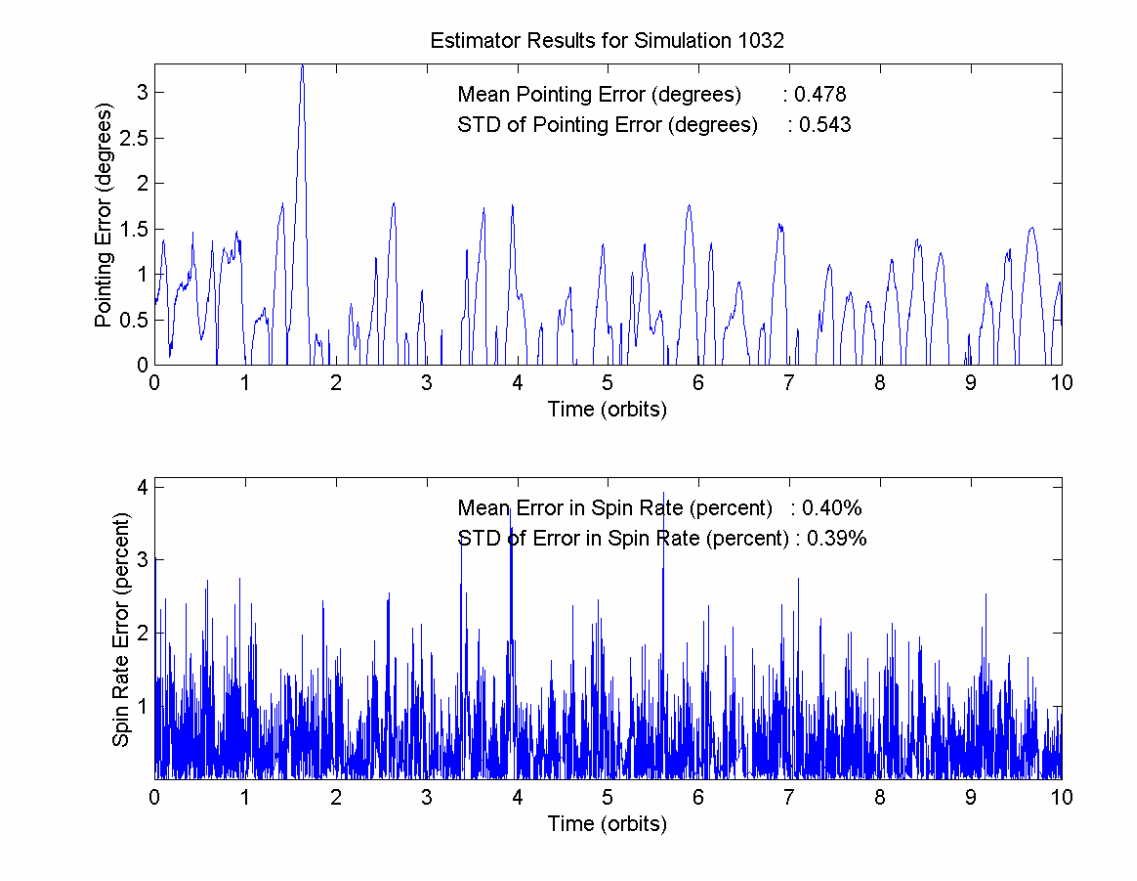

Figure 4.6.1 - Sample of Output for Results Analysis Program

4.7. Incorporation of Additional Sensors

To use the Sun sensor within the estimator we first need to calculate the vector of the Sun's position from the satellite given the sensor reading. Each channel of the Sun sensor gives an angle to the Sun, from this we can determine that the Sun vector is somewhere on a cone. Using two channels we have two cones, the Sun vector must therefore lie on one of the two intersection points of those two cones. If the position of the Sun is known a few seconds before we can determine which of the two intersections is most likely to be correct.

All Sun sensors are in the x-y plane. The SCS co-ordinate system is rotated in the x-y plane so that the new x-axis is aligned with the sensor plane nornal axis. This new co-ordinate system is refered to as the Sun Sensor Co-ordinate System (SSCS). In this co-ordinate system it can be easily seen that the vector of all possible Sun vectors is given by,

$$\begin{bmatrix} {}^{ss}x \\ {}^{ss}y \\ {}^{ss}z \end{bmatrix} = \begin{bmatrix} \cos\alpha \\ \sin\alpha\cos\phi \\ \sin\alpha\sin\phi \end{bmatrix} \tag{4.7.1}$$

where α is the Sun sensor reading and ϕ is the parameter describing the cone. This can then be rotated back into the satellite co-ordinate system using the rotation matrix,

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{ss}x \\ {}^{ss}y \\ {}^{ss}z \end{bmatrix} \tag{4.7.2}$$

Considering first the Sun sensor which points in the positive y direction, θ for channel 1 will be 60^o and 120^o for channel two. Therefore, substituting 4.7.1 into 4.7.2 with the two values for θ gives the two cone vectors.

$$\begin{bmatrix} \frac{1}{2}\cos\alpha - \frac{\sqrt{3}}{2}\sin\alpha\cos\phi \\ \frac{\sqrt{3}}{2}\cos\alpha + \frac{1}{2}\sin\alpha\cos\phi \\ \sin\alpha\sin\phi \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} -\frac{1}{2}\cos\beta - \frac{\sqrt{3}}{2}\sin\beta\cos\varepsilon \\ \frac{\sqrt{3}}{2}\cos\beta - \frac{1}{2}\sin\beta\cos\varepsilon \\ \sin\beta\sin\varepsilon \end{bmatrix} \tag{4.7.3}$$

Equating these two vectors and solving for ε and ϕ,

$$\cos\phi = \frac{2\cos\beta - \cos\alpha}{\sqrt{3}\sin\alpha} \tag{4.7.4}$$

$$\cos\varepsilon = \frac{\cos\beta - 2\cos\alpha}{\sqrt{3}\sin\beta} \tag{4.7.5}$$

Therefore, given both channels of the sensor reading, (i.e. α and β), ϕ or ε can be found from (4.7.4) or (4.7.5). Note, there will be two values of ϕ or ε because of the inverse cosine. These values can then be substituted into (4.7.3) to give the two possible Sun vectors.

To select the correct one there are two possible methods, 1) Using the last known position of the Sun vector, the distance between this and the two possible solutions can be determined. The one with the smallest distance is assumed to be the correct vector. Because the sample time is very small compared to the speed at which the satellite moves this should always be reliable. 2) The estimator uses a model of the Sun's position in OCS, this can be rotated into SCS using the last known attitude. The correct solution can then be select by calculating which is closest to this vector as in method 1. Both methods have been tested over a variety of orbits and both always select the correct vector. The current version of the program actually uses the second method.

For the sensor which looks in the negative x direction, (4.7.3) changed to (4.7.6) but everything else remains the same.

$$\begin{bmatrix} -\frac{\sqrt{3}}{2}\cos\alpha - \frac{1}{2}\sin\alpha\cos\phi \\ \frac{1}{2}\cos\alpha - \frac{\sqrt{3}}{2}\sin\alpha\cos\phi \\ \sin\alpha\sin\phi \end{bmatrix} \tag{4.7.6}$$

This calculation has been implemented as a Simulink S-function, the program listing is given in Appendix A.

The code has been tested to ensure it calculates the Sun Vector accurately. By comparing the result to the true Sun vector the average error is of the order of 10^{-12}. This was taken as validation that the method is reliable. To cope with the case when there is no reading from the sensor or the sensor is picking up background light, if either channel is reading less than 0.1 then the code outputs a flag indicating that no reading is available.

The output of this S-function is fed directly to the estimator. The estimator must be modified to deal with two sensor readings. The equations in the estimator will be different depending on whether the Sun sensor measurements is available or not. If it is not the equations remain as they are in Section 4.3, if the Sun sensor is available (4.3.3) will change to (4.7.7). Note, if two Sun sensor readings are available the average of the two readings is used.

$$d\hat{z}_{k+1/k+1} = d\hat{z}_{k+1/k} + K_{k+1}\left[\begin{bmatrix}(b_{meas,k+1} - D \times b_{orb,k+1})\\ (s_{meas,k+1} - D \times s_{orb,k+1})\end{bmatrix} - H_{k+1/k} d\hat{z}_{k+1/k}\right] \tag{4.7.7}$$

where $s_{meas,k+1}$ is the Sun Vector measurement (in SCS) at time k+1

$s_{orb,k+1}$ is the Sun Vector predicted by the model (in OCS) at time k+1.

To use (4.7.7), K needs to be a 6x7 matrix (previously it was 3x7). This is achieved by redefining H which was defined by (4.3.2). So, if the Sun sensor measurement is available (4.3.2) becomes equation (4.7.8). If K is now recalculated using equation (4.3.1) it will be 6x7 as required.

$$H_{k+1/k} = \begin{bmatrix}0_{6x3} & h_1 & h_2 & h_3 & h_4\end{bmatrix} \tag{4.7.8}$$

$$h_1 = 2\begin{bmatrix}\begin{bmatrix}\hat{q}_{1,k} & \hat{q}_{2,k} & \hat{q}_{3,k}\\ \hat{q}_{2,k} & -\hat{q}_{1,k} & \hat{q}_{4,k}\\ \hat{q}_{3,k} & -\hat{q}_{4,k} & -\hat{q}_{1,k}\end{bmatrix} b_{orb,k+1}\\ \begin{bmatrix}\hat{q}_{1,k} & \hat{q}_{2,k} & \hat{q}_{3,k}\\ \hat{q}_{2,k} & -\hat{q}_{1,k} & \hat{q}_{4,k}\\ \hat{q}_{3,k} & -\hat{q}_{4,k} & -\hat{q}_{1,k}\end{bmatrix} s_{orb,k+1}\end{bmatrix}$$

$$h_3 = 2\begin{bmatrix}\begin{bmatrix}-\hat{q}_{3,k} & \hat{q}_{4,k} & \hat{q}_{1,k}\\ -\hat{q}_{4,k} & -\hat{q}_{3,k} & \hat{q}_{2,k}\\ \hat{q}_{1,k} & \hat{q}_{2,k} & \hat{q}_{3,k}\end{bmatrix} b_{orb,k+1}\\ \begin{bmatrix}-\hat{q}_{3,k} & \hat{q}_{4,k} & \hat{q}_{1,k}\\ -\hat{q}_{4,k} & -\hat{q}_{3,k} & \hat{q}_{2,k}\\ \hat{q}_{1,k} & \hat{q}_{2,k} & \hat{q}_{3,k}\end{bmatrix} s_{orb,k+1}\end{bmatrix}$$

$$h_2 = 2\begin{bmatrix}\begin{bmatrix}-\hat{q}_{2,k} & \hat{q}_{1,k} & -\hat{q}_{4,k}\\ \hat{q}_{1,k} & \hat{q}_{2,k} & \hat{q}_{3,k}\\ \hat{q}_{4,k} & -\hat{q}_{3,k} & -\hat{q}_{2,k}\end{bmatrix} b_{orb,k+1}\\ \begin{bmatrix}-\hat{q}_{2,k} & \hat{q}_{1,k} & -\hat{q}_{4,k}\\ \hat{q}_{1,k} & \hat{q}_{2,k} & \hat{q}_{3,k}\\ \hat{q}_{4,k} & -\hat{q}_{3,k} & -\hat{q}_{2,k}\end{bmatrix} s_{orb,k+1}\end{bmatrix}$$

$$h_4 = 2\begin{bmatrix}\begin{bmatrix}\hat{q}_{4,k} & \hat{q}_{3,k} & -\hat{q}_{2,k}\\ -\hat{q}_{3,k} & \hat{q}_{4,k} & \hat{q}_{1,k}\\ \hat{q}_{2,k} & -\hat{q}_{1,k} & \hat{q}_{4,k}\end{bmatrix} b_{orb,k+1}\\ \begin{bmatrix}\hat{q}_{4,k} & \hat{q}_{3,k} & -\hat{q}_{2,k}\\ -\hat{q}_{3,k} & \hat{q}_{4,k} & \hat{q}_{1,k}\\ \hat{q}_{2,k} & -\hat{q}_{1,k} & \hat{q}_{4,k}\end{bmatrix} s_{orb,k+1}\end{bmatrix}$$

The matrix R, the measurement error covariance, also needs to be redefined as a 6x6 matrix if the Sun sensor measurement is available. The values of this matrix must be determined by tuning, see section 4.8.

At this stage the Earth horizon sensor has not been included in the estimator. The EHS only provides good readings when the attitude perturbation is less than 5^o (the angle between the z-axis in OCS and the z-axis in SCS). In future work inclusion of the EHS may allow greater accuracy at low perturbation angles.

4.8. Estimator Tuning

4.8.1. Introduction

There are three matrices used within the estimator which must be determined before it can be used. These are $P_{o\ (7x7)}$ – the covariance of the error in the initial state, $Q_{(7x7)}$ – the covariance of the system model error, and $R_{(3x3\ or\ 6x6)}$ – the covariance of the measurement error. (*Note: the R matrix size changes depending on whether the Sun sensor is included*) Although it would be possible to attempt to measure these covariances, modern views believe that obtain the best closed loop controller performance it is better to tune these covariances. This section of the report discusses methods to find initial values for these matrices and methods for their optimisation.

Initially it would seem that there are 77 parameters which need to be determined (noting that covariance matrices are always symmetric). To simplify the problem various parameters within each matrix can be grouped by assuming that the covariance of the error in the terms will be the same. For example, it is reasonable to assume that the covariance of the errors in the Ω^x and Ω^y terms will be the same. The Ω^z term may to different because it involves the spin of the satellite. Using these agreements the P and Q matrices have been grouped as shown in (4.8.1) and R matrix as shown in (4.8.2).

$$P \text{ and } Q = \begin{bmatrix} a & b & c & j & j & k & k \\ b & a & c & j & j & k & k \\ c & c & d & l & l & m & m \\ j & j & l & e & g & i & i \\ j & j & l & g & e & i & i \\ k & k & m & i & i & f & h \\ k & k & m & i & i & h & f \end{bmatrix} \tag{4.8.1}$$

This grouping reduces the total number of parameters to 32. The P matrix can be tuned separately from Q and R because P only effects the initial transient errors. Q and R can be tuned ignoring transient effects. This therefore reduced the problem further. A number of assumptions are made in grouping these parameters which in the general case may not be true. However, to make the tuning problem manageable it is necessary to reduce the number of parameters. If more time is available or more computing power it would not be necessary to use these groupings.

$$R = \begin{bmatrix} a & c & c & 0 & 0 & 0 \\ c & b & d & 0 & 0 & 0 \\ c & d & b & 0 & 0 & 0 \\ 0 & 0 & 0 & e & f & f \\ 0 & 0 & 0 & f & e & f \\ 0 & 0 & 0 & f & f & e \end{bmatrix} \tag{4.8.2}$$

4.8.2. Initial Values

All the tuning methods proposed below require a starting point. Within this estimator it is not possible to use any set of numbers as a starting point because if they are not reasonable accurate the system covariance matrix P will diverge. This will rapidly cause the inverted matrix in equation 4.3.1 to become singular. The estimator has an automatic shut-down mode so that the simulator will not crash if this occurs, it just switches the estimator off and gives a warning message.

The initial values used in these programs were obtained by taking a typical value of each quantity and assuming a 1% error. Typical values were obtained by studing the results of a number of simulations. Assuming a mean error of zero, the variance of the error will just be the square of this 1% error. Covariances are just the product of 1% errors in the two quantities considered. This is a very '*hand wavy*' method, however, it does seem to produce a good starting point.

Using this method the initial matrix set gave any estimation pointing accuracy of approximately 5^o.

4.8.3. Manual Tuning

The first method of tuning tried was simple trial and error manual tuning. This is simply a matter of taking each parameter in turn and trying different values to try to find a good set. This is quite an effective method. It is possible to get a feel for which parameters are the most important and for the sensitivity of each parameter. For example, it is clear the four quaternion variances and the z angular velocity variance have a large effect of the results (i.e. $Q_{(3,3)}$, $Q_{(4,4)}$, $Q_{(5,5)}$, $Q_{(6,6)}$ and $Q_{(7,7)}$). However, this method is very time consuming. It can easily take a week to tune two or three parameters and with this method it is not possible to just leave it running, it requires continuos user intervention. Using this method the pointing accuracy was reduced from approximately 5^o to $\approx 1.5^o$

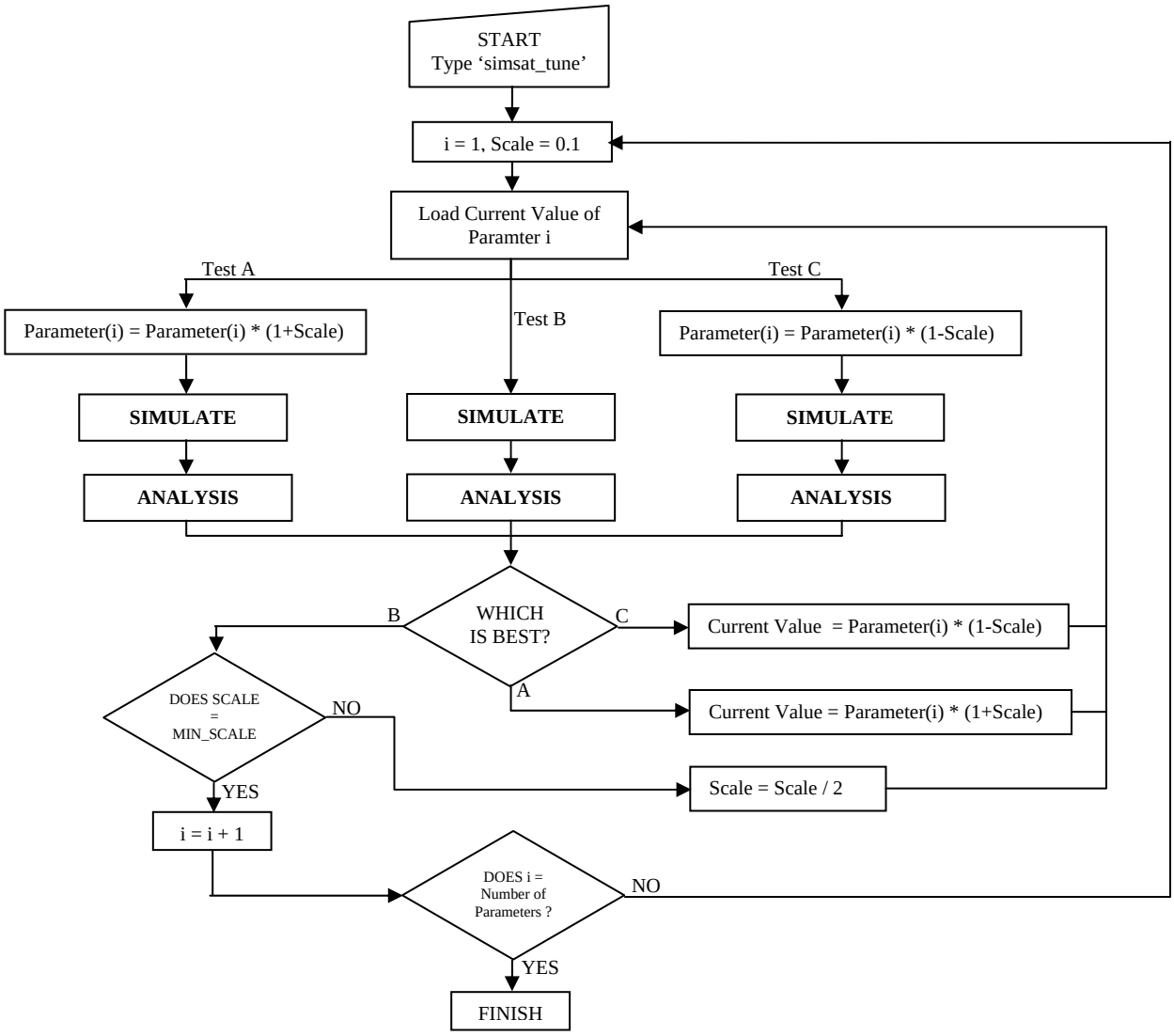

Figure 4.8.1 - Flow Chart of Automatic Tuner

4.8.4. Automatic Tuning

The method attempts to automate the manual tuning process. The method is shown in a flow chart in Figure 4.8.1. This algorithm has been run starting with a 'scale' of 10% (see flow chart), testing each new value over one five orbit simulation with starting conditions $\psi=5.0^o$, $\theta=7.0^o$, $\varphi=0.0^o$, $\Omega=[0.0005, 0, 0.02]$rads/s and the spin reference of 0.02rads/s. The simulation results where analysed using the cost function (Mean Point Error + S.D. Point Error + Mean Spin Rate Error + S.D. Spin Rate Error).

This method didn't improve the estimator accuracy within the week it was run. It is a very slow method and tends to find only local minima. Even if the method found the global minimum of one variable, if a different variable is then changed the minimum of the first will also change. This method has therefore been abandoned.

4.8.5. Genetic Algorithm Method

Because of the multi-parameter, non-linear, multi-minimum nature of this optimisation problem a logical approach was to use a genetic algorithm (GA) and so a simple genetic tuner has been written. Figure 6.8.2 shows the general construction of the algorithm. The full program listing (as a MatLab m-file) is given in Appendix A.

Each chromosome contains all the covariance parameters in the matrices that are being tuned. The initial population contains one previous 'good' chromosome, and a slight mutation of this good chromosome. The rest are randomly generated where each gene is of the form of (4.8.3).

$$Gene(i) = rand \times 10^{-(rand \times \max\_power)} \tag{4.8.3}$$

where *rand* is a random number between 0 and 1.

max_power is a constant (set to 12 for these simulations)

Each chromosome is tested over one five-orbit simulation with starting conditions ψ=5.0°, θ=7.0°, φ=0.0°, Ω=[0.0005, 0, 0.02]rads/s and the spin reference is 0.02rads/s. The simulation results are analysed and the cost function is evaluated. The cost used is:

$$Cost = \frac{1}{\text{Mean Pointing Error} + \text{S.D.Pointing Error} + \text{Mean Spin Rate Error} + \text{S.D.Spin Rate Error}}$$

Once each chromosome in the population has been simulated the costs are sorted into a quadratic distribution between 1-10 (the quadratic distribution is used so that when a parent is select the better the cost the more likely it is to be selected). A new population is then generated either by Mutation or by Cross-Over. The Mutation option takes a single parent chromosome and then for each gene depending on the value of a random number it either keeps the same value or generate a new random gene. The Cross-Over option takes two parent chromosomes ('A' and 'B') and forms a new chromosome by randomly selecting each gene from either parent 'A' or parent 'B'. For example, gene one in the new chromosome may be gene one from parent 'B', whereas gene two in the new chromosome may be gene two from parent 'A'. Once a large enough new population is generated the cycle starts again.

Because of the time taken to run simulations this algorithm is still relatively slow. However, after running for about one week this algorithm improved the estimator's accuracy (with the Sun sensor) from a mean pointing error of 1.5° to an accuracy of 0.58°. This is a good improvement and shows that the algorithm works. If a wide range of simulations were included (i.e. a full spin test, see section 5.1) and larger populations used (this run used a population of ten chromosomes) then even better results are likely to be obtained. To do this either more powerful computers or more time would be required.

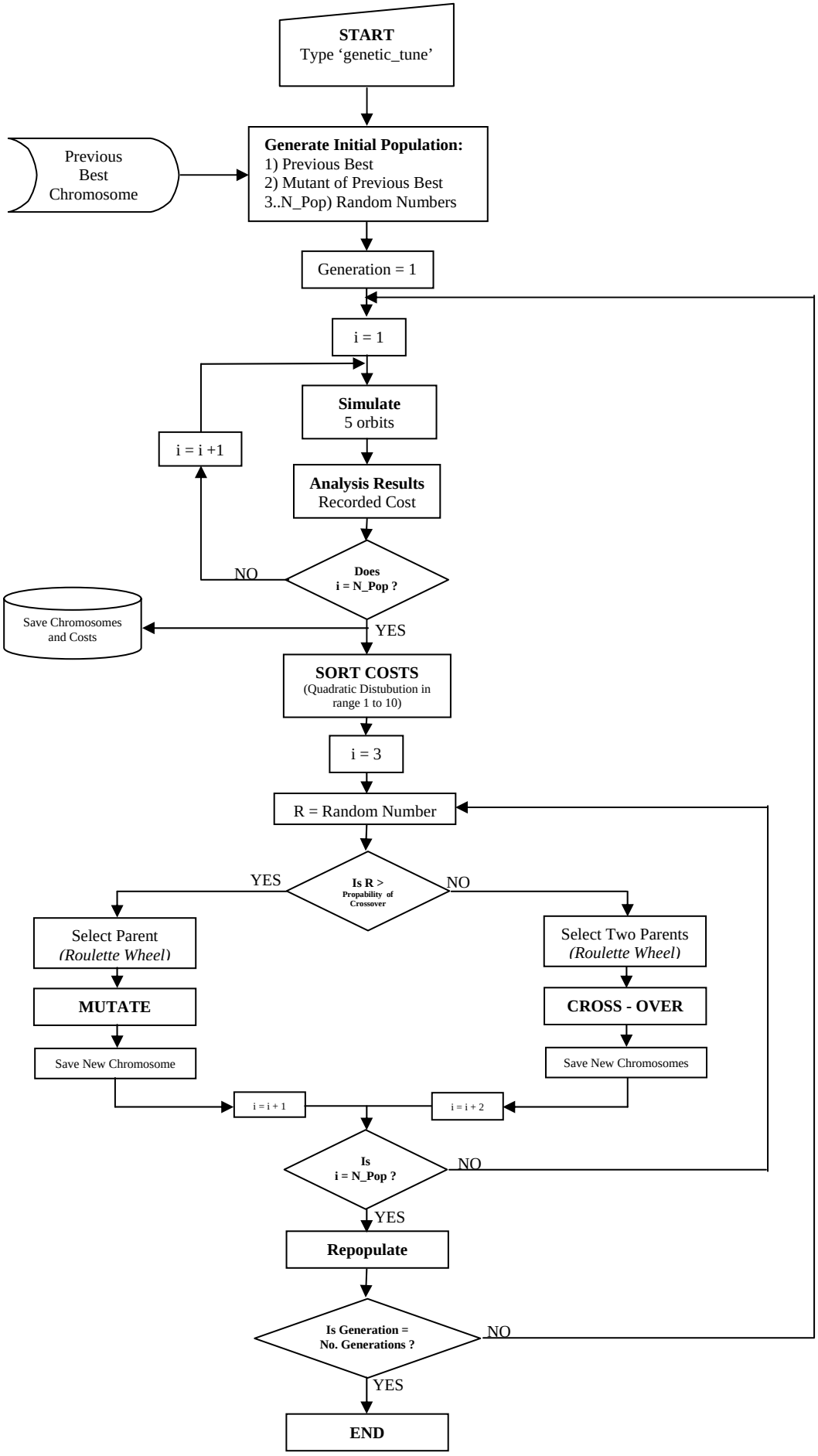

Figure 4.8.2 - Flow Chart of Genetic Algorithm

4.9. Estimator Accuracy Results

The best average results obtained over a number of orbits are shown in Table 4.9.1 *(these results are averages from Spin Test A and B with the predictive controller, see section 1.10).* Better results have been obtained by tuning the estimator for a specific orbit (this has resulted in pointing errors as low as 0.1^0), however, these don't work in the general case.

	Mean	**Standard Deviation**	**Worst**
Pointing Error (deg)	0.58	0.56	1.72
Spin Rate Error, %	0.39	0.37	1.23

Table 4.9.1 - Estimator Accuracy Results

5. Satellite Stability with Estimator

5.1. Introduction

There are three controllers of interest that are available within the simulator, these are the Predictive Controller [Tabuada-99], the Energy Controller [Wisniewski-96], and the Alpha-Beta Controller [Ong-92]. The predictive controller attempts to predict, for all possible actuations, the change in angular velocity of the satellite if that actuation was applied. It then determines which of these will cause the greatest reduction in the satellite's kinetic energy. The energy controller uses the control law given in (5.1.1),

$$^{c}m(t) = h^{c}\boldsymbol{\Omega}_{co}(t) \times {}^{c}B(t) \tag{5.1.1}$$

where h is a positive constant. The alpha-beta controller is the controller currently used on PoSat-1. This controller uses the control law given in (5.1.2) where α is the angle between the z-axis of the OCS and the expected geomagnetic field and β is the

$$^{c}m^{z} = k\left(\frac{d\beta}{dt} - \frac{d\alpha}{dt}\right) \tag{5.1.2}$$

angle between the z-axis of the OCS and the measured geomagnetic field. The alpha-beta controller does not require an estimator. Both of Predictive Controller and the Energy controller can control the spin of the satellite as well as the attitude, while the alpha-beta controller only controls attitude. These controllers are described and their performance without an estimator compared in detail in [Tabuada-98].

To compare the performance of each of these controllers three standard tests are used which are modifications of tests proposed in [Tabuada-98]. Each test consists of a batch of ten simulations, each ten orbits long with different starting conditions. The original tests where all conducted on the same date but at different times of day. The time was changed to change the magnetic field experienced by the satellite, only the relative position of the satellite and Earth were important. However, now the Sun sensor is used, the relative position of the Sun and Earth is also important so it is also necessary to test at different times of year. The three tests are referred to as Spin Test A, B, and C and are described below:

Spin Test A: Each simulation is started at a pitch angle = 60^{o} and a yaw angle = 0^{0}. the roll angle is different for each simulation. The initial angular velocity is [0.001037 0 0.02]rads/s and reference spin (the spin rate the controller should try to maintain) is set to 0.02rads/s.

Spin Test B: Each simulation is started at a pitch angle = 5^{o} and a yaw angle = 0^{0}. The roll angle is different for each simulation. The initial angular velocity is [0.001037 0 0]rads/s and reference is set to 0.02rads/s.

Spin Test C: Each simulation is started at pitch = 3^{o}, yaw = 0^{o} and roll = 3^{o} and angular velocity [0 0 0]rads/s. The reference spin is set to zero. This test is used to run the alpha-beta controller. (if an initial angular velocity is included the alpha-beta with not stabilise the satellite).

Test A simulates the case where the satellite is distubed by a large angle, test B simulates the case where the satellite needs to be spun up from zero spin rate. Full details of these tests and all the settings used within the controllers are given in Appendix B.

5.2. Bench Marks

To be able to see the effect of the estimator on the energy and predictive controller, tests are first conducted without the estimator. Table 5.2.1, Table 5.2.2 and Table 5.2.3 show a summary of the results from the predictive controller for the three tests. Table 5.2.4, Table 5.2.5 and Table 5.2.6 show the results of Tests A, B and C for the energy controller. Table 5.2.7 shows Test C for the Alfa-Beta controller. Although, the alpha-beta does not require an estimator it is included because this is the controller currently used on PoSat-1. For a controller-estimator package to be used it must perform better than this alpha-beta controller. These results can thus be used to determine the overall effectiveness of the controller and estimator together.

	Settling Time to within 5° (orbits)	**Steady State Pointing Accuracy (degrees)**	**Spin Rate Accuracy, ${}^{c}\Omega^{z}$ (rad/s)**	**Satellite Energy (J)**
MEAN	3.22	1.89	$5.7\text{x}10^{-5}$	$2.99\text{x}10^{5}$
STANDARD DEVIATION	1.73	0.26	$5.0\text{x}10^{-5}$	$3.99\text{x}10^{4}$
WORST	6.21	2.22	$1.6\text{x}10^{-4}$	$3.95\text{x}10^{5}$

Table 5.2.1 - Predictive Controller, Spin Test A (no estimator)

	Settling Time to within 5° (orbits)	**Steady State Pointing Accuracy (degrees)**	**Spin Rate Accuracy, ${}^{c}\Omega^{z}$ (rad/s)**	**Satellite Energy (J)**
MEAN	1.65	1.83	$3.6\text{x}10^{-5}$	$2.57\text{x}10^{5}$
STANDARD DEVIATION	1.06	0.11	$8.0\text{x}10^{-6}$	$2.26\text{x}10^{4}$
WORST	4.30	1.99	$4.8\text{x}10^{-5}$	$2.97\text{x}10^{5}$

Table 5.2.2 - Predictive Controller, Spin Test B (no estimator)

	Settling Time to within 1° (orbits)	**Steady State Pointing Accuracy (degrees)**	**Spin Rate Accuracy, ${}^{c}\Omega^{z}$ (rad/s)**	**Satellite Energy (J)**
MEAN	4.00	0.14	*No Spin*	$1.66\text{x}10^{5}$
STANDARD DEVIATION	2.50	0.08	*No Spin*	$1.18\text{x}10^{4}$
WORST	7.30	0.28	*No Spin*	$1.78\text{x}10^{5}$

Table 5.2.3 - Predictive Controller, Spin Test C (no estimator)

	Settling Time to within 5° (orbits)	**Steady State Pointing Accuracy (degrees)**	**Spin Rate Accuracy, ${}^{c}\Omega^{z}$ (rad/s)**	**Satellite Energy (J)**
MEAN	3.53	1.76	$8.3\text{x}10^{-4}$	$2.52\text{x}10^{5}$
STANDARD DEVIATION	2.32	0.24	$4.4\text{x}10^{-4}$	$6.26\text{x}10^{4}$
WORST	10.0	2.02	$1.4\text{x}10^{-3}$	$2.78\text{x}10^{5}$

Table 5.2.4 - Energy Controller, Spin Test A (no estimator)

	Settling Time to within 5^o (orbits)	Steady State Pointing Accuracy (degrees)	Spin Rate Accuracy, ${}^c\Omega^z$ (rad/s)	Satellite Energy (J)
MEAN	1.99	1.94	9.4×10^{-4}	2.60×10^5
STANDARD DEVIATION	0.51	0.31	4.6×10^{-4}	1.38×10^3
WORST	2.74	2.33	1.4×10^{-3}	2.63×10^5

Table 5.2.5 - Energy Controller, Spin Test B (no estimator)

	Settling Time to within 1^o (orbits)	Steady State Pointing Accuracy (degrees)	Spin Rate Accuracy, ${}^c\Omega^z$ (rad/s)	Satellite Energy (J)
MEAN	0.8	0.01	*No Spin*	2.51×10^5
STANDARD DEVIATION	0.2	0.02	*No Spin*	0.00×10^5
WORST	1.0	0.05	*No Spin*	2.51×10^5

Table 5.2.6 - Energy Controller, Spin Test C (no estimator)

	Settling Time to within 1^o (orbits)	Steady State Pointing Accuracy (degrees)	Spin Rate Accuracy, ${}^c\Omega^z$ (rad/s)	Satellite Energy (J)
MEAN	No Settling	1.26	*No Spin*	3.20×10^3
STANDARD DEVIATION	No Settling	0.22	*No Spin*	5.63×10^1
WORST	No Settling	1.79	*No Spin*	3.29×10^3

Table 5.2.7 – Alfa-Beta Controller, Spin Test C

Comparing the mean results for the predictive controller with those of the energy controller, for test A the energy controller has a slightly better pointing accuracy, but for test B the predictive controller has better accuracy. However, considering the standard deviations of these results, the performances can be considered equivalent. Similarly, with the satellite's energy, both the energy controller and predictive controller have very similar results for both tests A and B. Considering the spin rate accuracy, the predictive controller is considerably better than the energy controller. In conclusion, for test A and B without the estimator both the energy and predictive controller have similar performance with pointing accuracy and energy but the predictive is better at controlling spin rate.

Considering test C, the energy controller clearly has the best pointing accuracy and settling time. However, the energy consumed is considerably larger than that used by the alpha-beta controller. The predictive controller is similar to the energy controller in having good settling time and pointing accuracy but high energy. For a small satellite like PoSat-1 it is not necessary to have a pointing accuracy of 0.04^0, it would be much more important to conserve energy. The conclusion is that for test C, the predictive and energy controllers are firing much more than is necessary. Section 5.4 considers the possibility of tuning the controllers to improve this situation.

5.3. Estimator Results

Table 5.3.8 and Table 5.3.9 show the results for test A and B for the predictive controller with the estimator, but without the Sun sensor measurements. Table 5.3.10 and Table 5.3.11 show the results for test A and B for the predictive controller with the estimator and with the Sun sensor measurements. Table 5.3.12, Table 5.3.13 and Table 5.3.14 show results for tests A,B and C for the energy controller with the estimator with the Sun sensor measurements included.

	Settling Time to within 5º (orbits)	Steady State Pointing Accuracy (degrees)	Spin Rate Accuracy, ω_z (rad/s)	Satellite Energy (J)	Estimator Pointing Error (deg)	Estimator Pointing Error s.d. (deg)	Estimator Spin Rate Error, %	Estimator Spin Rate Error s.d., %
MEAN	8.90	3.14	1.2×10^{-4}	3.89×10^{5}	0.58	0.56	0.39	0.39
STANDARD DEVIATION	0.74	1.66	8.1×10^{-5}	9.60×10^{3}	0.07	0.05	0.02	0.03
WORST	10.0	4.97	3.0×10^{-4}	4.04×10^{5}	0.65	0.62	0.43	0.42

Table 5.3.8 - Predictive Controller, with Estimator (no Sun sensor), Spin Test A

	Settling Time to within 5º (orbits)	Steady State Pointing Accuracy (degrees)	Spin Rate Accuracy, ω_z (rad/s)	Satellite Energy (J)	Estimator Pointing Error (deg)	Estimator Pointing Error s.d. (deg)	Estimator Spin Rate Error, %	Estimator Spin Rate Error s.d., %
MEAN	8.62	3.29	1.4×10^{-4}	3.79×10^{5}	0.59	0.57	0.39	0.37
STANDARD DEVIATION	1.41	1.50	1.4×10^{-4}	5.74×10^{3}	0.07	0.06	0.02	0.03
WORST	10.0	5.89	4.1×10^{-4}	3.88×10^{5}	0.68	0.70	0.43	0.43

Table 5.3.9 - Predictive Controller, with Estimator (no Sun sensor), Spin Test B

	Settling Time to within 5º (orbits)	Steady State Pointing Accuracy (degrees)	Spin Rate Accuracy, ω_z (rad/s)	Satellite Energy (J)	Estimator Pointing Error (deg)	Estimator Pointing Error s.d. (deg)	Estimator Spin Rate Error, %	Estimator Spin Rate Error s.d., %
MEAN	9.17	3.22	1.1×10^{-4}	4.13×10^{5}	0.86	0.72	0.46	0.42
STANDARD DEVIATION	0.712	1.41	7.1×10^{-5}	6.81×10^{3}	0.06	0.03	0.03	0.02
WORST	10.0	6.10	2.2×10^{-4}	4.25×10^{5}	0.92	0.78	0.51	0.46

Table 5.3.10 - Predictive Controller, with Estimator (with Sun sensor), Spin Test A

	Settling Time to within 5º (orbits)	Steady State Pointing Accuracy (degrees)	Spin Rate Accuracy, ω_z (rad/s)	Satellite Energy (J)	Estimator Pointing Error (deg)	Estimator Pointing Error s.d. (deg)	Estimator Spin Rate Error, %	Estimator Spin Rate Error s.d., %
MEAN	9.20	3.07	1.2×10^{-4}	4.04×10^{5}	0.77	0.77	0.45	0.42
STANDARD DEVIATION	0.75	1.49	8.0×10^{-5}	4.14×10^{3}	0.07	0.07	0.03	0.03
WORST	10.0	5.48	3.1×10^{-4}	4.11×10^{5}	0.87	0.89	0.50	0.46

Table 5.3.11 - Predictive Controller, with Estimator (with Sun sensor), Spin Test B

	Settling Time to within 5° (orbits)	Steady State Pointing Accuracy (degrees)	Spin Rate Accuracy, $^{c}\Omega^{z}$ (rad/s)	Satellite Energy (J)	Estimator Pointing Error (deg)	Estimator Pointing Error s.d. (deg)	Estimator Spin Rate Error, %	Estimator Spin Rate Error s.d., %
MEAN	3.20	2.13	$6.0\text{x}10^{-4}$	$2.76\text{x}10^{5}$	0.96	0.84	1.80	1.33
STANDARD DEVIATION	0.25	0.16	$5.0\ \text{x}10^{-4}$	$8.57\text{x}10^{4}$	0.17	0.08	0.52	0.34
WORST	3.96	2.73	$1.6\ \text{x}10^{-3}$	$2.83\text{x}10^{5}$	1.06	0.92	2.02	1.52

Table 5.3.12 – Energy Controller with Estimator (with Sun sensor), Spin Test A

	Settling Time to within 5° (orbits)	Steady State Pointing Accuracy (degrees)	Spin Rate Accuracy, $^{c}\Omega^{z}$ (rad/s)	Satellite Energy (J)	Estimator Pointing Error (deg)	Estimator Pointing Error s.d. (deg)	Estimator Spin Rate Error, %	Estimator Spin Rate Error s.d., %
MEAN	2.27	1.94	$6.9\text{x}10^{-4}$	$2.65\text{x}10^{5}$	0.90	0.83	1.96	1.44
STANDARD DEVIATION	0.48	0.41	$3.9\text{x}10^{-4}$	$1.66\text{x}10^{3}$	0.10	0.07	0.02	0.04
WORST	2.83	2.53	$1.4\text{x}10^{-3}$	$2.69\text{x}10^{5}$	1.03	0.92	2.00	1.48

Table 5.3.13 - Energy Controller, with Estimator (with Sun sensor), Spin Test B

	Settling Time to within 5° (orbits)	Steady State Pointing Accuracy (degrees)	Spin Rate Accuracy, $^{c}\Omega^{z}$ (rad/s)	Satellite Energy (J)	Estimator Pointing Error (deg)	Estimator Pointing Error s.d. (deg)	Estimator Spin Rate Error, %	Estimator Spin Rate Error s.d., %
MEAN	6.39	6.9	$1.9\text{x}10^{-2}$	$2.73\text{x}10^{5}$	6.5	9.9	32.5	89.7
STANDARD DEVIATION	3.60	10.6	$5.9\text{x}10^{-3}$	$1.64\text{x}10^{4}$	11.3	17.6	26.5	96.4
WORST	10.0	31.6	$2.2\text{x}10^{-2}$	$3.05\text{x}10^{5}$	29.9	43.8	89.8	329.4

Table 5.3.14 - Energy Controller, with Estimator (with Sun sensor), Spin Test C

It is immediately noticeable from these results that the estimator has worse performance with the Sun sensor included. This is because a lot less time has been available to tune the estimator version with the Sun sensor. It is therefore difficult to draw conclusions about the improvement due to the Sun sensor inclusion. In fact, it would be difficult to ever make this comparison because it can always be argued that either version could be further tuned. Logically the more measurements available to the estimator the better it will work so we would expect, if more tuning time was available, the estimator with the Sun sensor would work better than without it. A curious results is that even though the estimator is working worse with the Sun sensor the overall performance of the estimator and controller is the same with and without the Sun sensor. This is difficult to explain but does suggest that there is merit in exploring the option of tuning the estimator-controller package together, i.e. closed loop tuning (see Section 5.4).

Looking at the effect of the estimator on the predictive controller. The pointing accuracy has deteriorated from $\approx 1.8^{o}$ to $\approx 3.2^{o}$. Similarly, the spin rate accuracy has been halved. The energy consumed has also increased slightly. Considering the effect of the estimator on the energy controller, the reduction in accuracy is much smaller. Pointing accuracy is reduced from $\approx 1.8^{o}$ to $\approx 2.0^{o}$. Spin rate and energy consumed are not significantly effected. Thus, comparing the controller estimator packages the energy controller is better from pointing accuracy, rapid settling and energy. However, the predictive still has a better spin rate accuracy.

As the energy controller results in Table 5.3.14 show the estimator does not work at all well during spin test C (the predictive controller results are not presented because the estimator was unstable so results could not be produced). The reason for this is that the estimator has been tuned with a z-component angular velocity ≈0.02rad/s, while in this test this has been reduced to zero. The estimator only works well under the conditions for which it was tuned. If test C is conducted with the normal spin rate these large errors don't occur. As a result it is impossible to compare the predictive or energy controllers with estimator to the alpha-beta controller. However, if the same reduction in performance is extrapolated from tests A and B to test C, it is clear that on every point, except energy consumed, the predictive and energy controller would be better than the alpha-beta controller.

A factor which has not been consider yet is the processor time required by the estimator and controller. On a small satellite it is important to minimise the computation effort that is required to control the attitude. Table 5.3.15 presents the time taken to run a five orbit simulation for each of the controllers. Although this is meaningless in terms of the satellite's computer time it does give us insight into the relative CPU time that may be required by each controller.

Controller	**Time Taken to Complete 5 orbits (min : sec)**
Alpha-Beta	14:30
Predictive + Estimator	43:28
Energy + Estimator	38:13

Table 5.3.15 - Processor Time Required to Simulate

5.4. Closed Loop Tuning

Both the predictive controller and the energy controller contain a number of parameters which can be adjusted to change there performance. All the tests above have be performed with standard values. The main area where improvements are necessary is the total energy consumed. This is the only area where the alpha-beta controller is better than the predictive and energy controllers. Time was not available during this work for this sort of tuning. However, the genetic algorithm used could be easily modified to do this simply by adding the required controller parameters to the chromosomes and changing the cost functions.

To determine the potential of this sort of tuning a number of tests have been performed varying the time between consecutive fires in the predictive controller. The original version attempted to fire the magnetorquers on a ten second cycle. Using a 'backoff' parameter this can be increased. The results in Table 6.10.16 show the predictive controller (without estimator) for spin test C after this parameter has been tuned (these results were obtained with Backoff = 250seconds). Note, that this test can not be conducted with the estimator as discussed above.

	Settling Time to within 1° (orbits)	Steady State Pointing Accuracy (degrees)	Spin Rate Accuracy, $^{c}\Omega^{z}$ (rad/s)	Satellite Energy (J)
MEAN	1.80	0.02	*No Spin*	7.67×10^{3}
STANDARD DEVIATION	0.32	0.01	*No Spin*	1.91×10^{1}
WORST	2.37	0.03	*No Spin*	1.01×10^{3}

Table 5.4.16 – Tuned Predictive Controller (no estimator), Spin Test C

Comparing these results to the alpha-beta controller results in table 5.4.7 the controller now uses similar energy but with a pointing accuracy about 50 times better. It is also noticeable that these accuracy results are better than the original controller on Test C when it spent much more energy.

6. Conclusions

New and improved models of the satellite's Sun sensor, Earth horizon sensor and magnetometer have been developed and implemented. Results from these sensors simulations compare well with telemetry data from PoSat-1. An attitude estimator has been developed and implements which has a pointing accuracy of just over one degree within one standard deviation and a spin rate error of less than one percent within one standard deviation. A number of methods of tuning the estimator have been tried and a successful genetic tuner has been implemented. This estimator has then been tested with the predictive and energy controller and compared to the bench mark alpha-beta controller.

Comparisons of the three controllers without the estimator showed that the predictive controller performs best on all criteria except energy consumed were the alpha beta controller in better. However, limited tuning of the controller demonstrated the potential to bring the energy used by the predictive in line with the alpha-beta controller. With the estimator in the loop the system worked best with the energy controller where there was a reduction in performance of only 0.2 degrees. With the predictive controller the drop in performance was nearly 1.4 degrees. Although direct comparison of these controllers with estimator to the alpha-beta was not possible, extrapolation of the results obtained suggest that they will still out perform the alpha-beta controller on accuracy.

7. Further Work

The follow list gives details of the areas in which further work is required:

1) Further improvements could be made to the Earth horizon sensor if a realistic model of the noise was developed. This could either be done by adding a random noise signal to the sensor in a attempt to match the telemetry or by developing greater understanding of the cause of the noise and modelling this.

2) The estimator can currently only cope with an initial state error of 20%. Further development of the initial state error covariance matrix would allow this condition to me removed. Once a large error could be tolerated a standard initial condition could be used removing the need for an initial input.

3) The estimator currently only uses the magnetometer and the Sun sensors. The inclusion of the Earth horizon sensor as well could help improve the estimator accuracy at low attitude perturbation angles. To include the EHS the vector to the Earth's horizon needs to calculated from the sensor reading. This can be calculated easily knowing the field of view vectors of the sensor. This can then be added to the estimator in the same ways as the Sun sensor.

4) Further work on closed loop tuning could reap significant rewards. The example given in section 5.4 demonstrated that tuning the controllers can make major improvements in areas like energy consumed and pointing accuracy. Modification of the genetic tuning algorithm presented in section 4.8.5 would be very simple to enable it to be used to tune the estimator and controller together.

5) A important area of development to allow this system to be used on a real satellite will be the reduction of the computation effort required to control the attitude. One potential method to do this would be to linearise the equations of motion about a number of attitudes and then use gain scheduling to determine the motion between these linearised points. At the moment the equations are linearised at every time step. Because, for the real satellite, the attitude perturbation from the stabilised position will be small, few linearisation would be required to give accurate results.

References

[Tabuada-99] Tabuada, Paulo; Alves, Pedro; Tavares, Pedro; Lima, Pedro. (1999) A Predictive Algorithm for Attitude Stabilisation and Spin Control of Small Satellites. European Control Conference (ECC'99), Kurlsruhe, Germany.

[Tabuada-98] Tabuada, Paulo; Alves, Pedro; Tavares, Pedro; Lima, Pedro. (1998). Attitude Control Strategies for Small Satellites. Institute for Systems and Robotics / IST internal report.

[Tavares-98a] Tavares, Pedro; Sousa, Bruno; Lima, Pedro. (1998). *A Simulator of Satellite Attitude Dynamics*. Proc. of the 3rd Portuguese Conference on Automatic Control, Vol. II, pp. 459-464, Coimbra, Portugal.

[Tavares-98b] Tavares, Pedro; Tabuada, Paulo; Lima, Pedro. (1998). *Project ConSat: Control of Small Satellites*. Control of Complex Systems Annual Joint Workshop, Ohrid, FYR of Macedonia.

[Tavares-99] Tavares, Pedro; Clements, Robert (1999) Update of Status of Small Satellite Simulator. ISR Internal Report RT-402-99.

[Steyn] Steyn, W.H., Full Satellite State Determination from Vector Observations

[Bar-Itzhack] Bar-Itzhack, I.Y. and Oshman, Y. (1985) Attitude Determination from Vector Observations: Quaternion Estimation, IEEE Transaction on Aerospace and Electronic Systems Vol.21 No.1 pp128-135.

[Wertz] Wertz, J.R. (1978), Spacecraft Attitude Determination and Control, Astrophysics and Space Science Library Vol.73 D.Reidel Publishing Company.

[Ong-92] Ong, W. T. (1992), Attitude Determination and Control of Low Earth Orbit Satellites, MSc Thesis, Dept. of Electric and Electrical Engineering, University of Surrey.

[Wisniewski-96] Wisniewski, R. (1996), Satellite Attitude Control using only Electromagnetic Actuation, Ph.D. Thesis, Dept. of Control Engineering, Aalborg University.

Appendix A

7.1. EHS Program Code

```
%% Earth Horizon Sensor Simulator
%
%          ConSat Simulator
%      Ver 2.0 16th March 1999
%
% (C) Institute for Systems and Robotics
%       IST / Lisbon - Portugal
%               <RAC>
%

function [sys,xo,str,ts]=ehs(t,x,u,flag)

global mstep;

switch flag
case 0
  sys=[0 0 1 14 0 1 1];
  x0=[];
  str=[];
  ts=[mstep 0];

case 3
   Angle_Earth=u(1);
   Sol=[u(2);u(3);u(4)];
   Att=[u(5) u(8) u(11);
        u(6) u(9) u(12);
        u(7) u(10) u(13)];
   Sensor_flag=u(14);

%  Calculate Vectors of Upper and Lower Field of View Limits
if Sensor_flag==1
   Upper_Limit_SCS =[-cos(16*pi/180);0; -sin(16*pi/180)];
   Lower_Limit_SCS =[-cos(38*pi/180);0; -sin(38*pi/180)];
else
   Upper_Limit_SCS =[0; cos(16*pi/180); -sin(16*pi/180)];
   Lower_Limit_SCS =[0; cos(38*pi/180); -sin(38*pi/180)];
end;

Upper_Limit_OCS =Att*Upper_Limit_SCS;
Lower_Limit_OCS =Att*Lower_Limit_SCS;

% Calculate intersection point of Upper and Lower Limits with plane passing
% through centre of Earth and at right angles to the Earth-Satellite Vector.

Scaler = -(cos(Angle_Earth)/(tan(Angle_Earth)*Upper_Limit_OCS(3)));
X_upper = Scaler*Upper_Limit_OCS(1);
Y_upper = Scaler*Upper_Limit_OCS(2);

Scaler = -(cos(Angle_Earth)/(tan(Angle_Earth)*Lower_Limit_OCS(3)));
X_lower = Scaler*Lower_Limit_OCS(1);
Y_lower = Scaler*Lower_Limit_OCS(2);

App_REarth = cos(Angle_Earth); %Radius of Earth as seen from Satellite
(Assume true Radius=1)

% Project Sol into X-Y Plane
Sol_XY=[Sol(1);Sol(2)];
temp=sqrt(Sol_XY(1)^2+Sol_XY(2)^2);
if temp>0.001
   Sol_XY=Sol_XY.*(1/temp);
end;

% Calculate Angle of Sun (in the plane) from the x-axis
Alfha = (pi)-(acos(Sol_XY(1)));

% Form Rotation Matrix from X,Y Space to U,V Space

Rotation=[cos(Alfha) -sin(Alfha); sin(Alfha) cos(Alfha)];

% Rotate Upper and Lower points to U,V space

Upper=[(Rotation(1,1)*X_upper)+(Rotation(1,2)*Y_upper)
(Rotation(2,1)*X_upper)+(Rotation(2,2)*Y_upper)];
Lower=[(Rotation(1,1)*X_lower)+(Rotation(1,2)*Y_lower)
(Rotation(2,1)*X_lower)+(Rotation(2,2)*Y_lower)];

% Determine Scaler for Elipice of Earth Horizon

a=abs(Sol(3));

% Thus Eqn of Earth Circumference is u^2+v^2=App_REarth
% Eqn of inner ellipse (u/a)^2+v^2=1

% Rotate Sun Vector to U,V Space

U_Sol=(Rotation(1,1)*Sol_XY(1))+(Rotation(1,2)*Sol_XY(2));

%Calculate Eqn of Line of the FOV of the sensor
temp=(Upper(1)-Lower(1));
if temp<0.001
   temp=0.001;
end;
if abs(atan((Upper(2)-Lower(2))/temp))<=(pi/4)
   m = (Upper(2)-Lower(2))/(Upper(1)-Lower(1));
   c = Upper(2)-(m*Upper(1));
   b = ((2*m*c)^2+(4*(1+m^2)*((App_REarth^2)-c^2)));
   inter1=[((-(2*m*c)+sqrt(b))/(2*(1+m^2))) m*((-
(2*m*c)+sqrt(b))/(2*(1+m^2)))+c];
   inter2=[((-(2*m*c)-sqrt(b))/(2*(1+m^2))) m*((-(2*m*c)-
sqrt(b))/(2*(1+m^2)))+c];
   Gradient_Flag=0;
else
   m = (Upper(1)-Lower(1))/(Upper(2)-Lower(2));
   c = Upper(1)-(m*Upper(2));
   b = ((2*m*c)^2+(4*(1+m^2)*((App_REarth^2)-c^2)));
   inter1=[m*((-(2*m*c)+sqrt(b))/(2*(1+m^2)))+c ((-
(2*m*c)+sqrt(b))/(2*(1+m^2)))];
   inter2=[m*((-(2*m*c)-sqrt(b))/(2*(1+m^2)))+c ((-(2*m*c)-
sqrt(b))/(2*(1+m^2)))];
   Gradient_Flag=1;
end;

NIP=-1;
% NIP (Number of Intersection Points is used to declare how many valid
intersection
% points exist. It can take a value of 0, 2 or 4. -1 is used as an initial value.

if b<=0 % Determines if there are any real interesctions
   NIP=0;

%This part deals with the case when a is small i.e. we are looking at a half
circle.
%It determines which side of the cirle the Sun is and then calculates the two
intersection
%points.

elseif a<0.01 % This ensures a can't tend to zero
   NIP=2;
   if (U_Sol>0)
      if (inter1(1)<=0)&(inter2(1)<=0)
         NIP=0;
      elseif (inter1(1)<=0)
         if Gradient_Flag
            inter1=[c 0];
         else
            inter1=[0 c];
         end;
      elseif (inter2(1)<=0)
         if Gradient_Flag
            inter2=[c 0];
         else
            inter2=[0 c];
         end;
      end
   end
   if (U_Sol<0)
      if (inter1(1)>=0)&(inter2(1)>=0)
         NIP=0;
      elseif (inter1(1)>=0)
         if Gradient_Flag
            inter1=[c 0];
         else
            inter1=[0 c];
         end;
      elseif (inter2(1)>=0)
         if Gradient_Flag
            inter2=[c 0];
         else
            inter2=[0 c];
         end;
      end;
   end;

% This part now deals with the case when a is significant. There are four
cases depending
% on the position of the Sun.

else
   if Gradient_Flag
      b = ((2*m*c/(a^2))^2+(4*(((m/a)^2)+1)*(1-(c/a)^2)));
   else
      b = ((2*m*c)^2+(4*((1/(a^2))+m^2)*(1-c^2)));
   end;
   if or(((b<=0)&(inter1(1)>0)&(U_Sol>0)),((b<=0)&(inter1(1)<0)&(U_Sol<0)))
      NIP=2;
```

```
  elseif (b<=0)
    NIP=0;
  else
    if Gradient_Flag
      inter3=[m*((-(2*m*c/(a^2))+sqrt(b))/(2*(((m/a)^2)+1)))+c ((-
(2*m*c/(a^2))+sqrt(b))/(2*(((m/a)^2)+1)))];
      inter4=[m*((-(2*m*c/(a^2))-sqrt(b))/(2*(((m/a)^2)+1)))+c ((-(2*m*c/(a^2))-
sqrt(b))/(2*(((m/a)^2)+1)))];
    else
      inter3=[((-(2*m*c)+sqrt(b))/(2*((1/(a^2))+m^2))) m*((-
(2*m*c)+sqrt(b))/(2*((1/(a^2))+m^2)))+c];
      inter4=[((-(2*m*c)-sqrt(b))/(2*((1/(a^2))+m^2))) m*((-(2*m*c)-
sqrt(b))/(2*((1/(a^2))+m^2)))+c];
    end;
    beta=acos(Sol(3));

% The next two statements deal with the case when the elliptic intersections
are outside
% the Apperant Earth Radius.

    if ((sqrt(inter3(1)^2+(inter3(2))^2))>App_REarth)
      inter3=inter1;
    end;
    if ((sqrt(inter4(1)^2+(inter4(2))^2))>App_REarth)
      inter4=inter2;
    end;
    if (beta<(pi/2))&(U_Sol>=0)
      if (inter1(1)>=0)&(inter2(1)>=0)
        NIP=2;
      elseif (inter1(1)<=0)&(inter2(1)<=0)
        inter1=inter3;
        inter2=inter4;
        NIP=2;
      else
        NIP=2;
        if inter1(1)<0
          inter1=inter2;
        end;
        if inter3(1)<0
          inter2=inter3;
        else
          inter2=inter4;
        end;
      end;
    elseif (beta<(pi/2))&(U_Sol<0)
      if (inter1(1)<=0)&(inter2(1)<=0)
        NIP=2;
      elseif (inter1(1)>=0)&(inter2(1)>=0)
        inter1=inter3;
        inter2=inter4;
        NIP=2;
      else
        NIP=2;
        if inter1(1)>0
          inter1=inter2;
        end;
        if inter3(1)>0
          inter2=inter3;
        else
          inter2=inter4;
        end;
      end;
    elseif (beta>=(pi/2))&(U_Sol>=0)
      if
(inter3(1)>=0)&(inter4(1)>=0)&(inter3(1)~=inter1(1))&(inter4(1)~=inter2(1))
        NIP=4;
      elseif
(inter3(1)>=0)&(inter4(1)>=0)&((inter3(1)==inter1(1))|(inter4(1)==inter2(1)))
        NIP=0;
      elseif (inter1(1)<=0)&(inter2(1)<=0)
        NIP=0;
      else
        NIP=2;
        if inter1(1)<0
          inter1=inter2;
        end;
        if inter3(1)>0
          inter2=inter3;
        else
          inter2=inter4;
        end;
      end;
    elseif (beta>=(pi/2))&(U_Sol<0)
      if
(inter3(1)<=0)&(inter4(1)<=0)&(inter3(1)~=inter1(1))&(inter4(1)~=inter2(1))
        NIP=4;
      elseif
(inter3(1)>=0)&(inter4(1)>=0)&((inter3(1)==inter1(1))|(inter4(1)==inter2(1)))
        NIP=0;
      elseif (inter1(1)>=0)&(inter2(1)>=0)
        NIP=0;
      else
        NIP=2;
        if inter1(1)>0
          inter1=inter2;
        end;
        if inter3(1)<0
          inter2=inter3;
        else
          inter2=inter4;
        end;
      end;
    end;
  end;
end;
Output=0;
switch NIP
case 0
  Output = 0;
case 2
  i=Gradient_Flag+1;
  if inter2(i)<inter1(i)
    temp=inter1;
    inter1=inter2;
    inter2=temp;
  end;
  if
(Upper(i)>inter1(i))&(Upper(i)>inter2(i))&(Lower(i)>inter1(i))&(Lower(i)>inter2(
i))
    Output=0;
  elseif
(Upper(i)<inter1(i))&(Upper(i)<inter2(i))&(Lower(i)<inter1(i))&(Lower(i)<inter2(
i))
    Output=0;
  elseif
(Upper(i)>inter1(i))&(Upper(i)<inter2(i))&(Lower(i)>inter1(i))&(Lower(i)<inter2(
i))
    Output=1;
  elseif
((Upper(i)>inter2(i))&(Lower(i)<inter1(i)))|((Upper(i)<inter1(i))&(Lower(i)>inter
2(i)))
    Output=(inter2(i)-inter1(i))/(abs(Upper(i)-Lower(i)));
  else
    if Upper(i)>Lower(i)
      if (Upper(i)>inter2(i))&(Lower(i)>inter1(i))
        Output=(inter2(i)-Lower(i))/(Upper(i)-Lower(i));
      end;
      if (Upper(i)<inter2(i))&(Lower(i)<inter1(i))
        Output=(Upper(i)-inter1(i))/(Upper(i)-Lower(i));
      end;
    else
      if (Lower(i)>inter2(i))&(Upper(i)>inter1(i))
        Output=(inter2(i)-Upper(i))/(Lower(i)-Upper(i));
      end;
      if (Lower(i)<inter2(i))&(Upper(i)<inter1(i))
        Output=(Lower(i)-inter1(i))/(Lower(i)-Upper(i));
      end;
    end;
  end;
case 4
  i=Gradient_Flag+1;
  if inter1(i)>inter2(i)
    temp=inter1;
    inter1=inter2;
    inter2=temp;
  end;
  if inter3(i)>inter4(i)
    temp=inter3;
    inter3=inter4;
    inter4=temp;
  end;
  if Lower(i)>Upper(i);
    temp=Upper;
    Upper=Lower;
    Lower=temp;
  end;
  if Lower(i)<inter1(i)
    if Upper(i)<inter1(i)
      Output=0;
    elseif Upper(i)<inter3(i)
      Output=(Upper(i)-inter1(i))/(Upper(i)-Lower(i));
    elseif Upper(i)<inter4(i)
      Output=(inter3(i)-inter1(i))/(Upper(i)-Lower(i));
    elseif Upper(i)<inter2(i)
      Output=((inter3(i)-inter1(i))+(Upper(i)-inter4(i)))/(Upper(i)-Lower(i));
    elseif Upper(i)>inter2(i)
      Output=((inter3(i)-inter1(i))+(inter2(i)-inter4(i)))/(Upper(i)-Lower(i));
    end;
  elseif Lower(i)<inter3(i)
    if Upper(i)<inter3(i)
      Output=1;
    elseif Upper(i)<inter4(i)
      Output=(inter3(i)-Lower(i))/(Upper(i)-Lower(i));
    elseif Upper(i)<inter2(i)
      Output=((inter3(i)-Lower(i))+(Upper(i)-inter4(i)))/(Upper(i)-Lower(i));
    elseif Upper(i)>inter2(i)
      Output=((inter3(i)-Lower(i))+(inter2(i)-inter4(i)))/(Upper(i)-Lower(i));
    end;
  elseif Lower(i)<inter4(i)
    if Upper(i)<inter4(i)
      Output=0;
    elseif Upper(i)<inter2(i)
      Output=(Upper(i)-inter4(i))/(Upper(i)-Lower(i));
    elseif Upper(i)>inter2(i)
```

```
      Output=(inter2(i)-inter4(i))/(Upper(i)-Lower(i));
    end;
  elseif Lower(i)<inter2(i)
    if Upper(i)<inter2(i)
      Output=(Upper(i)-Lower(i))/(Upper(i)-Lower(i));
    elseif Upper(i)>inter2(i)
      Output=(inter2(i)-Lower(i))/(Upper(i)-Lower(i));
    end;
  elseif Lower(i)>inter2(i)
    Output=0;
  end;
end;
sys=Output;
end;
```

7.2. Estimator Program

```
% This function estimates Attitude from Magnetic Field Measurments
%
%          ConSat Simulator
%       Ver 2.0 17th November 1998
%
% (C) Institute for Systems and Robotics
%        IST / Lisbon - Portugal
%             <RAC>
%

function [sys,x0,str,ts]=estimate(t,x,u,flag)

global I1 I2 I3;
global mstep;
global w1 w2 w3;
global q1 q2 q3 q4;
global P z dz; % Covariance Matrix and State Vector
global wo;
global failed_flag;
global estimator_algorithm;

switch flag

case 0
  rand('state',sum(100*clock));
  load est_data P; % Loads Initial State Covariance Matrix
  sys=[0 0 8 20 0 1 1];
  x0=[];
  w=[w1; w2; w3]; %Initial Values in Orbital Coodinates
  z=[w; q1; q2; q3; q4];
  D=dcm(z);
  w=w+D*[wo; 0; 0]; % Initial Value in Inertial Coodinates
  z=[w; q1; q2; q3; q4];
  % Add error term to initial state
  for i=1:7
    z(i)=z(i)*(1+(0*(2*rand-1)));
  end;
  for i=4:7
    if (z(i)>1)|(z(i)<(-1))
      z(i)=round(z(i));
    end;
  end;
  % Normalize the quarternions
  for i=4:7
    z(i)=z(i)/norm([z(4) z(5) z(6) z(7)]);
  end;
  dz=zeros(7,1);
  str=[];
  ts=[mstep 0];

case 3
  % Generate Interia Matrix and it's inverse
  I = [I1 0 0; 0 I2 0; 0 0 I3];
  Iinv = [(1/I1) 0 0; 0 (1/I2) 0; 0 0 (1/I3)];
  % Obtain Covarience Matrices from File
  load est_data Q R
  Bmeas=[u(1); u(2); u(3)]; % Current Magnetic Field Measurement (in SCS)
  Borb=[u(4); u(5); u(6)]; % Predicted Magnetic Field (in OCS)
  SSorb=[u(18); u(19); u(20)]; % Predicted Sun Vector (in OCS)
  if (u(13)==0)&(u(17)==0)
    SS_flag=0;
    SSmeas=[0; 0; 0];
  elseif (u(13)==1)&(u(17)==1)
    SS_flag=1;
    SSmeas=[(u(10)+u(14))/2; (u(11)+u(15))/2; (u(12)+u(16))/2];
  elseif (u(13)==1)&(u(17)==0)
    SS_flag=1;
    SSmeas=[u(10); u(11); u(12)];
  else
    SS_flag=1;
    SSmeas=[u(14); u(15); u(16)];
  end;
  na=[u(7); u(8); u(9)]; % Control Moments
  if ~SS_flag
    R=R(1:3,1:3);
  end;
  % Calculate Kalman Gain
  H=Innovation(z,Borb,SSorb,SS_flag);
  K=P*(H.')*((H*P*(H.')+R)^(-1));
  % State Update
  D=dcm(z);
  if SS_flag
    dz=dz+K*([(Bmeas-D*Borb); (SSmeas-D*SSorb)] - H*dz);
  else
    dz=dz+K*((Bmeas-D*Borb) - H*dz);
  end;
  q_old=z(4:7);
  z=z+dz;
  % Normalise Quaternions
  q=z(4:7);
  q=q./norm(q);
  z=[z(1:3); q];
  % Update Covarience
  H=Innovation(z,Borb,SSorb,SS_flag);
  P=(eye(7)-K*H)*P*((eye(7)-K*H).')+K*R*(K.');
  % Propergate State Vector
  znew=z;
  int_level=5; % Sets number of intergration steps
  for i=1:int_level % Numerical Intergration Loop
    Omega=[znew(1); znew(2); znew(3)];
    D=dcm(znew);
    ngg=3*(wo^2)*(I1-I3)*D(3,3)*[-D(2,3); D(1,3); 0];
    OmegaDot=(Iinv)*(ngg+na-cross(Omega,(I*Omega)));
    OmegaOrb=Omega-D*[wo; 0; 0]; % Modified to give wo about i not j
    OMEGA=0.5*[0 OmegaOrb(3) -OmegaOrb(2) OmegaOrb(1);...
         -OmegaOrb(3) 0 OmegaOrb(1) OmegaOrb(2);...
         OmegaOrb(2) -OmegaOrb(1) 0 OmegaOrb(3);...
         -OmegaOrb(1) -OmegaOrb(2) -OmegaOrb(3) 0];
    QDot=OMEGA*[znew(4); znew(5); znew(6); znew(7)];
    znew=[OmegaDot; QDot]*(mstep/int_level)+znew;
  end;
  % Propergate Covarience Matrix
  Atemp=Iinv*[0 (I2-I3)*z(3) (I2-I3)*z(2); (I3-I1)*z(3) 0 (I3-I1)*z(1); (I1-I2)*z(2)
(I1-I2)*z(1) 0];
  Btemp=6*(wo^2)*(I2-I3)*Iinv*[-D(3,3)*z(7)+D(2,3)*z(4) -
D(3,3)*z(6)+D(2,3)*z(5) -D(3,3)*z(5)-D(2,3)*z(6) -D(3,3)*z(4)-D(2,3)*z(7);...
       D(3,3)*z(6)-D(1,3)*z(4) -D(3,3)*z(7)-D(1,3)*z(5) D(3,3)*z(4)+D(1,3)*z(6)
-D(3,3)*z(5)+D(1,3)*z(7);...
       0 0 0 0];
  Ctemp=0.5*[z(7) -z(6) z(5); z(5) z(6) -z(4); -z(5) z(4) z(7); -z(4) -z(5) -z(6)];
  F=[Atemp Btemp; Ctemp OMEGA];
  clear *temp;
  phi=eye(7)+F*mstep;
  P=phi*P*(phi.')+Q;
  % LOOP INSERTED TO STABILISE ESTIMATOR DURING TUNING
  for i=1:7
    for j=1:7
      if P(i,j)>10
        if failed_flag==0
          disp(' ESTIMATOR FAILED - AUTO SHUTDOWN ');
        end;
        estimator_algorithm=0;
        failed_flag=1;
      end;
    end;
  end;
  % Propergate change in quaternion (add so that quaternions are
normalised)
  dz=[zeros(3,1); (OMEGA*(q_old*(q_old.')*dz(4:7)))];
  z=znew;
  sys=[z(1) z(2) z(3) z(4) z(5) z(6) z(7) estimator_algorithm];
end;

function H=Innovation(z,Borb,SSorb,SS_flag) % Computes H matrix given
current state
switch SS_flag
case 0
  h1=2*[z(4) z(5) z(6); z(5) -z(4) z(7); z(6) -z(7) -z(4)]*Borb;
  h2=2*[-z(5) z(4) -z(7); z(4) z(5) z(6); z(7) z(6) -z(5)]*Borb;
  h3=2*[-z(6) z(7) z(4); -z(7) -z(6) z(5); z(4) z(5) z(6)]*Borb;
  h4=2*[z(7) z(6) -z(5); -z(6) z(7) z(4); z(5) -z(4) z(7)]*Borb;
  H=[zeros(3,3) h1 h2 h3 h4];
case 1
  temp=2*[z(4) z(5) z(6); z(5) -z(4) z(7); z(6) -z(7) -z(4)];
  h1=[temp*Borb; temp*SSorb];
  temp=2*[-z(5) z(4) -z(7); z(4) z(5) z(6); z(7) z(6) -z(5)];
  h2=[temp*Borb; temp*SSorb];
  temp=2*[-z(6) z(7) z(4); -z(7) -z(6) z(5); z(4) z(5) z(6)];
  h3=[temp*Borb; temp*SSorb];
  temp=2*[z(7) z(6) -z(5); -z(6) z(7) z(4); z(5) -z(4) z(7)];
  h4=[temp*Borb; temp*SSorb];
  H=[zeros(6,3) h1 h2 h3 h4];
end; %case
% End of Function Innovation

function D=dcm(z) % Computes the Directional Cosine Matrix, D
D=[z(4)^2-z(5)^2-z(6)^2+z(7)^2 2*(z(4)*z(5)+z(6)*z(7)) 2*(z(4)*z(6)-
z(5)*z(7));...
     2*(z(4)*z(5)-z(6)*z(7)) -z(4)^2+z(5)^2-z(6)^2+z(7)^2
2*(z(5)*z(6)+z(4)*z(7));...
     2*(z(4)*z(6)+z(5)*z(7)) 2*(z(5)*z(6)-z(4)*z(7)) -z(4)^2-
z(5)^2+z(6)^2+z(7)^2];
% Eng of Fuction dcm
```

7.3. Sun Sensor Processor

```
% given the Sun Sensor Reading
%
%           ConSat Simulator
%       Ver 2.0 17th November 1998
%
% (C) Institute for Systems and Robotics
%        IST / Lisbon - Portugal
%             <RAC>
%
% Simulink S-Function
%
% Inputs  u(1)      Sun Sensor Reading Channel 1
%         u(2)      Sun Sensor reading Channel 2
%         u(3-5)    Sun's Position in OCS (modelled)
%         u(6-14)   Attitude Matrix from Previous Iteration
%         u(15)     Switch between sensor 1 and 2
%
% Outputs sys(1-3)  Sun's Position in SCS
%         sys(4)    Flag - Zero indicates no measurement possible
%                     - One indicates measurment available
%
% Fuction Created 28/4/99

function [sys,x0,str,ts]=Sun_sensor_est(t,x,u,flag)

global mstep;

switch flag

case 0
   sys=[0 0 4 15 0 1 1];
   x0=[];
   str=[];
   ts=[mstep 0];

case 3
   u(1)=(u(1)/5);
   u(2)=(u(2)/5);
   if (u(1)<0.1)|(u(2)<0.1) % If signal below 0.1 then no output
      sys=[0 0 0 0];
   else
      phi_1=acos((2*u(2)-u(1))/(sqrt(3)*sin(acos(u(1)))));
      if u(15)==2
         Sun_Vector_1=[0.5*u(1)-(sqrt(3)/2)*sin(acos(u(1)))*cos(phi_1)
                       (sqrt(3)/2)*u(1)+0.5*sin(acos(u(1)))*cos(phi_1)
                       sin(acos(u(1)))*sin(phi_1)];
      end;
      if u(15)==1
         Sun_Vector_1=[-(sqrt(3)/2)*u(1)-0.5*sin(acos(u(1)))*cos(phi_1)
                        0.5*u(1)-(sqrt(3)/2)*sin(acos(u(1)))*cos(phi_1)
                        sin(acos(u(1)))*sin(phi_1)];
      end;
      Sun_Vector_2=[1 0 0; 0 1 0; 0 0 -1]*Sun_Vector_1;
      Att=[u(6) u(9) u(12); u(7) u(10) u(13); u(8) u(11) u(14)];
      Est_Sun_Vector=Att*[u(3); u(4); u(5)];
      if norm(Est_Sun_Vector)==0
         sys=[0 0 0 0];
      else
         Est_Sun_Vector=Est_Sun_Vector./norm(Est_Sun_Vector);
         Distance_1=abs(Sun_Vector_1(3)-Est_Sun_Vector(3));
         Distance_2=abs(Sun_Vector_2(3)-Est_Sun_Vector(3));
         if Distance_1<Distance_2
            sys=[(Sun_Vector_1).' 1];
         else
            sys=[(Sun_Vector_2).' 1];
         end; % if
      end; % if
   end; % if
end; %case
```

7.4. Genetic Tuner

```
function genetic_tune

global failed_flag;
global chromos fchromos copy1 copy2;
global fit n_pop n_points irepl p_mut1 nrepl max_power;

% get current path
CurrentPath=cd;
% add ConSat Simulator path to Matlab search Path
addpath(CurrentPath);
addpath(strcat(CurrentPath,'\Interface'));
addpath(strcat(CurrentPath,'\Orbit'));
addpath(strcat(CurrentPath,'\Attitude&Control'));
addpath(strcat(CurrentPath,'\Data'));
addpath(strcat(CurrentPath,'\Estimator'));

p_mut1 = 0.005;    % Probablity of Mutation
n_pop = 10;       % Total Number of Population
nrepl = n_pop;     % Number of Reproductions
ngens = 25;       % Number of Generations
p_crossover = 0.6;
n_points = 19;     % Number of covariance parameters
max_power = 12;     % Sets the maximum power that a covarence can have

% Generate Initial Population - chromos

% First load previous solution
load solution pre_chromos
chromos(:,1)=pre_chromos;

% Generate random set of chromosones
rand('state',sum(100*clock));
for i=2:n_pop
  for j=1:n_points
    chromos(j,i)=rand*10^(-rand*max_power);
  end;
end;
disp(' ');
disp(' Initialisation Complete ');

cost_best=realmax;
% START MAIN LOOP
for igen=1:ngens
  disp(' ');
  disp(sprintf(' Starting Generation %i',igen));

  % EVALUATE
  for ident=1:n_pop
    failed_flag=0;
    generate_qpr(chromos(:,ident));  % Generates the Estimator data
    batch_tune(ident);     % Runs the simulator
    if failed_flag==1
      cost=10e5;
    else
      cost=results_anaylsis(ident);    % Evaluates Results
    end;
    if cost<cost_best
      temp=chromos(:,ident);
      save tune_results cost temp
      cost_best=cost;
      disp(' ');
      disp(' RESULTS SAVED ');
      disp(' ');
      disp(sprintf(' Best Cost = %.4f',cost_best));
    end;
    fit(ident)=1/cost;
  end; % for
  [fitbest,id_best]=max(fit);
  disp(' ');
  disp(sprintf(' Top Fitness = %.4f',fitbest));

  % POPULATION
  fitavg=mean(fit);
  fitrange=max(fit)-min(fit);     % CHK if this is needed
  fitr2=fitbest-fitavg;
  fitbase=0.5;
  for j=1:n_pop
    fitc1 = (4*(fit(j)-fitavg)^2)/(fitr2^2);
    if fit(j)<fitavg
      fitc2=0;
    else
      fitc2 = (5*(fit(j)-fitavg)^2)/(fitr2^2);
    end;
    fit(j)=fitbase+fitc1+fitc2;
  end;
  afitmax=max(fit);      % CHK if this is needed
  afitmin=min(fit);     % CHK if this is needed
  afitavg=mean(fit);     % CHK if this is needed
  sumfscale=sum(fit);

  % REPRODUCTION
  isel=0;
  irepl=0;
  duplicate(isel,id_best);
  ir=0;
  while (ir<(50*nrepl))&(irepl<nrepl)
    ir=ir+1;
    ran_number=rand;
    if ran_number<p_crossover
      isel=2;
    else
      isel=1;
    end;
    [ipar1 ipar2]=select(isel,sumfscale);
    if isel==2
      crossover(ipar1,ipar2);
    else
      mutate(ipar1);
    end;
    duplicate(isel,id_best);
  end;
  file = sprintf('tunning_results\\chromos%i',igen);
  save(file,'chromos','fit');
  repopulate;
```

```
  if igen==80
    p_mut1=2*p_mutl;
    p_crossover=0.6;
  end;
  disp(' TUNNING COMPLETE !! ');
end;

% FUNCTION SELECTS PARENTS FOR REPRODUCTION
% isel=2, CROSSOVER
% isel=1, MUTATION
function [ipar1,ipar2]=select(isel,sumfscale)
global fit n_pop;
ipar1=0;
ipar2=0;
rannum=rand*(sumfscale-fit(n_pop));
sumfit=0;
for j=1:n_pop
  sumfit=sumfit+fit(j);
  if (sumfit>=rannum)&(ipar1==0)
    ipar1=j;
  end;
end;
if isel>1
  sumfit=0;
  for j=1:n_pop
    sumfit=sumfit+fit(j);
    if (sumfit>=rannum)&(ipar2==0)&(ipar1~=j)
      ipar2=j;
    end;
  end;
end;


function mutate(ident)
global chromos n_points copy1 p_mut1 max_power
for ip=1:n_points
  a1ran=rand;
  if (a1ran<p_mut1)
    copy1(ip)=rand*10^(-rand*max_power);
  else
    copy1(ip)=chromos(ip,ident);
  end;
end;

function crossover(ipar1,ipar2)
global n_points;
global copy1 copy2 chromos

iposs=realmax;
while iposs>n_points
  ipos1=round(rand*n_points+0.5);
  ipos2=round(rand*n_points+0.5);
  iposs=min(ipos1,ipos2);
  iposf=max(ipos1,ipos2);
  if iposf>n_points
    iposf=n_points;
  end; % if
end; % while
icpt=1;
icpar1=ipar1;
icpar2=ipar2;
for i=1:n_points
  copy1(i)=chromos(i,icpar1);
  copy2(i)=chromos(i,icpar2);
  icpt=icpt+1;
  if icpt>iposs
    icpar1=ipar2;
    icpar2=ipar1;
  end;
  if icpt>iposf
    icpar1=ipar1;
    icpar2=ipar2;
  end;
end;


function duplicate(isel,id_best)
global chromos fchromos copy1 copy2
global n_points irepl nrepl
if isel==0
  irepl=2;
  fchromos(:,1)=chromos(:,id_best);
  fchromos(:,2)=chromos(:,id_best).*(1+0.5*((2*rand(n_points,1))-1));
else
  idiff=0;
  for idchk=1:irepl
    flag=0;
    ip=0;
    while (flag==0)&(ip<n_points)
      ip=ip+1;
      if abs(copy1(ip)-fchromos(ip,idchk))>eps
        idiff=idiff+1;
        flag=1;
      end; % if
    end; % while
  end; % for
  if (idiff==irepl)
    irepl=irepl+1;
    fchromos(:,irepl)=(copy1.');
  end;
  if (isel~=1)&(irepl~=nrepl)
    idiff=0;
    for idchk=1:irepl
      flag=0;
      ip=0;
      while (flag==0)&(ip<n_points)
        ip=ip+1;
        if abs(copy2(ip)-fchromos(ip,idchk))>eps
          idiff=idiff+1;
          flag=1;
        end; % if
      end; % while
    end; % for
    if (idiff==irepl)
      irepl=irepl+1;
      fchromos(:,irepl)=(copy2.');
    end;  % if
  end;  % if
end;


function repopulate
global chromos fchromos n_points n_pop
for j=1:n_pop
  for i=1:n_points
    chromos(i,j)=fchromos(i,j);
  end;
end;


function generate_qpr(qpr_vec)
load estimator/est_data P
Q = [qpr_vec(1)  qpr_vec(2)  qpr_vec(3)  qpr_vec(10) qpr_vec(10)
qpr_vec(11) qpr_vec(11)
    qpr_vec(2)  qpr_vec(1)  qpr_vec(3)  qpr_vec(10) qpr_vec(10) qpr_vec(11)
qpr_vec(11)
    qpr_vec(3)  qpr_vec(3)  qpr_vec(4)  qpr_vec(12) qpr_vec(12) qpr_vec(13)
qpr_vec(13)
    qpr_vec(10) qpr_vec(10) qpr_vec(12) qpr_vec(5)  qpr_vec(7)  qpr_vec(9)
qpr_vec(9)
    qpr_vec(10) qpr_vec(10) qpr_vec(12) qpr_vec(7)  qpr_vec(5)  qpr_vec(9)
qpr_vec(9)
    qpr_vec(11) qpr_vec(11) qpr_vec(13) qpr_vec(9)  qpr_vec(9)  qpr_vec(6)
qpr_vec(8)
    qpr_vec(11) qpr_vec(11) qpr_vec(13) qpr_vec(9)  qpr_vec(9)  qpr_vec(8)
qpr_vec(6)];
R = [qpr_vec(14) qpr_vec(16) qpr_vec(16) 0.0        0.0        0.0
    qpr_vec(16) qpr_vec(15) qpr_vec(17) 0.0        0.0        0.0
    qpr_vec(16) qpr_vec(17) qpr_vec(15) 0.0        0.0        0.0
    0.0        0.0        0.0        qpr_vec(18) qpr_vec(19) qpr_vec(19)
    0.0        0.0        0.0        qpr_vec(19) qpr_vec(18) qpr_vec(19)
    0.0        0.0        0.0        qpr_vec(19) qpr_vec(19) qpr_vec(18)];
save estimator/est_data Q P R


function batch_tune(ident)

global BATCH_RUN;
global ano mes dia hour min sec tfinal;
global mstep; % simulation step
global w1 w2 w3;
global q1 q2 q3 q4;
global referencia;
global control_algorithm % <PTS>
global Kw               % <PTS>
global K h_gain g_gain epsilon;
global wo;
global tfinal;
global estimator_algorithm;
global predict_all
global record

BATCH_RUN=1;
duracao = 5.0;
record = 0;

disp(' ');
disp(sprintf(' Starting Simulation %i',ident));
initsat;
ano = 1997;
mes = 1;
dia = 1;
mstep = 1;
wo = 0.0010385;
tfinal = 2*pi/wo * duracao;

Roll  = 7.0*pi/180;
Pitch = 5.0*pi/180;
Yaw   = 0.0;

q4=sqrt((1+cos(Pitch))*(1+cos(Yaw)*cos(Roll)-sin(Yaw)*sin(Roll)))/2;
q1=(sin(Pitch)*(cos(Yaw)+cos(Roll)))/(4*q4);
q2=(sin(Pitch)*(sin(Roll)-sin(Yaw)))/(4*q4);
q3=((1+cos(Pitch))*(cos(Yaw)*sin(Roll)+sin(Yaw)*cos(Roll)))/(4*q4);
```

```
clear Roll Pitch Yaw

referencia = 0.02;

w1 = 0.0005;
w2 = 0;
w3 = 0.02;

hour = 1;
min  = 23;
sec  = 22;

% sim_orbits=orbit_available(ano,mes,dia,hour,min,sec,duracao);

estimator_algorithm = 1;
control_algorithm = 10;

Kw = [1 1 1];
K=0;
h_gain=0;
g_gain=0;
epsilon=0;

predict_all = 1;

% simulate
disp(' Calculating attitude')
drawnow;
pause(0.5);
attitude;

disp(' Simulation Complete');


function sim_orbits=orbit_available(ano,mes,dia,hour,min,sec,duracao)
RomuloPath='H:\nãoapa~1\simsat';
load(strcat(RomuloPath,'\data\orbs'));
data=[ano mes dia hour min sec];
sim_orbits=1;
i=1;
while (sim_orbits==1)&(i<=size(orbs,1))
  if (orbs(i,1:6)==data)&(duracao<=orbs(i,7))
    folder=sprintf('\\data\\orbits\\orb%i\\',i);
    copyfile(strcat(RomuloPath,folder,'oribt.mat'),'data\orbit.mat');

copyfile(strcat(RomuloPath,folder,'magnetic4th.mat'),'data\magnetic4th.mat');
    copyfile(strcat(RomuloPath,folder,'sol.mat'),'data\sol.mat');
    copyfile(strcat(RomuloPath,folder,'tamterra.mat'),'data\tamterra.mat');
    sim_orbits=0;
  end;
  i=i+1;
end;
```

```
function cost=results_anaylsis(ident)

load est_quaternion
load quaternions
load angular
load est_angular

q1=est_quaternion;
w1=est_angular;
clear est_quaternion est_angular;

kx_real=2*(q(2,:).*q(4,:)-q(3,:).*q(5,:));
ky_real=2*(q(3,:).*q(4,:)+q(2,:).*q(5,:));
kz_real=-q(2,:).^2-q(3,:).^2+q(4,:).^2+q(5,:).^2;
kx_est=2*(q1(2,:).*q1(4,:)-q1(3,:).*q1(5,:));
ky_est=2*(q1(3,:).*q1(4,:)+q1(2,:).*q1(5,:));
kz_est=-q1(2,:).^2-q1(3,:).^2+q1(4,:).^2+q1(5,:).^2;
temp=(kx_real.*kx_est)+(ky_real.*ky_est)+(kz_real.*kz_est);
for j=1:size(temp,2)
  if abs(temp(j))>1
    temp(j)=floor(temp(j));
  end;
end;
Pointing_Error=(180/pi)*acos(temp);
Spin_Rate_Error=100*abs((w(4,:)-w1(4,:))./w(4,:));
for j=1:size(Spin_Rate_Error,2)
  if w(4,j)<=0.0001
    Spin_Rate_Error(j)=0;
  end;
end;
NoPoints=size(Pointing_Error,2);
Mean_Pointing_Error=mean(Pointing_Error(round(0.25*NoPoints):NoPoints))
;
STD_Pointing_Error=std(Pointing_Error(round(0.25*NoPoints):NoPoints));
Mean_Spin_Rate_Error=mean(Spin_Rate_Error(round(0.25*NoPoints):NoPo
ints));
STD_Spin_Rate_Error=std(Spin_Rate_Error(round(0.25*NoPoints):NoPoints
));

res=[Mean_Pointing_Error STD_Pointing_Error Mean_Spin_Rate_Error ...
    STD_Spin_Rate_Error];

disp(sprintf(' Estimator Results for Simulation %i',ident));
disp(sprintf(' Mean Pointing Error (degrees)       : %.4f',res(1)));
disp(sprintf(' STD of Pointing Error (degrees)     : %.4f',res(2)));
disp(sprintf(' Mean Error in Spin Rate (percent)   : %.4f%%',res(3)));
disp(sprintf(' STD of Error in Spin Rate (percent) : %.4f%%',res(4)));

cost=sum(res);
```

Appendix B

SPIN TEST A

For all simulations Ω=[0.001037, 0, 0.02]rads/s, Ω_{REF}=0.02rads/s:

Simulation	Year	Month	Day	Hour	Minute	Second	Pitch	Roll	Yaw
1	1997	1	1	1	23	22	60	301.7	0
2	1997	1	1	8	28	8	60	7.0	0
3	1997	1	1	19	30	58	60	245.6	0
4	1997	1	1	0	14	12	60	136.6	0
5	1997	1	1	3	20	0	60	299.5	0
6	1997	1	1	4	51	59	60	181.0	0
7	1997	1	1	4	46	10	60	255.0	0
8	1997	1	1	14	29	28	60	154.0	0
9	1997	1	1	6	31	57	60	109.7	0
10	1997	1	1	4	46	18	60	68.3	0

SPIN TEST B

For all simulations Ω=[0.001037, 0, 0.02]rads/s, Ω_{REF}=0.02rads/s:

Simulation	Year	Month	Day	Hour	Minute	Second	Pitch	Roll	Yaw
1	1997	1	1	1	23	22	5	301.7	0
2	1997	1	10	8	28	8	5	7.0	0
3	1997	1	21	19	30	58	5	245.6	0
4	1997	2	2	0	14	12	5	136.6	0
5	1997	2	13	3	20	0	5	299.5	0
6	1997	2	25	4	51	59	5	181.0	0
7	1997	3	4	4	46	10	5	255.0	0
8	1997	3	14	14	29	28	5	154.0	0
9	1997	3	21	6	31	57	5	109.7	0
10	1997	3	2	4	46	18	5	68.3	0

SPIN TEST C

For all simulations Ω=[0, 0, 0]rads/s, Ω_{REF}=0.0rads/s:

Simulation	Year	Month	Day	Hour	Minute	Second	Pitch	Roll	Yaw
1	1997	1	1	1	23	22	0	0	0
2	1997	1	1	8	28	8	0	0	0
3	1997	1	1	19	30	58	0	0	0
4	1997	1	1	0	14	12	0	0	0
5	1997	1	1	3	20	0	0	0	0
6	1997	1	1	4	51	59	0	0	0
7	1997	1	1	4	46	10	0	0	0
8	1997	1	1	14	29	28	0	0	0
9	1997	1	1	6	31	57	0	0	0
10	1997	1	1	4	46	18	0	0	0

Controller Parameters

PREDICTIVE CONTROLLER

Cost Function Gain = [1 1 1],

Predict All = 1.

ENERGY CONTROLLER

$\varepsilon = 1.0 \times 10^5$,

$h\_gain = 7.0 \times 10^7$,

$g\_gain = 1.0 \times 10^5$.

ALPHA-BETA CONTROLLER

$K = 5.0 \times 10^4$.